Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

République de Côte d'Ivoire
Union - Discipline - Travail

Institut National Polytechnique
Félix HOUPHOUËT-BOIGNY



N° d'ordre : 084/2021

**THÈSE UNIQUE**

Pour l'obtention du grade de

**Docteur de l'Institut National Polytechnique Félix Houphouët Boigny de Yamoussoukro**

Mention : Sciences Informatiques

Spécialité : Informatique

GNIMASSOUN Ban'délé Jean Edgard

# Ordonnancement d'Applications Parallèles (Workflows Scientifiques) sur les ressources IaaS du Cloud Computing

Soutenue le 17/03/2021 devant le jury :

| | | | |
|---|---|---|---|
| M. ZOUEU Thouakesseh Jérémie | Professeur Titulaire | INP-HB | Président du jury |
| M. ADOU Kablan Jérôme | Professeur Titulaire | UFHB | Rapporteur |
| M. AKA Boko | Professeur Titulaire | UNA | Rapporteur |
| M. KONAN Kouadio Fransisco | Maître de Conférences | ENS | Examinateur |
| M. OUMTANAGA Souleymane | Professeur Titulaire | INP-HB | Directeur de thèse |

# Dédicace

*A ma famille.*

# Remerciements

L'aboutissement de cette thèse est le fruit de longues et périlleuses épreuves qui, sans le soutien inconditionnel et l'aide de certaines personnes, aurait pu ne pas aboutir. Je tiens donc à remercier chacune de ces personnes :

— Monsieur Souleymane OUMTANAGA, Professeur Titulaire à l'INP-HB, qui a accepté de diriger cette thèse mais aussi de me transmettre des valeurs telles que la rigueur dans le travail, le dépassement de soi, l'amour du prochain. Ses conseils et orientations ont à chaque fois été un déclic pour l'aboutissement de cette thèse, il a toujours été disponible ;

— Messieurs les membres du jury, qui malgré leurs occupations professionnelles ont accepté de juger ce travail de recherche. Monsieur Boko AKA, Professeur Titulaire à l'UNA, pour sa disponibilité, ses conseils et sa lecture minutieuse du manuscrit. Il m'a donné le savoir de base depuis la Licence et demeure pour moi un modèle dans le travail. Monsieur Kablan ADOU, Professeur Titulaire à l'UFHB, pour ses orientations et qui a lu attentivement le manuscrit afin d'améliorer celui-ci. Ils ont été tous deux d'une aide précieuse lors des échanges pour corriger les erreurs tant dans la forme que dans le fond. Monsieur Fransisco KONAN, Maître de Conférences à l'ENS, pour ses remarques et suggestions qu'il m'a adressé lors de la soutenance pour améliorer mon travail. Monsieur Jérémie ZOUEU, Professeur Titulaire à l'INP-HB, qui a accepté de présider le jury et a explique avec des mots simples ma thèse au public non spécialiste.

— Monsieur Benjamin YAO, Professeur Titulaire à l'INP-HB et Directeur de l'Ecole Doctorale Polytechnique de l'INP-HB, Monsieur Jérémie ZOUEU, Professeur Titulaire à l'INP-HB et Directeur de l'UMRI 78 et Messieurs Doudjo SORO et ABRO Koutouan Désiré, respectivement Maître de Conférences et Attaché de Recherche à l'INP-HB, qui par leurs actions contribuent à la promotion de la recherche fondamentale et appliquée en Côte d'Ivoire ;

— Monsieur Michel BABRI, Professeur Titulaire à l'INP-HB et Directeur du LARIT, qui m'a accompagné tant scientifiquement que moralement. Il est toujours disponible et est à l'écoute de nos préoccupations ;

— Monsieur Tchimou N'TAKPE, Maître Assistant à l'UNA, pour m'avoir donné une approche méthodique et éclairé lors de mes premiers pas dans la recherche. Monsieur Frédéric SUTER, Professeur Titulaire à CC-IN2P3 France, qui avec Monsieur Tchimou N'TAKPE, ont été tous deux d'une aide précieuse lors des différentes contributions ;

— Messieurs, Joël ADEPO, Assistant à l'UVCI, pour m'avoir orienté sur certains aspects théoriques ; Hervé DIEDIE, Maître Assistant à l'UPGC, pour les discussions et la correction de ce manuscrit ; Georges ANOH, Assistant à l'UVCI ; Hamidja KAMAGATE,

Assistant à l'ESATIC ; Nouho OUATTARA Assistant à UAO ; Armel KEUPONDJO Assistant à l'INP-HB et Philip KIMENYI, pour leurs encouragements et conseils ;

— tous les membres de l'équipe IREF, en particulier Ferdinand ATTA, qui m'a fourni une aide très précieuse dans l'avancement de nos travaux ;

— Monsieur Karim SIDIBE, pour sa lecture approfondie du manuscrit afin de faire des critiques et suggestions objectives ;

— Monsieur Carlos IRIÉ, enseignant de français au Lycée Moderne de Bouaflé, pour la correction des fautes de grammaire et d'orthographe ;

— Monsieur et Madame GNIMASSOUN, mes parents biologiques, pour leurs soutiens spirituel, moral et financier. Mes sœurs et frères, Emma, Prisca, Nicole, Sandra, Igor et Ronald qui m'ont toujours motivé à ne jamais abandonné ;

— toutes les familles (POÉ, OGAH, ADEKPEDJOU, ...) qui m'ont soutenu durant mon cursus scolaire et universitaire.

# Résumé

Aujourd'hui de nombreuses applications scientifiques nécessitent d'être parallélisées. Cette parallélisation permet d'exécuter simultanément plusieurs tâches indépendantes d'une même application sur des cœurs de processeurs différents. Ainsi on peut considérablement réduire le temps d'exécution (makespan) d'une telle application par rapport à sa version séquentielle (où toutes les tâches s'exécutent sur un même cœur de processeur l'une après l'autre). Les applications parallélisables sont aussi diverses que variées. On trouve de nombreuses applications de calcul scientifique dans les domaines de la recherche de médicaments, de la simulation nucléaire, de la simulation des propriétés mécaniques des engins, de la recherche astronomique, de la simulation bancaire, du traitement d'images, etc.

De nombreuses applications sont modélisables sous forme de *workflows* scientifiques, c'est-à-dire que ces applications peuvent être représentées par des graphes orientés acycliques où les nœuds représentent les différentes tâches de l'application à exécuter. Les arcs représentent, quant à eux, les contraintes de dépendances entre les tâches (une tâche ne peut commencer son exécution que lorsque toutes les tâches parentes de cette tâche ont terminé leur exécution). Dans les *workflows* scientifiques, certaines tâches chargent des fichiers en entrée et/ou produisent des fichiers en sortie lors de leurs exécutions. Un fichier en sortie d'une tâche peut être réutilisé en entrée d'une autre tâche. Il se pose donc la problématique de l'ordonnancement de ces tâches et de ces fichiers afin de réduire l'impact des transferts de fichiers sur le réseau et de minimiser le makespan et/ou le coût d'exécution de l'application parallèle, surtout lorsque les différents cœurs de processeurs sollicités ne se trouvent pas nécessairement sur la même machine.

Les *workflows* scientifiques s'exécutaient principalement sur les grappes de calcul (*clusters*). Aujourd'hui les offres IaaS (*Infrastrure as a Services*) des fournisseurs de cloud computing représentent un nouvel environnement d'exécution de ce type d'applications parallèles. Toutefois, cette migration vers le *cloud computing* entraîne de nouveaux défis d'ordonnancement à relever. En effet, puisque sur les plateformes IaaS, les ressources paraissent virtuellement illimitées et que chaque utilisateur qui souhaite y exécuter son application est facturé en fonction des ressources utilisées, le problème majeur qui se pose est comment trouver un bon compromis entre le temps d'exécution de son application et le coût d'utilisation des ressources.

Nous proposons ainsi une solution pour ce problème d'optimisation bi-critères de recherche de compromis entre le makespan des *workflows* scientifiques et le coût d'utilisation des ressources du cloud, en menant cette étude en plusieurs étapes. La première étape consiste à proposer un algorithme d'optimisation monocritère qui tente de minimiser le makespan des *workflows* lorsque les ressources à utiliser sont fixées. Cet algorithme a été comparé au très populaire algorithme HEFT.

Ensuite, dans une autre étude nous montrons que pour un même nombre total de cœurs de processeurs souhaité, utiliser moins de machines virtuelles contenant de nombreux cœurs de processeurs chacune est plus bénéfique qu'utiliser plus de machines virtuelles contenant chacune moins de cœurs de processeurs. En effet, cela permet de réduire considérablement le makespan des *workflows* scientifiques.

Nous avons aussi proposé une méthode plus rapide de génération d'un front de Pareto à partir de l'algorithme proposé précédemment et en lançant les simulations sur un nombre réduit de plateformes par rapport à l'approche concurrente qui est l'algorithme MOHEFT. MOHEFT étant le meilleur algorithme de la littérature pour la résolution du problème bicritère sus-énoncé. Il faut rappeler qu'un front de Pareto représente les points optimaux obtenus à l'issu de l'exécution d'un algorithme d'optimisation multicritères. Les résultats obtenus montrent que l'approche proposée donne de meilleurs fronts de Pareto par rapport à MOHEFT en plus de donner plus de points (donc plus de choix à l'utilisateur) dans ces fronts de Pareto par rapport à ceux de MOHEFT, et de sortir les fronts de Pareto en des temps très inférieurs à ceux de MOHEFT. Enfin dans une dernière contribution, nous avons proposé une procédure par recherche dichotomique pour suggérer moins de points que ceux qui sont sur le front de Pareto complet afin de faciliter le choix à l'utilisateur. Cette approche élimine des points du front de Pareto qui ne sont pas intéressants en pratique, bien qu'étant des points optimaux théoriquement car non dominés par les autres points.

*Mots clés* : ordonnancement workflows scientifiques, makespan, front de Pareto, machines virtuelles, multi-cœur, workflows à forte intensité de données, cloud computing, IaaS.

# Abstract

**"Scheduling of Parallel Applications (Scientific Workflows) on Cloud Computing IaaS resources."**

Nowadays, many scientific applications need to be parallelized. This parallelization allows to complete simultaneously several independent tasks with the same application on different processor cores. Thus, the execution time (makespan) of such a application can be shortened compared to its sequential version (where all the tasks are executed on the same processor core one after the other). Parallelizable applications are as diverse and varied. There are many scientific computing application in the fields of drug research, nuclear simulation, simulation of mechanical properties of machines, astronomical research, banking simulation, image processing, etc.

Many applications can be modeled as scientific workflows, i.e. They can be represented by Directed Acyclic Graphs (DAG) where the nodes represent the different tasks of the application to be completed. The arcs represent the dependency constraints between the tasks (a task can only start its execution when all its parent tasks have finished their execution). In scientific workflows, some tasks load input files and/or produce output files during their execution. An output file of a task can be reused as input for another task. This raises the issue of scheduling these tasks and files in order to reduce the impact of file transfers on the network and to minimize the makespan and/or the cost of completing the parallel software, especially when the different processor cores required are not necessarily on the same machine.

Scientific workflows used to run mainly on clusters. Today, the IaaS (Infrastructure as a Services) offers of cloud computing providers represent a new execution environment for this type of parallel applications. However, this migration to cloud computing brings new scheduling challenges. Indeed, since on IaaS platforms, resources appear virtually unlimited and each user who wishes to run his application is charged according to the resources used, the major problem that arises is how to find a good compromise between the execution time of his application and the cost of using the resources.

We thus propose a solution for this bi-criteria optimization problem of finding a trade-off between the makespan of scientific workflows and the cost of using cloud resources, by conducting this study in several steps. The first step is to suggest a single-criteria optimization algorithm that attempts to minimize the makespan of workflows when the resources to be used are fixed. This algorithm has been compared to the very popular HEFT algorithm.

Then, in another study we showed that for the same total number of processor cores, using fewer virtual machines containing many processor cores each one is more beneficial than using more virtual machines each containing fewer processor cores. This is because it significantly reduces the makespan of scientific workflows.

We also suggest a faster method of generating a Pareto front from the previously suggested algorithm and running the simulations on fewer platforms than the competing approach which is the MOHEFT algorithm. MOHEFT being the best algorithm in the literature for solving the above mentioned bicriteria problem. It should be recalled that a Pareto front represents the optimal points obtained from the execution of a multi-criteria optimization algorithm. The outcomes show that the suggested approach gives better Pareto fronts compared to MOHEFT in addition to giving more points (thus more choices to the user) in these Pareto fronts compared to those of MOHEFT, and to get out the Pareto fronts in much less time than MOHEFT. Finally, in a last contribution, we suggested a dichotomous search procedure to suggest fewer points than those on the full Pareto front in order to facilitate the user's choice. This approach eliminates points from the Pareto front that are not interesting in practice, although they are theoretically optimal points because they are not dominated by the other points.

*Keywords* : scientifics workflows scheduling, makespan, Pareto front, virtuals machines, multi-cores, data-intensives workflows, cloud computing, IaaS.

# Table des matières

# Liste des figures

# Liste des tableaux

# Introduction générale

## Contexte et problématique

Les workflows scientifiques sont couramment utilisés pour modéliser une application scientifique [1–3]. Ils décrivent un ensemble de traitement qui permettent d'analyser les données de manière structurée et distribuée et ont été utilisés pour réaliser des avancées scientifiques significatives dans divers domaines tels que la biologie [4, 5], la physique [6–8], la médecine [9, 10], l'astronomie [11, 12], etc. Leur importance est mise en évidence à l'ère du Big Data [13, 14]. Le Big Data est un concept permettant de stocker et d'analyser un nombre indicible d'informations sur une base numérique. Les données à traiter deviennent de plus en plus volumineuses et de grande taille, rendant ces applications des "data-intensives" [15–17]. Une application data-intensive est une classe d'applications parallèles qui utilisent une approche parallèle des données pour traiter de gros volumes de données, généralement de taille téraoctet ou pétaoctet [18–20].

L'émergence du cloud computing a apporté de nombreux avantages au déploiement de workflows scientifiques à grande échelle [21]. En particulier, les ressources du cloud computing en tant que service offrent une infrastructure facilement accessible, flexible et évolutive pour le déploiement des workflows scientifiques [22]. L'infrastructure en tant que service (*Infrastructure as a Service* : IaaS) est une forme de cloud computing offrant des ressources informatiques au sein d'un environnement virtualisé (le cloud computing) par le biais d'internet ou d'une connexion spécialisée. L'IaaS est l'une des trois principales catégories de services de cloud computing, au même titre que le logiciel en tant que service (*Software as a Service* : SaaS) et la plateforme en tant que service (*Platform as a Service* : PaaS) [23]. Les fournisseurs de cloud IaaS offrent la possibilité d'exécuter des workflows sur leurs infrastructures en louant des ressources de calcul "barre-métal", des machines virtuelles (VM), des ressources de stockage, et bien d'autres. Cela permet de conditionner, de déployer facilement les workflows et de permettre aux systèmes de gestion de workflow d'accéder à un ensemble virtuellement infini de machines pouvant être acquises et libérées

en fonction du besoin. Ces ressources sont facturées en fonction de l'utilisation [24, 25]. Ainsi, l'utilisation des ressources par un workflow peut être ajustée selon le temps d'exécution souhaité ou du budget prévisionnel des ressources à utiliser [26].

Les algorithmes d'ordonnancement sont essentiels pour tirer profit des avantages du cloud computing afin d'automatiser efficacement l'exécution des workflows scientifiques dans des environnements distribués [24, 27]. Ces algorithmes sont un composant essentiel pour les systèmes de gestion de workflows qui sont chargés d'orchestrer l'exécution des tâches sur un ensemble de ressources de calcul tout en préservant les dépendances de données. Les décisions prises par ces algorithmes d'ordonnancement sont généralement guidées par un ensemble d'exigences de qualité de service (QoS) définies par l'utilisateur. Leur succès dans la satisfaction de ces exigences de qualité de service repose sur l'utilisation efficace des ressources sous-jacentes. Par conséquent, les ordonnanceurs doivent être conscients des divers défis dérivés des fonctionnalités inhérentes au modèle de ressources cloud.

En général, le processus d'ordonnancement d'un workflow scientifique dans un système distribué consiste à affecter des tâches à des ressources et à orchestrer leur exécution de manière à préserver leurs dépendances [28]. L'affectation (mapping) est également effectuée de sorte que les différentes exigences de l'utilisateur soient satisfaites. Ces exigences déterminent les objectifs de l'ordonnancement et sont généralement définis en termes de métriques de performance telles que le temps d'exécution [29, 30], le coût d'utilisation du service [31] et d'exigences non fonctionnelles telles que la sécurité et la consommation d'énergie [32].
Pour planifier l'exécution d'un workflow dans un environnement cloud computing, deux sous-problèmes doivent être pris en compte [33, 34]. Le premier est connu sous le nom de l'approvisionnement (*provisionning*) des ressources qui consiste à sélectionner et à s'approvisionner de ressources de calcul qui seront utilisées pour exécuter chaque tâche du workflow scientifique. Ainsi une question fondamentale se pose, à savoir comment trouver la bonne configuration des ressources indispensables à l'exécution d'un workflow scientifique ? Cela signifie de disposer de méthodes heuristiques capables de déterminer le nombre de cœurs de calcul (machines virtuelles) à louer. Le second problème est l'étape d'ordonnancement ou d'allocation des tâches, dans laquelle chaque tâche est mappée sur la ressource la mieux adaptée. Le terme "ordonnancement" est souvent utilisé pour désigner la combinaison de ces deux sous-problèmes par certains auteurs de la littérature développant des algorithmes pour les ressources du cloud [35, 36]. Cette définition d'ordonnancement est également adoptée dans cette thèse.

Par rapport aux autres systèmes distribués tels que les clusters et les grilles de calcul, le cloud offre davantage de contrôle sur le type et la quantité de ressources utilisées. Cette

flexibilité et cette abondance de ressources créent le besoin d'une stratégie d'approvisionnement en ressources en adéquation avec l'algorithme de placement des tâches sur les ressources ; donnant ainsi naissance à des méthodes qui déterminent le nombre de machines virtuelles à utiliser, le moment opportun pour lancer l'exécution d'une tâche sur une ressource de calcul. Les ordonnanceurs doivent également relever un autre défi : trouver un compromis entre la durée d'exécution de l'application et le coût monétaire dû à l'exploitation des ressources [37–39]. Cela permet d'éviter de payer des prix inutiles et potentiellement prohibitifs. Certains algorithmes doivent se fonder sur la répartition géographique des ressources cloud et des incertitudes que cela comporte pour répondre aux exigences des utilisateurs telle que le temps d'exécution de l'application [40]. Par exemple, les délais de "*provisionning*" et de "*de-provisioning*" des machines virtuelles sont généralement très variables et imprévisibles. Ses variations de performances sont aussi observées dans les liaisons réseau et les systèmes de stockage. Cet indéterminisme empêche les algorithmes de prendre des décisions d'ordonnancement précises.

## Objectifs et contributions

Exécuter une application scientifique nécessite des ressources de calcul très performantes pour avoir les résultats dans un temps raisonnable. Ces ressources peuvent nécessiter de gros investissements si l'on doit déployer sa propre infrastructure.

Cette thèse porte sur l'ordonnancement des applications parallèles à forte intensité de données sur les ressources du Cloud Computing. Il s'agit de proposer des algorithmes qui proposent un bon compromis entre le temps d'exécution de l'application parallèle et le coût d'exploitation des ressources du Cloud Computing.

Les principales contributions concernent premièrement, la minimisation du temps d'exécution des workflows scientifiques par la réduction des données transférées dans le réseau. Pour atteindre cet objectif, nous nous sommes basés sur l'approche du *list scheduling* qui prend en compte toutes les contraintes liant les différentes tâches du workflow scientifique. Notre approche est fondée sur la localisation des données pour trouver un meilleur planning d'exécution des tâches du workflow scientifique. Deuxièmement nous proposons une procédure d'aide au choix des plateformes, basée sur la méthode de Pareto, afin de trouver le meilleur compromis entre le temps d'exécution et le coût d'utilisation des ressources du cloud computing. Nous terminons cette étude par une recherche dichotomique sur les solutions du front de Pareto pour proposer à l'utilisateur, moins de solutions de compromis.

## Organisation de la thèse

Notre approche méthodologique va s'articuler autour des chapitres suivants.

Dans le chapitre I, nous décrivons les ressources de calculs parallèles depuis le cluster jusqu'au cloud computing en passant par les grilles de calcul. Ce chapitre met en évidence la différence entre un workflow, un workflow scientifique et les systèmes de gestion de workflow. Cette étude permet également de comprendre l'intérêt du cloud computing pour les workflows scientifiques.

Le chapitre II présente les principales méthodes d'ordonnancement ainsi que les approches proposées dans chaque méthode. Cette description met en évidence les différentes approches proposées dans la littérature.

Le chapitre III décrit notre proposition (WSRDT) minimisant le temps d'exécution par la réduction des transferts de données dans le réseau. Cette étude est par la suite confrontée à l'un des algorithmes de list scheduling le plus connu dans le domaine.

Le chapitre IV nous permet de présenter le problème multi-objectif dans le domaine de l'ordonnancement dans le cloud computing et de proposer une approche basée sur le front de Pareto. Dans ce chapitre nous décrivons le principe de dominance et proposons un ensemble de solutions non-dominées. Ces solutions non-dominées permettent de présenter un ensemble de solutions que nous jugeons intéressantes selon un critère (soit le temps d'exécution, soit le coût d'utilisation des ressources) pour l'utilisateur.

Nous allons terminer ce travail par une conclusion générale ainsi que les perspectives issues de ce projet de recherche.

Chapitre I

# Ressources de calculs parallèles et leur utilisation

## Sommaire

## I.1 Introduction partielle

Dans une architecture de système en grappe, les groupes de processeurs sont organisés en centaines ou milliers de nœuds, au sein desquels les unités centrales communiquent via une mémoire partagée. Les nœuds sont interconnectés avec un système de communication organisé en réseau. Les applications parallèles utilisent des groupes de processeurs (CPU : *Central Process Unit*) sur un ou plusieurs nœuds [41].

Pour exploiter la puissance des ordinateurs en grappe, les applications parallèles doivent piloter plusieurs processus pour résoudre simultanément différentes parties d'un calcul. Pour être efficace, une application parallèle doit être conçue pour des systèmes spécifiques et également adaptée pour y fonctionner [42]. Ces systèmes sont différents par le nombre d'unités centrales (de calcul) connectées à une mémoire partagée ou à des mémoires caches. Cette différence se remarque également sur les caractéristiques du mécanisme de communication pour le passage des messages (et/ou données) entre les unités centrales [41, 43].

Pour les applications parallèles complexes, les ordinateurs en grappe modernes exigent deux techniques fondamentales. Premièrement, la conception de codes pour l'architecture du système. Deuxièmement, au moment de l'exécution, le code (généralement le Système d'Exploitation) doit contrôler l'agent logiciel qui gère l'ordonnancement des tâches et les ressources du système. Cet ensemble logiciel contrôle cette planification des tâches et la configuration des ressources du système en définissant les paramètres de l'environnement au moment de l'exécution [44].

Les paradigmes de programmation parallèle impliquent les points suivants :

1. l'utilisation efficace des unités de calcul ;
2. la communication entre les nœuds pour assurer l'exécution des tâches interdépendantes fonctionnant sur différents nœuds et échangeant des données.

Une application parallèle consiste généralement en un ensemble de processus (tâches) qui partage des données entre eux en communiquant par le biais d'une mémoire partagée sur une plateforme où les différents équipements sont interconnectés en réseau.

Il existe une panoplie d'environnements pour l'exécution des applications parallèles. Dans ce chapitre, nous présentons dans la section I.2 les différents ressources de calcul dans le domaine des calculs hautes performances (*High Performance Computing* : HPC), en partant des grappes de calculs (section I.2.1), jusqu'au cloud computing (section I.2.3) en passant par les grilles de calculs (section I.2.2). Dans la section I.3, il est question de présenter les applications parallèles de types workflow scientifique et de donner les systèmes de gestion des workflows. Nous terminons ce chapitre en montrant l'intérêt du

cloud pour exécuter les applications parallèles à la section I.4, que nous avons subdivisé en plusieurs sous-sections. D'abord, nous donnons les caractéristiques du cloud (section I.4.1), puis, donnons les modèles de déploiement du cloud (section I.4.2), suivie de la section I.4.3 où nous parlons de la technologie de la virtualisation qui vient rendre possible le paradigme du cloud computing. Ensuite, nous abordons les différents services proposés (section I.4.4). Enfin nous évoquons le service le plus adapté à l'exécution des workflows scientifiques à la section I.4.5.

## I.2 Différentes ressources de calcul

Pour réaliser des calculs intensifs et complexes, les scientifiques ont généralement recours à des ressources de calcul performantes : ordinateurs, grappes de serveurs et aujourd'hui nous avons le cloud computing qui offre plusieurs types de ressources quasi-illimitées. Les calculs intensifs font partie du domaine de calcul haute performance (HPC : *Hight Performance Computing*). Cette partie présente le contexte sur les différentes ressources de calcul afin de positionner et de donner une visibilité à notre travail.

### I.2.1 Grappes de calcul

Les scientifiques ont besoin d'effectuer des calculs de plus en plus massifs dans différents domaines de recherche, poussant à développer des ordinateurs toujours plus puissants : les superordinateurs ou supercalculateurs. Ainsi, pour satisfaire cette demande en ressource de calcul, les concepteurs (fabricants) préfèrent concevoir des ordinateurs sur la base des processeurs existants plutôt que de fabriquer un superordinateur possédant un processeur très puissant [45, 46].

Une grappe de calcul, appelé aussi grappe de serveur ou ferme de calcul, est constituée de différents serveurs composés d'un ou plusieurs processeurs généralement similaires constituant ainsi une plate-forme de calcul homogène. Ces différents serveurs sont connectés au moyen de réseau de communication très rapide. De plus, les serveurs d'une ferme de calcul sont physiquement localisés au même endroit [47].

De nos jours, les grappes de calcul sont très répandus dans les centres de recherche et les universités pour les calculs hautes performances (HPC) afin d'effectuer des simulations de tous genres dans les domaines de la génétique, la médecine, la physique, la météorologie, etc. La première place du classement 2020 des cinq cent (500) meilleurs grappes de calcul est attribuée à Supercomputer Fugaku avec sept million deux cent quatre-vingt-dix-neuf mille soixante-douze (7.299.072) cœurs pour atteindre cinq cent treize mille huit

cent cinq cent cinq (513.855) TFlop/s en puissance de calcul [48]. Propriété de l'entreprise japonnaise Fujitsu, ce grappe de calcul est principalement dédié à un large éventail d'applications qui abordent des questions sociales et scientifiques hautement prioritaires.

### I.2.2 Grilles de calcul

Une grille informatique ou grille de calcul (*grid computing*) peut être considérée comme un superordinateur virtuel qui est physiquement constitué de plusieurs grappes de calcul différents et délocalisées. Cette infrastructure est dite hétérogène car les éléments qui la constituent ont des vitesses de calcul différentes ; les éléments constituants une grille de calcul peuvent aller d'une ferme de calcul à un ordinateur personnel. Ces machines sont interconnectées par des réseaux de communication au moyen de câbles (fibre optique, la paire torsadée, le câble coaxial, ...) ou sans fil (boucle locale radio), pour atteindre un objectif de calcul commun et faire ainsi exécuter des applications scientifiques, qui sont généralement des calculs intensifs ou des traitements de très gros volumes de données [49].

Les grilles sont soit privées soit publiques et dans le dernier cas, elles sont accessibles à tous depuis Internet. Pour permettre à un plus grand public de bénéficier des différentes ressources et d'optimiser l'utilisation de celles-ci, la technologie de la virtualisation vient permettre de faire abstraction du matériel et d'augmenter le taux d'utilisation des ressources de la grille qui était auparavant sous-exploiter (des temps d'inactivité) ; donnant naissance à un nouveau paradigme : le cloud computing.

### I.2.3 Cloud computing

L'idée a été émise pour la toute première fois dans les années 60, où John McCarthy avait imaginé que les ressources informatiques seront fournies comme des services d'utilité publique [50]. Ce n'est qu'en 2006 que le cloud computing a véritablement fait son apparition avec l'avènement d'Amazon EC2 (Elastic Compute Cloud) [51]. En 2009, l'on a assisté à une explosion du cloud sur le marché avec les sociétés telles que Google (Google App Engine) [52], Microsoft (Microsoft Azure) [53], IBM (IBM Smart Business Service) [54], Sun (Sun Cloud) et Canonical Ltd (Ubuntu Enterprise Cloud). Le cloud computing est ainsi un modèle informatique qui permet d'accéder d'une façon transparente et à la demande, à un ensemble de ressources hétérogènes physiques (Bare-metal) ou virtualisées (Machine Virtuelles : VM) par le biais d'Internet. Ces ressources sont délivrées sous forme de services fexibles et élastiques, à base d'un modèle de facturation à l'usage (pay-as-you-go), dont les garanties sont offertes par le fournisseur via des contrats de niveau de service (SLA : Service Level Agreement).

## I.3 Workflows scientifiques et système de gestion de workflows

### I.3.1 Workflows scientifiques

La notion de workflow (“flux de travail” en français) est apparue pour la toute première fois dans l’industrie de l’imagerie électronique et de la gestion assistée par ordinateur [55], afin d’automatiser les processus de travail au sein des organisations. De plus, dans les infrastructures actuelles distribuées, gérant des ressources hétérogènes, telles que le cloud computing, bénéficier d’un environnement autorisant la définition et l’exécution des chaînes de traitement constitue une des fonctionnalités essentielles recherchée par les scientifiques et par le grand public.

Deux grandes catégories d’usages utilisent la notion de workflow : les protocoles expérimentaux, dans des domaines tels que la biologie, l’astronomie, la physique, la neuroscience, la chimie, etc. (appelées workflows scientifiques) et les chaînes de traitement pratiquées dans des domaines commerciaux, financiers, pharmaceutiques (appelés processus métiers). Elles donnent lieu à plusieurs pistes de recherche diverses, mais connexes. Dans le cadre de cette thèse, nous traitons plus particulièrement les workflows scientifiques.

Selon la WfMC (Workflow Management Coalition) [56], nous pouvons retenir la définition suivant du workflow indépendamment des domaines spécifiques : “Un workflow est l’automatisation d’un processus métier, en tout ou en partie, au cours de laquelle des documents, des informations ou des tâches sont passées d’un participant à un autre pour l’action, selon un ensemble de règles procédurales”. Cependant plusieurs définitions ont été proposées selon les catégories d’usages, ainsi en ce qui concerne le workflow scientifique, nous retiendrons la définition suivante :
Un workflow scientifique est la description d’un processus pour atteindre un objectif scientifique, généralement constitué d’un ensemble de tâches et des dépendances de données entre celles-ci.
En règle générale, les tâches de workflows scientifiques sont des étapes de calcul pour des simulations scientifiques ou des étapes d’analyse de données. Les éléments ou étapes courants des workflows scientifiques sont : l’acquisition, l’intégration, la réduction, la visualisation et la publication (par exemple, dans une base de données partagée) de données scientifiques. Les tâches d’un workflow scientifique sont organisées (au moment de la conception) et orchestrées (au moment de l’exécution) en fonction du flux de données et éventuellement d’autres dépendances spécifiées par le concepteur de workflow. Les work-

flows peuvent être conçus visuellement, par exemple , à l'aide de diagrammes de blocs, ou textuellement à l'aide d'un langage spécifique au domaine.

#### I.3.1.1 Description du workflow scientifique

Dans le but d'automatiser les traitements et gagner du temps, les scientifiques ont très souvent besoin de rassembler les tâches logicielles et de les interconnecter pour former une application. A l'entrée de l'application, des données commencent à être traitées par les tâches qui n'ont aucune dépendance avec d'autres tâches, puis celles-ci transmettent les résultats aux suivantes. Généralement, la structure d'une telle application se représente par un graphe orienté sans cycle : DAG (*Directed Acyclic Graph*). Dans ce graphe, chaque nœud (sommet) est une tâche et les arcs sont les contraintes de dépendance. Cette dépendance modélise aussi une communication entre deux tâches. Dans la littérature, la notion de workflow est généralement définie comme structure abstraite d'une application décrite par un DAG. D'autres modèles pour décrire les workflows existent depuis quelques décennies. Par exemple, les réseaux de Petri sont utilisés pour modéliser les workflows. L'étude de [57] discute de l'utilisation de réseaux de Petri dans le contexte des workflows. Quel que soit le modèle de workflow, les travaux de recherche distinguent l'exécution d'une application munie d'une structure workflow qui réalise un traitement sur un ensemble de données d'entrée et l'exécution en parallèle de plusieurs tâches d'une même application. Afin de simplifier l'emploi du terme workflow dans notre contexte de travail, nous donnons une autre définition du terme workflow.

Pour nous, un worfklow scientifique peut être défini comme un ensemble de tâches d'une (unique) application parallèle décrites par un DAG. Dans la suite nous utiliserons simplement le mot workflow pour désigner un workflow scientifique.

Exécuter une application parallèle peut prendre énormément de temps si le nombre de ressources est insuffisant. C'est pourquoi quand la taille des données devient très importante ou quand les calculs deviennent extrêmement intensifs, il est nécessaire d'utiliser des plates-formes distribuées comme le cloud computing, afin d'obtenir des gains de temps.

Le cycle de vie des workflows est divisé en quatre parties :

1. la composition du DAG, c'est-à-dire l'écriture du programme sous forme de DAG ;
2. le placement (l'ordonnancement des tâches sur les ressources) ;
3. l'exécution ;
4. la gestion des méta-données (les fichiers de sortie ou de résultat).

Ce qui nous intéresse dans cette thèse est l'étape du placement : l'ordonnancement, qui

est un point crucial pour obtenir de bonnes performances et gagner du temps. Bien ordonnancer les tâches d'un workflow sur les ressources du cloud computing, c'est optimiser l'affectation des différentes tâches sur les différentes ressources disponibles.

#### I.3.1.2 Caractéristiques du workflow

Les applications parallèles à forte intensité de données (notamment les workflows scientifiques) sont couramment utilisées dans la majorité des disciplines exploitant souvent des ressources de données riches et variées ainsi que des plateformes informatiques parallèles et distribuées. Les workflows fournissent un moyen systématique de décrire les méthodes nécessaires pour représenter une application parallèle. Ils assurent l'interface entre les spécialistes du domaine et les infrastructures informatiques. Les systèmes de gestion de workflow (WMS) effectuent les analyses complexes sur une variété de ressources distribuées. Avec l'augmentation spectaculaire des volumes de données et de la diversité dans chaque domaine, les workflows jouent un rôle de plus en plus important, permettant aux chercheurs de formuler des méthodes de traitement et d'analyse afin d'extraire des informations à partir de sources de données multiples et d'exploiter un très large éventail de données et de plateformes de calcul. Pour mener à bien une étude, les workflows sont très souvent modélisés sous forme de DAG comme sus-mentionné.

Un workflow, comme l'illustre la Figure I.1, est un ensemble de nœuds et d'arcs. Les nœuds représentent les tâches du workflow et les arcs entre les différentes tâches représentent soit une dépendance de données, c'est-à-dire un transfert de fichier, soit une dépendance de flux entre deux tâches. Chacune des tâches composant le workflow a une durée estimée, nécessite un ensemble de fichier d'entrée pour démarrer son exécution et produit un ensemble de fichier de sortie à la fin. Lorsqu'un fichier de sortie produit par une tâche est consommé en entrée par une autre, cela crée une dépendance de données entre les deux tâches. La dépendance entre les deux est représentée par un arc orienté. Les fichiers d'entrée qui ne sont générés par aucune des tâches du workflow sont appelés fichiers d'entrée du workflow. Inversement, les fichiers de sortie qui ne sont pas consommés par une tâche sont appelés les fichiers de sortie du workflow.

### I.3.2 Systèmes de gestion des workflows

Les workflows scientifiques sont utilisés pour modéliser les applications complexes au format DAG (Directed Acyclic Graph) avec des nœuds et des arcs qui expriment facilement l'ensemble du processus de données avec ses dépendances [58]. Lorsque plusieurs données (en entrée et sortie) sont consommées et produites pendant les expériences scientifiques, cela rend les workflows gourmands en données (*data-intensive*) . Au fur et à mesure que

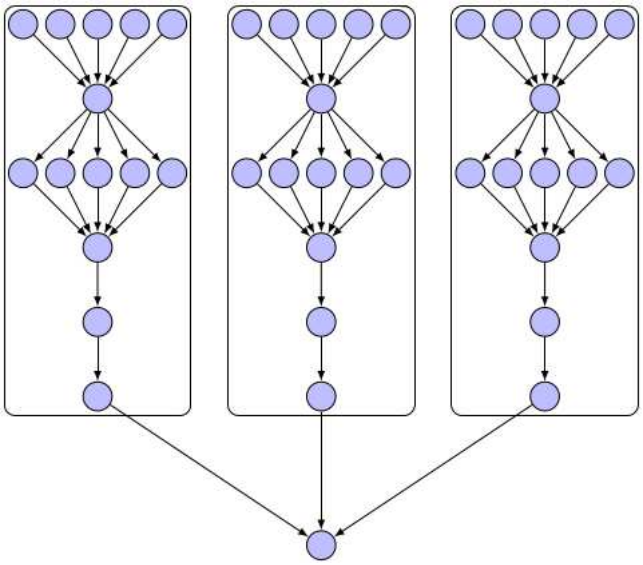


FIGURE I.1 – Exemple simple de workflow.

la complexité des applications scientifiques augmente, le besoin d'utiliser des systèmes de gestion de workflow (*Scientific Workflow Management System*) augmente également pour automatiser et orchestrer le traitement (exécution) de bout en bout. Afin de traiter les données à grande échelle, ils doivent être exécutés dans un environnement distribué tel que le cloud.

Le système de gestion de workflow est un framework efficace pour gérer des ensembles de données massives dans un environnement informatique (grille ou cloud). De nos jours, il existe plusieurs systèmes de gestion de workflow pour les grilles de calcul et le cloud, dont les plus répandus sont KEPLER [59], TAVERNA [60], TRIANA [61], ASKALON [62], PEGASUS [63]. La gestion de workflow est principalement la coordination des tâches qui constituent l'ensemble organisationnelle du workflow. Ainsi, le système de gestion de workflow est principalement l'ensemble des outils indispensables à une meilleure organisation (définir, mapper et exécuter) des tâches tout en se fondant sur les différentes données (fichiers) en entrée et en sortie (comme l'illustre la Figure I.1). Cette section décrit les systèmes de gestion de workflow (les plus populaires) sus-mentionnés, développés par la communauté scientifique. Chaque système de gestion de workflow développe sa propre représentation de workflow. Généralement, les modèles de workflow sont classés en deux catégories : les systèmes basés sur des scripts et les systèmes basés sur l'interface graphique [64–66].

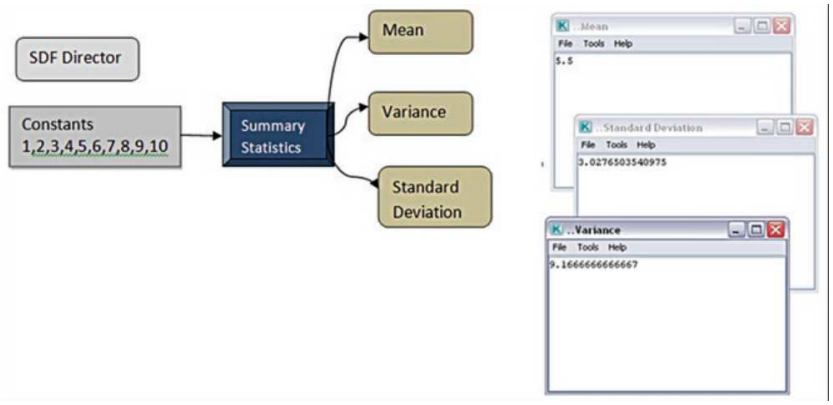



FIGURE I.2 – Représentation de workflow scientifique avec Kepler.

### I.3.2.1 KEPLER

KEPLER [59] est une collaboration inter-projets visant à mettre au point un système de gestion de workflow scientifique dans lequel les workflows scientifiques peuvent être conçus et exécutés.
KEPLER est un framework Java open-source développé et maintenu par la communauté Kepler et est un dérivé de Ptolemy [67, 68]. Il est conçu pour aider la communauté scientifique à analyser et à modéliser une vaste gamme d'applications scientifiques. Cette application est utilisée pour l'analyse et la modélisation de données scientifiques. Elle prend en charge la représentation visuelle des processus. A l'aide d'une représentation visuelle, elle simplifie l'effort de création.

KEPLER permet de créer des modèles en utilisant la représentation visuelle. Les workflows scientifiques sont créés par "glisser-déposer" des composants dans la zone de création du workflow et la connexion est établie entre les composants pour construire un flux de données spécifique. Les workflows scientifiques montrent le flux de données entre les composantes d'analyse discrète et de modélisation, comme le montre la Figure I.2. Il permet pour concevoir et exécuter des workflows scientifiques à partir de l'interface graphique.

Les différents modules de Kepler, comme l'illustre la Figure I.3, permettent de définir la structure du workflow scientifique afin d'automatiser l'exécution des différentes tâches :

— Director : il est utilisé pour représenter les différents types de composants avec son modèle de calcul dans le workflow, et contrôle le flux d'exécution global ;

— Actor : exécute les instructions du *Director*, et l'acteur composite exécute l'action des opérations complexes ;

— Port : Les acteurs (tâches) sont reliés entre eux à l'aide de ports. Un acteur peut contenir un ou plusieurs ports pour produire ou consommer des données afin de

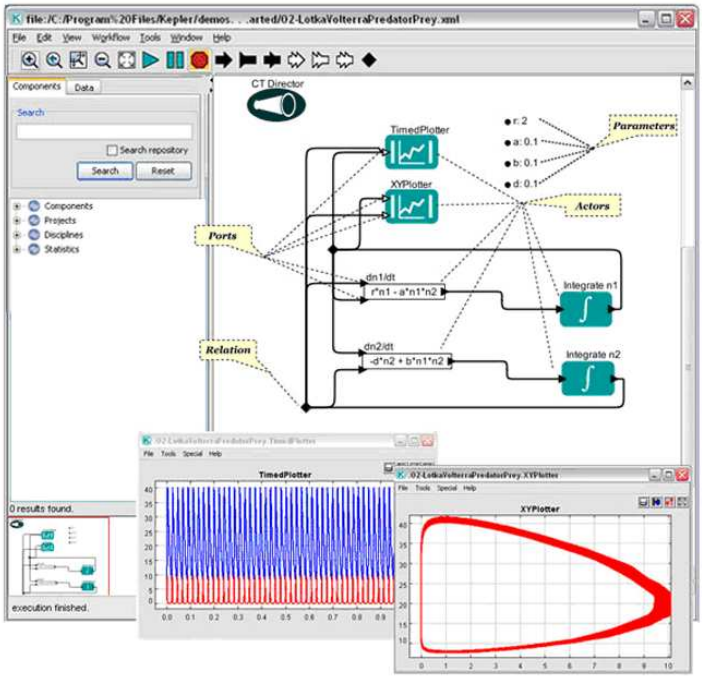



FIGURE I.3 – Composants de base de Kepler.

communiquer avec d'autres acteurs. Un lien est utilisé pour transmettre le flux de données d'un port acteur à un autre ;

— Relations : permet de rélier un flux de données ;

— Parameters : les valeurs configurables rattachées à un workflow, un director ou un actor.

Les caractéristiques de Kepler :

— le moteur d'exécution et l'interface utilisateur graphique sont pris en charge, ce qui est utile pour exécuter les workflows avec l'interface graphique ou l'interface ligne de commandes ;

— la réutilisabilité est prise en charge dans Kepler ; des modules et composants peuvent être créés, sauvegardés et réutilisés pour d'autres applications de workflow également ;

— prise en charge native des applications de traitement parallèle ;

— la bibliothèque réutilisable contient 350 composants de processus prêts à l'emploi. Ces composants peuvent être facilement personnalisés et utilisés pour d'autres applications parallèles.

### I.3.2.2 TAVERNA

TAVERNA [60] est un système de gestion de workflow open-source basé sur Java, créé par l'équipe myGrid qui conçoit et exécute des workflows scientifiques. L'objectif principal de Taverna est d'étendre le soutien au domaine des sciences de la vie, la chimie, la médecine afin d'exécuter des workflows scientifiques et de soutenir l'expérimentation sur le silicium, où les expériences sont réalisées par simulation informatique et reflètent de près le monde réel. Il prend en charge les services web, des interfaces de programmation (API) Java, des scripts R et des fichiers de données tabulaires (CSV).

La Figure I.4 illustre la représentation d'un workflow scientifique avec TAVERNA. Les workflows conçue avec Taverna peuvent accéder aux ressources locaux ou sur le web (données) et aussi aux ressources de calcul d'une infrastructure distribuée grâce à des API. TAVERNA utilise un langage de flux unifié conceptuel simple basé sur XML, qui se compose de trois entités principales :

— les processus, représentant une activité (tâche) de calcul dans un workflow scientifique ;

— les liaisons de données, représentant les dépendances de données entre deux activités (tâches) ;

— les contraintes de coordination, représentant les dépendances de contôle entre deux tâches.

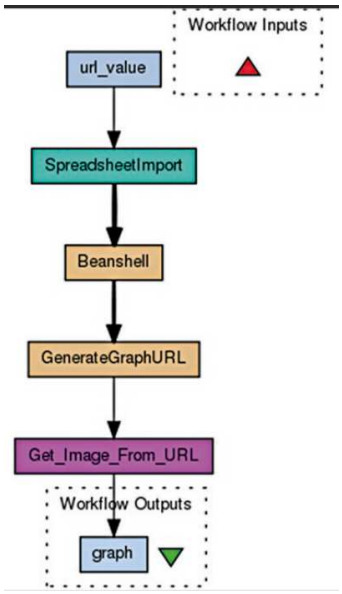



FIGURE I.4 – Représentation du workflow scientifique avec Taverna (exemple d'utilisation du service d'importation de feuilles de calcul pour importer des données à partir d'une feuille de calcul Excel).

TAVERNA intègre un module de stockage de données qui exploite généralement des systèmes de gestion de bases de données et des systèmes de fichiers pour gérer toutes les données pendant l'exécution du workflow.

#### I.3.2.3 TRIANA

TRIANA [61] est un système de gestion de workflow scientifique basé sur le langage Java et utilise une interface graphique avec des outils d'analyse de données. TRIANA dispose de divers outils intégrés pour la manipulation d'images, l'analyse de signaux et autres, qui permettent également aux chercheurs d'intégrer leurs propres outils. TRIANA dispose du moteur de workflow appelé *Triana Controlling Service* (TCS) pour exécuter les workflows. La Figure I.5 illustre le déroulement du workflow scientifique conçu dans Triana. L'interface graphique de Triana se connecte au système de commande de sécurité soit localement, soit à distance ; elle exécute l'application de workflow et la visualise localement. L'interface graphique utilisateur (GUI) de Triana contient la collection de composants Triana qui permet de créer des applications de workflow du comportement souhaité. Chaque composant contient des informations sur les données d'entrée et de sortie. Triana fournit également les interfaces de lecture et d'écriture généralisées permettant d'intégrer la représentation et les services du workflow d'une tierce personne grâce à un interface graphique.

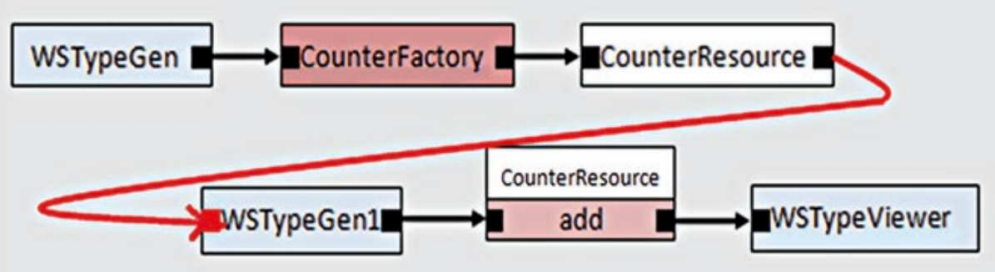



FIGURE I.5 – Représentation du workflow scientifique avec TRIANA.

#### I.3.2.4 ASKALON

Le projet ASKALON [62] est développé par l'Université d'Innsbruck (USA). Il fournit un environnement idéal basé sur de nouveaux services, outils et méthodologies pour l'exécution d'applications parallèles en environnement cloud et grid. Son objectif est de simplifier le développement et l'optimisation des applications parallèles. Les workflow sont décrits à l'aide du langage AGWL (*Abstract Grid Workflow Language*). Il permet aux chercheurs de concevoir des applications en utilisant la modélisation au lieu de la programmation et d'optimiser les applications parallèles en fonction des contraintes (dépendances). De nombreuses applications du monde réel (telles que *Wien2k* [69, 70], *Invmod* [71], *MetwoAG*

[72] et *GRASIL* [73]) ont été devéloppées par ASKALON. La Figure I.6 représente la représentation du workflow scientifique par ASKALON. Ces principaux composants d'ASKALON sont :

— *Resource broker* : réserve les ressources nécessaires à l'exécution des workflows ;

— *Resource monitoring* : surveillance des ressources de grille et de cloud computing par l'intégration et la mise à l'échelle des ressources à l'aide de nouvelles techniques ;

— *Information service* : service de découverte et d'organisation des ressources et des données ;

— *Workflow executer* : exécution d'applications parallèles dans des sites distants du cloud computing et de grille ;

— *Scheduler* : cartographie les tâches de l'application parallèle sur la grille ou les ressources du cloud computing ;

— *Performance prediction* : étudie de nouvelles techniques pour l'estimation des exécutions.de temps de transfert et de transfert ainsi que la disponibilité des ressources.

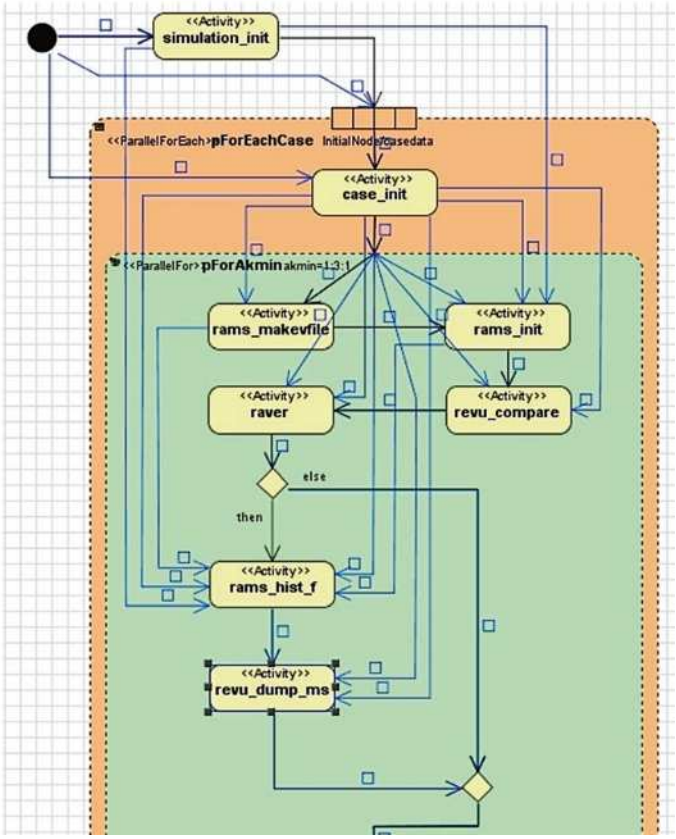



FIGURE I.6 – Représentation du workflow scientifique avec ASKALON.

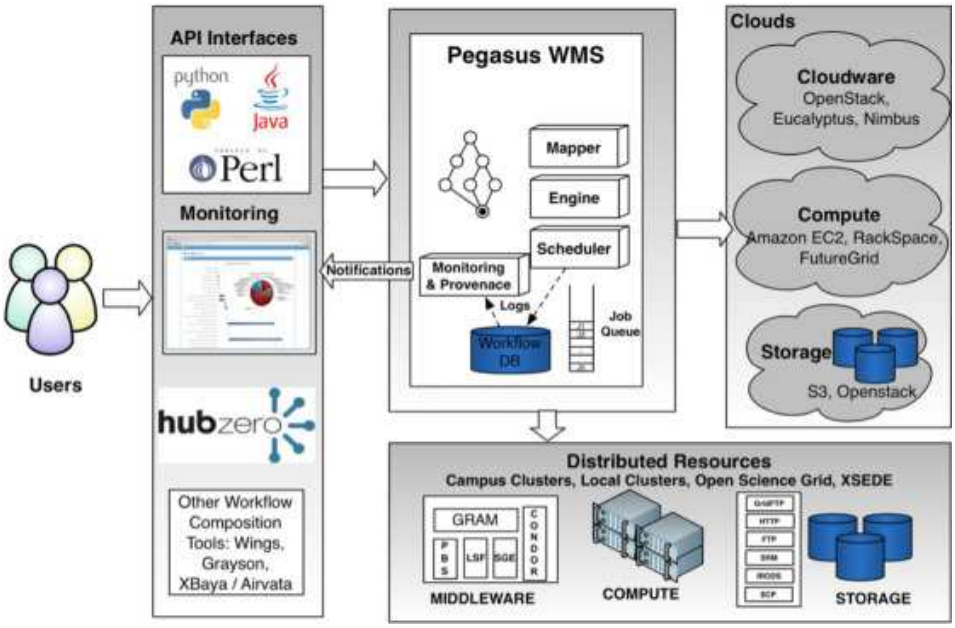



FIGURE I.7 – Architecture de PEGASUS.

### I.3.2.5 PEGASUS

PEGASUS [63] est un système de gestion de workflow open-source développé à l'Université de Californie du Sud, qui intègre un certain nombre de technologies pour exécuter des applications parallèles dans des environnements hétérogènes tels que la grille, les clusters et le cloud. Il a été utilisé dans un certain nombre de domaines scientifiques tels que la bioinformatique, la physique des ondes gravitationnelles, les sciences océaniques etc. L'architecture du système de gestion de workflow Pegasus est illustrée à la Figure I.7.

Pont entre les domaines scientifiques et l'environnement d'exécution, il permet de passer d'une description abstraite du workflow à une description exécutable en vue de le mapper sur les ressources et exécuter les tâches de l'application parallèle dans leur ordre de dépendance. La représentation du workflow scientifique dans Pegasus est illustrée à la Figure I.1.

Pegasus se compose de cinq (5) sous-systèmes (sous-couches) principaux (les) pour effectuer diverses fonctionnalités qui incluent le mappeur, le moteur d'exécution locale, le planificateur de tâches, le moteur d'exécution à distance et le composant de surveillance.

— *Mapper* : génère un workflow exécutable à partir des workflows abstraits fournis par l'utilisateur. Mapper trouve une ressource appropriée et d'autres composants connexes qui sont nécessaires à l'exécution du workflow. Il restructure également les workflows pour optimiser les performances ;

— *Local execution engine* : s'occupe de la soumission des tâches suivant l'état de la tâche, et determine la date d'exécution d'une tâche. Puis soumet les tâches à la file d'attente du *Job scheduler* ;

— *Job scheduler* : gère les tâches sur les ressources locales et distantes ;

— *Remote execution engine* : gère l'exécution des tâches sur les nœuds de calcul distants ;

— *Monitoring component* : surveille le workflow et fait le suivi de toutes les informations et les logs.

Les workflows scientifiques modélisent des applications complexes représentées sous forme de DAG où les noeuds et les arcs expriment facilement le processus de données entier avec ses dépendances. Les systèmes de gestion des workflows fournissent des outils pour définir, mapper et exécuter des applications parallèles. Cette section (I.3.2) a mis en évidence les principales caractéristiques des systèmes de gestion des workflows les plus populaires tels que KEPLER, PEGASUS, TRIANA, TAVERNA, et ASKALON. Cependant, les workflows créés à l'aide de WMS nécessitent les éléments importants suivants pour leur exécution efficace : algorithme d'ordonnancement et environnement informatique hétérogène et évolutif. La section suivante (I.4) présente l'environnement d'exécution des workflows qui connaît une popularité ces dernières années, où l'utilisateur a accès aux ressources informatiques à tout moment et comme il le souhaite.

## I.4 Intérêt de cloud computing pour les workflows scientifiques

Selon la NIST (*National Institute of Standards and Technology*) [74], le cloud computing est un modèle permettant un accès quel que soit le lieu et à la demande à un ensemble de ressources informatiques (réseaux, serveurs, stockage, application, etc.) partagées. Ce modèle possède cinq (05) caractéristiques essentielles, déployé sur trois (03) types d'architecture et fournissant trois (03) niveaux de service [75].

### I.4.1 Caractéristiques du cloud computing

La clé de ces caractéristiques introduites ici est le concept de "multi-location" (*multitenance*), c'est-à-dire l'idée que de nombreuses applications, utilisateurs et entreprises peuvent tirer profit des ressources utilisées selon les critères suivants :

**Libre-service à la demande :** un utilisateur peut demander autant de ressources qu'il souhaite chez un fournisseur et est approvisionné automatiquement selon ses besoins sans aucune interaction humaine.

**L'accès à un large réseau :** l'utilisateur a accès aux différents services du cloud plus facilement au travers du réseau, via une interface web et par le biais de plusieurs types de terminaux (e.g. : téléphones mobiles, tablettes, ordinateurs portables et stations de travail).

**Mutualisation des ressources (Pooling) :** les ressources physiques sont géographiquement réparties sur tout le globe terrestre, leur interconnexion permet d'avoir un super-ordinateur hors norme permettant de desservir plusieurs utilisateurs à l'aide d'un modèle multi-location. Ces différentes ressources (physiques) sont souvent virtualisées et sont par la suite attribuées et réaffectées de façon dynamique en fonction de la demande des consommateurs. L'utilisateur n'a aucune connaissance sur la localisation exacte des ressources, mais peut éventuellement définir une localisation abstraite permettant de définir la facturation.

**Scalabilité et élasticité :** la capacité d'une ressource en cours d'utilisation peut être augmentée ou diminuée (automatiquement) en fonction de la variation de la charge, et l'utilisateur peut aussi décider d'augmenter ou de diminuer le nombre de ses ressources en fonction de ses besoins. L'utilisateur a ainsi l'illusion que les ressources sont infinies.

**Facturation à l'usage :** la consommation des ressources du cloud computing s'adapte au plus près aux soins de l'utilisateur. Le fournisseur est capable de mesurer de façon précise la consommation (en durée et en quantité) des différents services (calcul, stockage, réseaux, bande passante,...). Cela permet au fournisseur de facturer l'utilisateur selon sa réelle consommation.

### I.4.2 Modèles de déploiement

L'architecture du cloud est fonction du public cible et du type de service à offrir. Ainsi le NIST définit trois types de modèle de déploiement. Un cloud destiné à l'usage restreint d'une entreprise porte le nom de cloud privé. Dans le cas contraire, c'est-à-dire ouvert au grand public, il porte le nom de cloud public. Il existe des situations pour lesquelles les utilisateurs de cloud privé font usage d'un cloud public pour des besoins précis. Cette configuration décrit une nouvelle forme de cloud connue sous le nom de cloud hybride [76, 77].

**Le coud privé :** une entreprise ou une organisation décide de mettre en place sa propre infrastructure pour une utilisation exclusive selon ses besoins. Cette infrastructure est composée d'un ou plusieurs serveurs dont l'entreprise a un contrôle sur les différents équipements. La mise en commun de plusieurs cloud (privés) de différentes entreprises ou organisations donne naissance à un autre modèle de déploiement, dénommé cloud communautaire. Dans

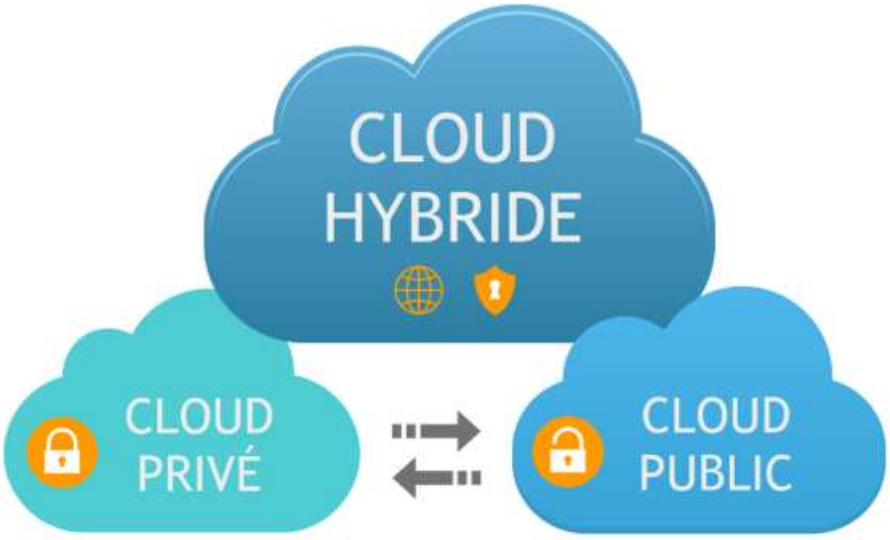



FIGURE I.8 – Différents modèles de déploiement du cloud.

ce modèle, les ressources sont reparties entre les membres de la communauté (entreprises, organisation, . . . ) selon une politique qu'ils ont définis.

**Le cloud public :** une entreprise, un établissement universitaire, un organisme gouvernemental, ou une combinaison de ceux-ci peuvent décider de mettre à la disposition d'un grand public une infrastructure cloud computing qui répond aux besoins de ceux-ci. Les utilisateurs peuvent ainsi décider d'externaliser leurs données, applications et autres, moyennant un coût forfaitaire défini dans le contrat qui lie l'utilisateur et le fournisseur de service. Les équipements sont ainsi détenus, gérés et exploités par le fournisseur.

**Le cloud hybride :** une entreprise disposant d'une infrastructure privée peut décider d'obtenir des équipements complémentaires auprès d'un fournisseur. La combinaison des deux infrastructures (privée et public), constitue ainsi une entité unique liée entre elle par une technologie standard ou exclusive qui permet la portabilité des données et des applications tout en répondant aux exigences de l'entreprise.

Il est certes important de parler des caractéristiques et des différents modèles de déploiement du cloud, mais, il est encore plus important de parler de l'une des technologies sur laquelle repose ce paradigme. Sachant les ressources de la grille de calcul sous-exploitées, il fallait trouver un moyen de rentabiliser les investissements en proposant aux utilisateurs qui le désirent, d'utiliser les ressources qu'ils ont besoin, sans toutefois pensé à déployer une infrastructure informatique. Cette technologie est la virtualisation.

### I.4.3 Place de la virtualisation dans le Cloud Computing

Tous les modèles de déploiement du cloud computing s'appuie principalement sur la technologie de la virtualisation, que l'on définit comme étant l'ensemble des techniques matérielles et logicielles qui permettent de faire fonctionner sur une seule machine physique différents systèmes d'exploitations (SE) [78–80]. Séparées les unes des autres, l'utilisateur a l'illusion d'être en présence de plusieurs machines distinctes. Concrètement, une couche d'abstraction logicielle communément appelée hyperviseur (*Virtual Machine Manager : VMM*) est installée dans la machine physique (machine hôte) et est en charge de créer des machines virtuelles (VMs).

En plus d'être chargée de la création des VMs, la VMM a pour rôle de partager et de coordonner les accès aux ressources de la machine physique sous-jacent entre les VMs crées. Bien qu'elle soit virtuelle, une VM dispose d'un ensemble de périphériques. Une VM possède ainsi une puissance de calcul (en nombre et capacité du processeur virtuel : vCPU), un espace de stockage (disque HDD ou SSD), une quantité de mémoire et une ou plusieurs cartes réseaux. Puisque plusieurs VMs peuvent s'exécuter sur la même machine physique, la virtualisation accorde un grand intérêt à l'isolation ; aussi bien à l'isolation des pannes qu'à l'isolation de performance ; et à la sécurité des données. L'isolation des pannes permet d'éviter que la compromission d'une VM ne se propage, quant à l'isolation de performance, elle garantit qu'aucune sur-exploitation de ressources par une VM ne prive d'autres de ce qui leur est réservé. Il existe plusieurs familles de solutions de virtualisation : virtualisation complète (Figure I.9(a)), la para-virtualisation (Figure I.9(b)) et la virtualisation assistée par le matériel (Figure I.9(c)).

**La virtualisation complète :** (Figure I.9(a)) consiste à émuler entièrement le matériel du système hôte pour le rendre accessible à une VM. Les accès aux périphériques s'effectuent comme si la VM était en présence d'un matériel réel. L'hyperviseur se charge d'effectuer les opérations nécessaires pour translater (convertir) les instructions privilégiées de la VM en opérations compréhensibles et réalisables sur les ressources de la machine physique. Dans cette forme de virtualisation, le SE invité et les applications qui y sont installées ne requièrent aucune modification ou adaptation. Toutefois, cette solution de virtualisation est sujette à une perte de performance importante due au nombre de couches logicielles "à traverser" lors du traitement de toute requête. Cette forme de virtualisation est implémentée entre autres par VMware Workstation [81] et Qemu [82].

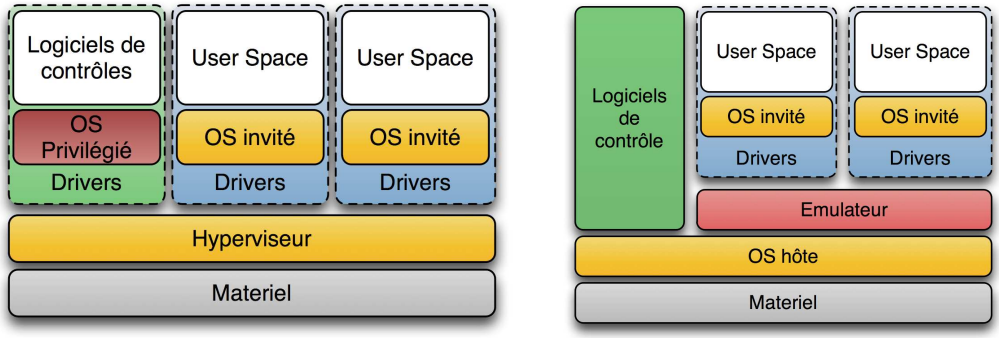
**La para-virtualisation :** (Figure I.9(b)) a été proposée afin de résoudre les problèmes de perte de performance observés dans l'utilisation de la virtualisation complète. La para-

virtualisation consiste à modifier le SE invité de façon à lui inclure des instructions spéciales nommées hypercalls. Ces dernières permettent des accès aux périphériques sans avoir recours aux translations. Cette forme de virtualisation a l'avantage d'offrir des performances d'applications exécutées dans les VMs semblables aux performances obtenues lors d'une exécution sur un système non virtualisé (aussi appelé système natif). Cette technologie a été principalement implémentée par Xen [83, 84] et VMware ESXi [85]. Toutefois, cette technologie de virtualisation exige de disposer du code source du SE invité. Cette contrainte limite l'exploitation de la paravirtualisation uniquement aux SEs non propriétaires.

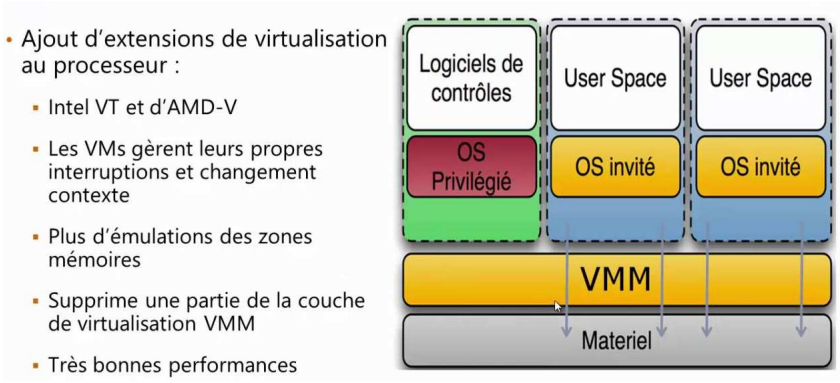
**La virtualisation assistée :** (Figure I.9(c)) l'apparition des processeurs intégrant des instructions de la virtualisation a fait naître une autre forme de technologie de la virtualisation : la virtualisation assistée par le matériel (HVM : *Hardware Virtual Machine*). Elle consiste à gérer l'ensemble des opérations de virtualisation de façon matérielle et pas logicielle comme dans les solutions précédentes. Cette forme de virtualisation offre des performances de VM identiques aux OS natifs mais a pour contrainte de disposer d'un matériel intégrant les instructions de virtualisation. Intel et AMD, grâce à leurs technologies respectives VT-x (*Virtual Technology*) et VM (*Secure Virtual Machine*), offrent ces instructions et sont exploitables par les hyperviseurs VMware ESXi, Xen, KVM, VirtualBox ou Hyper-V.

A l'heure actuelle, la virtualisation est très connue. On entend parler de virtualisation de serveur, de baremetal, mais aussi de virtualisation de poste de travail. Cette technologie présente des avantages tant pour les machines physiques que pour les environnements virtualisés. Pour les machines physiques, cela s'explique par une utilisation optimale des ressources du parc informatique à travers l'installation, le déploiement et la migration d'un environnement virtualisé d'une machine physique à une autre. Le taux de variation de la consommation énergétique, de l'entretien matériel et logiciel est relativement faible si la machine physique est utilisée à 5% ou 99% de ces capacités. Quant aux environnements virtualisés, la sécurité est plus accrue car invisible pour l'attaquant. L'isolation des différents utilisateurs simultanés met également un point d'honneur sur la sécurité. L'allocation dynamique de la puissance de calcul ou de l'espace de stockage est également l'un des avantages de la virtualisation. Comme toutes technologies, la virtualisation a également des inconvénients liés à la performance et à l'exécution de l'environnement virtualisé dû à la couche d'abstraction matérielle permettant d'avoir accès aux ressources de la machine physique. En cas de panne d'une machine physique, l'environnement virtualisé hébergé sur celle-ci sera impacté. Pour pallier ce problème, la virtualisation est souvent mise en œuvre avec des redondances de données.

La virtualisation intervient à tous les niveaux de services qu'offrent le cloud computing. Dans la suite, nous parlons des différents services du cloud computing et donnons le service




(a) Virtualisation Complète (b) Para-Virtualisation




(c) Virtualisation assitée par matériel

FIGURE I.9 – Description des hyperviseurs.

le mieux adapté à l'exécution de workflows scientifiques.

### I.4.4 Différents services

XaaS (X as a Service) représente la base du paradigme du cloud computing, où X représente un service tel qu'un logiciel, une plateforme ou une infrastructure. Nous présentons, dans cette section, les trois modèles de services [86] définit par le NIST, à savoir : (1) le logiciel en tant que services (SaaS : *Software as a Service*), (2) la plateforme en tant que service (PaaS : *Platform as a Service*), (3) l'infrastructure en tant que service (IaaS : *Infrastructure as a Service*). Contrairement aux systèmes traditionnels où l'utilisateur a un contrôle sur tous les équipements de l'infrastructure, dans le cloud, l'utilisateur n'a le contrôle que sur une partie des équipements de l'infrastructure.

#### I.4.4.1 Software as a Service (SaaS)

Le SaaS offre aux utilisateurs des applications sous forme de services en ligne déjà déployés dans le cloud. Ce service est géré par le fournisseur de SaaS de façon transparente

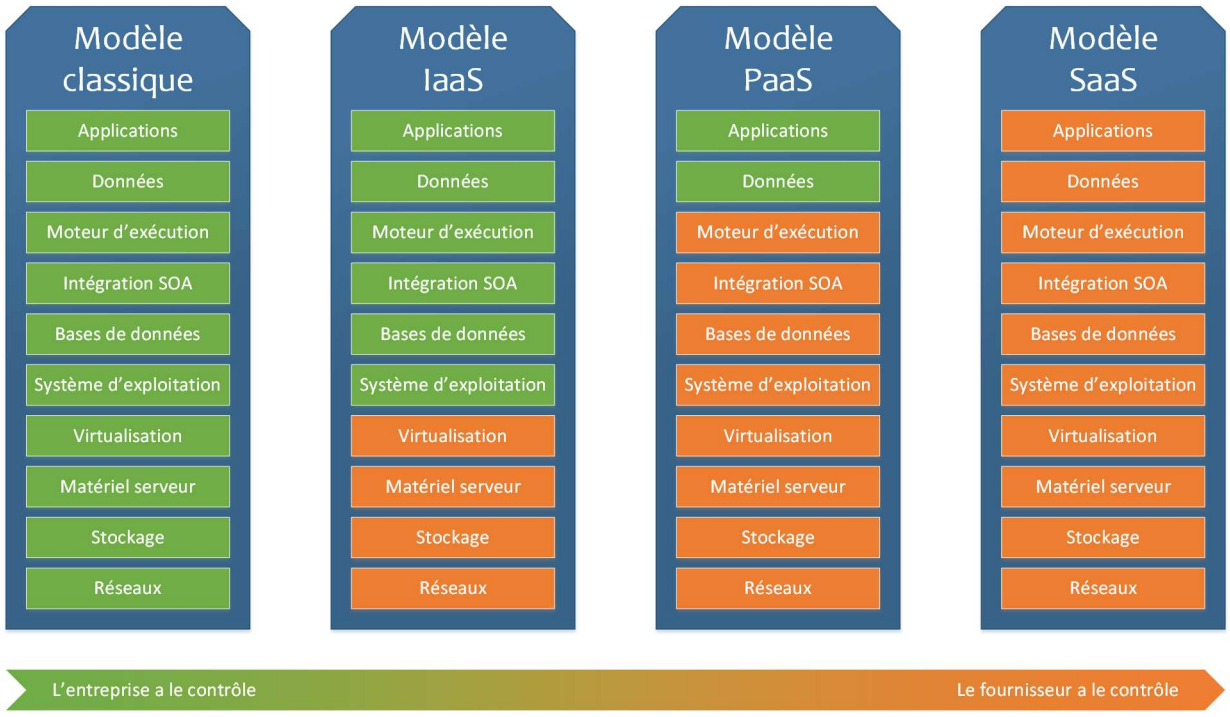



FIGURE I.10 – Modèle traditionnel vs les différents services du cloud.

pour les utilisateurs. Ces derniers ne savent ni où ni comment ces applications sont déployées. Ils ne paient pas pour les posséder, mais plutôt pour les utiliser. Ils peuvent accéder à ces applications à tout moment via internet par le biais de n'importe quels équipements connectés, tels que les ordinateurs portables, tablettes, smartphones. Les applications sont utilisées directement via l'interface web. Les principales applications actuelles de ce modèle sont les applications de gestion de la relation client (CRM : *Customer Relationship Management*), la vidéo conférence, les communications unifiées, les outils collaboratifs, la messagerie. Les principaux fournisseurs de cette catégorie sont Salesforce.com (logiciels CRM) et Google (Gmail, Google Apps).

### I.4.4.2 Platform as a Service (PaaS)

Ce service cloud est plus orienté pour servir des développeurs d'applications. Il offre une plateforme entièrement configurée et gérée, sur laquelle l'utilisateur (développeur) peut développer, tester et exécuter ses applications. Le déploiement des solutions PaaS est automatisé et évite à l'utilisateur d'avoir à acheter des logiciels ou d'avoir à réaliser des installations supplémentaires. Le PaaS offre une grande flexibilité, permettant notamment de tester rapidement un prototype (d'application). Il favorise la mobilité des utilisateurs, puisque l'accès aux données et aux applications peut se faire à partir de n'importe quel périphérique connecté. Les principaux fournisseurs du PaaS sont SalesForce.com (Force.com), Google (Google App Engine), Microsoft (Windows Azure), Facebook (Facebook Platform).

#### I.4.4.3 Infrastructure as a Service (IaaS)

L'IaaS permet à des utilisateurs de disposer des ressources d'infrastructure telles que des serveurs de calcul, de stockage, de réseau sous forme de services publics et privés. Les utilisateurs d'IaaS sont libérés des charges causées par la possession des équipements physiques, la gestion ou le contrôle du matériel sous-jacent. Par conséquent, ils n'ont pas accès directement aux machines physiques, mais indirectement à travers les machines virtuelles (VMs) et des machines de types *Bare-metal*. Les utilisateurs ont la plupart du temps un contrôle presque complet sur les machines (VMs et Bare-metal) qu'ils louent : ils peuvent choisir des images de systèmes d'exploitation pré-configurées par le fournisseur ou des images de machines personnalisées contenant leurs propres applications, bibliothèques et paramètres de configuration. Les utilisateurs peuvent également choisir les différents ports d'Internet à travers lesquels les machines seront accessibles, etc. Les utilisateurs peuvent héberger et exécuter des logiciels, des applications quelconques ou encore stocker des données sur l'infrastructure et ne paient que les ressources qu'ils consomment. La particularité de l'IaaS est de fournir des ressources informatiques, qui soient dynamiques, flexibles, extensibles et qui possèdent de bonnes propriétés de passage à l'échelle pour répondre aux besoins spécifiques des utilisateurs. Le cloud IaaS est vu ainsi comme une source infinie de ressources de calcul, de stockage et de réseau accessibles avec un modèle de paiement à l'usage. Il existe de nombreux fournisseurs commerciaux et des solutions open-source de l'IaaS. Parmi les plateformes commerciales, nous pouvons citer : Amazon EC2 [51], Rackspace [87], GoGrid [88], Microsoft Azure [53] et d'autres. Dans les communautés open-sources, plusieurs plateformes ont été développées, à savoir : Eucalyptus [89], Nimbus [90], OpenNebula [91], OpenStack [92] et bien d'autres.

Le cloud computing à travers ces services fiables, fournit à la demande sur Internet (généralement représentés sous la forme d'un nuage) avec un accès facile à des ressources de calcul, de stockage et de réseaux de machines virtuelles infinies. Ces ressources peuvent être reconfigurées dynamiquement pour s'adapter à une charge variable (mise en échelle : *scalability*), ce qui permet également une utilisation optimale des ressources. Ce pool de ressources est généralement exploité selon un modèle de paiement à l'utilisation, dans lequel des garanties sont offertes par le fournisseur de services cloud au moyen d'accords de niveau de service (*Service-Level Agreements* : SLA) personnalisé. Le SLA est un contrat dans lequel un service est formellement défini. Le SLA définit la qualité du service fournie aux utilisateurs par les fournisseurs de services. L'une des principales différences entre la grille de calcul et le cloud est la qualité de service, car la grille de calcul n'offre que le meilleur service possible. En outre, les services cloud fournissent un soutien pour la tarification, la comptabilité et la gestion des SLA.

Le SaaS, le PaaS et l'IaaS peuvent être utiles pour développer, partager et exécuter des composants de workflows scientifiques en tant que services dans le cloud. Dans cette thèse, nous nous intéressons principalement au cloud IaaS, qui offre des services de base (calcul, stockage, ...) nécessaires pour l'exécution des workflows scientifiques.

### I.4.5 Cloud IaaS et les workflows scientifiques

Les clouds IaaS offrent plusieurs avantages pour les applications de type data-intensive. Ces avantages facilitent :

1. l'approvisionnement en ressources ;
2. l'allocation de ressources à la demande ;
3. l'élasticité ;
4. la garantie des QoS via des SLA ;
5. le faible coût d'exploitation.

#### I.4.5.1 Approvisionnement en ressources

Dans les grilles de calcul, pour ordonnancer un ensemble de tâches, l'utilisateur spécifie la quantité de ressources et le temps d'utilisation de celles-ci (modèle en best-effort). Il affecte par la suite à l'ordonnanceur qui fonctionne en affectant les tâches par lot aux ressources et utilisant des files d'attentes. Par contre dans le cloud computing, l'utilisateur peut lui-même approvisionner les ressources nécessaires et ordonnancer les tâches par un ordonnanceur qu'il contrôle, au lieu de laisser l'allocation au gestionnaire de ressources. Ce type d'approvisionnement est particulièrement idéal pour les applications parallèles et data-intensive (workflows scientifiques), car cela permet au système de gestion de workflow d'acquérir une ressource une seule fois et de l'utiliser pour exécuter plusieurs tâches de l'application scientifique.

#### I.4.5.2 Allocation de ressources à la demande

Contrairement aux grilles de calcul, le cloud donne l'illusion que les ressources informatiques (ressources de calcul, de stockage, de bande passante, etc.) en particulier du cloud IaaS, disponibles sont illimitées. Cela signifie que les utilisateurs peuvent demander et s'attendre à obtenir des ressources suffisantes pour leurs besoins, à tout moment. Il existe deux façons d'approvisionnement :

— dynamique : allocation au fur et à mesure que le besoin se fait sentir ;

— statiquement : réserver à l'avance.

L'approvisionnement statique est idéal pour exécuter les workflows, car il réduit le surcoût (*overheads*) lié au démarrage et à l'arrêt des machines pendant l'ordonnancement. Cela permet d'améliorer considérablement les temps d'exécution du workflow [93, 94] et éviter les temps d'inactivité des machines louées. L'approvisionnement dynamique présente également des avantages qui permettent d'éviter de retarder l'exécution d'une tâche prête, s'il n'y a pas de ressources disponibles.

#### I.4.5.3 Élasticité

Outre l'approvisionnement des ressources à la demande, le cloud permet aussi aux utilisateurs de libérer des ressources à la demande. La nature élastique du cloud facilite le changement des quantités et des caractéristiques de ressources lors de l'exécution des workflows. Cette élasticité permet ainsi d'augmenter le nombre de ressources, quand il y a un grand besoin c'est-à-dire lorsqu'il y a plus de tâches à exécuter et d'en diminuer, lorsqu'il y a moins de tâches à exécuter. Cela permet aux systèmes de gestion de workflow de répondre facilement aux exigences de qualité de service (QoS) des applications, contrairement à l'approche traditionnelle qui nécessite de réserver à l'avance des ressources dans les environnements de grilles.

#### I.4.5.4 Garantie des QoS via des SLA

Avec l'arrivée des services du cloud computing proposés par de grandes organisations commerciales telles que Amazon, Google, etc. les accords de niveau de service (SLA) ont été une préoccupation importante pour les fournisseurs et les utilisateurs. En raison de compétitions entre les fournisseurs de services émergents, un grand soin est pris lors de la conception du SLA qui vise à offrir (i) de meilleures garanties de QoS aux utilisateurs et (ii) des termes clairs pour l'indemnisation en cas de violation du contrat. Cela permet aux systèmes de gestion de workflow de fournir de meilleures garanties de bout en bout en "mappant" chaque tâche du workflow scientifique des utilisateurs sur les services cloud selon leurs caractéristiques et les termes de contrat qui leur sont rattachés.

#### I.4.5.5 Faible coût d'exploitation

Économiquement motivés, les fournisseurs de cloud commercial s'efforcent d'offrir de meilleures garanties de services par rapport aux fournisseurs de grille. Les fournisseurs de cloud profitent également des économies d'échelle, en fournissant des ressources de calcul, de stockage et de bande passante, à un coût très faible grâce à la virtualisation. Ainsi, l'utilisation des services de cloud public pourrait être économique et une alternative moins coûteuse par rapport à l'utilisation de ressources dédiées qui sont plus chères. Un des avan-

tages de l'utilisation des ressources virtuelles pour l'exécution de workflow, plutôt que d'un accès direct à la machine physique, est le besoin croissant pour sécuriser les ressources physiques des codes malveillants. Cependant, l'effet à long terme de l'utilisation de ressources virtuelles dans les clouds qui partagent efficacement une "tranche" de la machine physique, au lieu d'utiliser des ressources dédiées pour les workflows de calculs intensifs, est une question de recherche intéressante.

## I.5 Conclusion partielle

Dans ce chapitre, nous avons présenté une vue d'ensemble sur les ressources de calcul et en particulier du cloud computing, du workflow et des intérêts de celui-ci pour les workflows scientifiques qui sont des applications à forte consommation de données (ou intensive en données). Nous avons examiné divers défis de recherche dans le cloud. Un de ces défis porte sur l'ordonnancement de workflow qui présente un élément essentiel des systèmes de gestion de workflows. En général, l'ordonnancement de workflows est de nature multi-objectifs et NP-complet qui nécessite d'utiliser des méta-heuristiques et des heuristiques spécifiques pour sa résolution.

Dans le chapitre suivant, nous présentons les principes de base des méta-heuristiques et heuristiques suivis d'un état de l'art des différents algorithmes d'ordonnancement de workflows dans le cloud utilisant ces deux approches différentes.

Chapitre II

# Algorithmes d'ordonnancement de workflows dans le cloud

## Sommaire

## II.1 Introduction partielle

Les workflows ont été adoptés comme mécanisme attrayant de représention des applications scientifiques intensives en données. Les workflows sont utilisés dans plusieurs domaines pour représenter des applications, où les tâches sont liées par des dépendances de données et la représentation de ces applications se fait sous forme de DAG. Les workflows scientifiques sont des applications de calcul à haute performance, nécessitent beaucoup de calculs et de données ainsi q'un environnement HPC pour leur exécution. Le cloud prend en charge les workflows scientifiques tout en offrant des ressources indispensables à leur exécution sous forme d'instance de calcul, de stockage et de réseau.

Le processus d'affectation d'une tâche à une instance pour son exécution est appelé ordonnancement de workflow (*scheduling workflow*). L'ordonnancement de workflow sur les ressources cloud s'est avéré être un problème NP-Difficile [95] compte tenu de la complexité du DAG. Ce DAG définit l'application parallèle à travers les dépendances qui existent entre les différentes tâches [96]. Cette complexité est due également à la diversité des ressources existantes où il faille trouver la meilleure ressource pour l'exécution d'une ou plusieurs tâches du workflow. Il est impossible de trouver la solution optimale globale en utilisant des algorithmes basiques. Par conséquent, ce problème a un enjeu majeur qui est la nécessité de trouver une manière efficace pour évaluer tous ces différents objectifs à l'aide de (méta) heuristiques.

Les applications scientifiques en pleine croissance ont besoin de ressources pour exécuter les différentes tâches dans un délai et un budget raisonnable. Dans un environnement cloud, les utilisateurs veulent terminer l'exécution de leur application dans un délai spécifié avec un coût minimum. Il existe différents fournisseurs de service cloud (*CSP : Cloud Service Providers*) qui proposent des services différents et prennent en charge divers paramètres (le budget, le délai, l'énergie, la sécurité, la tolérance aux pannes) [51, 53, 87]. Ces paramètres jouent un rôle crucial dans l'ordonnancement de workflow sur un environnement cloud.

Dans ce chapitre nous présentons les différents types d'algorithmes d'ordonnancement qui existent avec les différentes approches utilisées dans chaque type que les chercheurs ont proposé pour atteindre un certain nombre d'objectif. A la section II.2 nous abordons des méta-heuristiques où nous débutons en présentant et en donnant le principe de fonctionnement des méta-heuristiques à la section II.2.1, ensuite nous présentons des algorithmes d'ordonnancement basé sur les algorithmes génétiques (AG) à la section II.2.2, enfin nous donnons quelques algorithmes basé sur l'approche de la colonie de fourmis à la section II.2.3 et l'optimisation des essaims de particule (*PSO : Particle Swam Optimization*) à la section II.2.4. Quant à la section II.3, nous présentons les heuristiques où nous donnerons

d'abord les principes (section II.3.1), ensuite nous donnons quelques algorithmes de *clustering* (II.3.2), enfin nous parlons des algorithmes de *list scheduling*. Ce chapitre se termine par positionner notre travail.

## II.2 Méta-heuristiques

La méta-heuristique peut être regroupée en plusieurs familles, chacune d'entre elles étant basée sur une idée sous-jacente de la manière de mener le processus de recherche. Cette section énumère les différentes familles les plus courantes de la littérature qui permettent de résoudre le problème d'ordonnancement [97]. Ces familles sont les exemples les plus courants dans la littérature et elles englobent une série de caractéristiques majeures des algorithmes telles que (i) les méthodes basées sur une solution ou une population de solutions, (ii) la mémoire et l'apprentissage, (iii) les opérations stochastiques et déterministes.

Le Tableau II.1 présente une brève introduction catégorisant les différents types de méta-heuristique adaptées à l'ordonnancement de workflow dans un cloud. Ces algorithmes évolutionnaires sont capables de proposer une ou plusieurs solutions, selon les approches utilisées dans la littérature. Partant des problèmes mono-objectifs aux problèmes multi-objectifs, ces algorithmes peuvent offrir des solutions adaptées aux besoins des utilisateurs, selon leur(s) objectif(s), qui peut(vent) être la minimisation du makespan ou du coût d'utilisation des ressources cloud, selon le caractère urgent de la réception des résultats, ou même de la minimisation du makespan et du coût d'utilisation des ressources.

### II.2.1 Définition et principe

Le calcul de solutions optimales est insoluble pour de nombreux problèmes d'optimisation d'importance industrielle et scientifique. En pratique, on se contente généralement de "bonnes" solutions, obtenues par des algorithmes heuristiques ou méta-heuristiques. Les méta-heuristiques représentent une famille de techniques d'optimisation approximative qui ont acquis une grande popularité au cours des dernières décennies. Elles font partie des techniques les plus prometteuses et les plus réussies. Les méta-heuristiques fournissent des solutions "acceptables" dans un délai raisonnable pour résoudre des problèmes difficiles et complexes en science et en ingénierie. Ceci explique l'intérêt croissant porté aux approches méta-heuristiques. Contrairement aux algorithmes d'optimisation exacts, les méta-heuristiques ne garantissent pas l'optimalité des solutions obtenues. Contrairement aux algorithmes d'approximation, les méta-heuristiques ne définissent pas à quel point les

Tableau II.1 – Approches et références basées sur les méta-heuristiques.

| Approches | Références |
|---|---|
| GA | Szabo et Kroeger [98] |
| | Verma et Kaushal [99] |
| | Singh et Singh [100] |
| | Verma et Kaushal [101] |
| | Meena *et al.* [102] |
| | Zhu *et al.* [103] |
| ACO | Li *et al.* [104] |
| | Lu et Gu [105] |
| | Nishant *et al.* [106] |
| | Mod et Bhatt [107] |
| | Kushwah et Goyal [108] |
| PSO | Rodriguez et Buyy [109] |
| | Verma et Kaushal [110] |
| BAT | Kaur et Singh [111] |

solutions obtenues sont proches des solutions optimales.

#### II.2.1.1 Définition

Les méta-heuristiques ont reçu une popularité importante durant ces dernières années. Elles s'inspirent en généralement des phénomènes naturels, c'est-à-dire de la biologie, du comportement des animaux, etc. Une méta-heuristique est un algorithme visant à résoudre des problèmes d'optimisation difficiles pour lesquels on ne connaît pas de méthode classique plus efficace. Les méta-heuristiques sont généralement des algorithmes stochastiques itératifs, qui progressent vers un optimum global, c'est-à-dire l'extremum global d'une fonction par échantillonnage d'une fonction objectif. Elles se comportent comme des algorithmes de recherche tentant d'apprendre les caractéristiques d'un problème afin d'en trouver une approximation de la meilleure solution (d'une manière proche des algorithmes d'approximation).

Il existe un grand nombre de méta-heuristiques différentes, allant de la simple recherche locale à des algorithmes complexes de recherche globale. Ces méthodes utilisent cependant un haut niveau d'abstraction, leur permettant d'être adaptées à une large gamme de problèmes différents [112].

#### II.2.1.2 Principe

Les algorithmes méta-heuristiques sont une sorte d'algorithme heuristique générique. C'est la combinaison de la recherche aléatoire et de la recherche locale. Comparée à d'autres heuristiques, la méta-heuristique présente les deux caractéristiques suivantes : (i) le principe d'optimisation de l'algorithme méta-heuristique ne repose pas beaucoup sur les informations de structure organisationnelle de l'algorithme et peut être largement appliqué aux problèmes d'optimisation des combinaisons et autres calculs scientifiques et (ii) la recherche aléatoire est introduite pour rendre les solutions plus diverses. L'algorithme méta-heuristique classique imite principalement les phénomènes naturels, incluant principalement l'optimisation de l'essaim de particules (*Particle Swarm Optimization* : PSO), l'algorithme génétique (*Genetic Algorithm* : GA), l'optimisation des colonies de fourmis (*Ant Colony Optimization* : ACO), etc.

Contrairement aux méthodes exactes, les méta-heuristiques permettent de s'attaquer aux problèmes de grande taille en apportant des solutions satisfaisantes dans un délai raisonnable. Il n'y a aucune garantie de trouver des solutions optimales globales ou même des solutions bornées. Leur utilisation dans de nombreuses applications démontrent leur efficacité et leur efficience pour résoudre des problèmes importants et complexes. L'application de la méta-heuristique s'inscrit dans un grand nombre de domaines. Certains d'entre eux sont :

— la conception technique, l'optimisation de la topologie et l'optimisation structurelle dans les domaines de l'électronique, l'aérodynamique, la dynamique des fluides, les télécommunications,l'automobile et la robotique ;
— l'apprentissage automatique et l'exploration de données en bio-informatique et en finance ;
— la modélisation, la simulation et identification de systèmes en chimie, physique et biologie, le traitement du signal et des images ;
— la planification des problèmes de routage, la planification des robots, les problèmes de planification et de production, la logistique et le transport, la gestion de la chaîne logistique, les problèmes d'ordonnancement, etc.

Pour concevoir une méta-heuristique, deux critères contradictoires doivent être pris en compte : l'exploration de l'espace de recherche (diversification) et l'exploitation des meilleures solutions trouvées (intensification) (Figure II.1). Les régions prometteuses sont déterminées par les "bonnes" solutions obtenues. Dans le cadre de l'intensification, les régions prometteuses sont explorées plus en profondeur dans l'espoir de trouver de meilleures solutions. Dans la diversification, les régions non explorées doivent être visitées pour s'assurer que toutes les régions de l'espace de recherche sont explorées de manière égale et que

la recherche ne se limite pas à un nombre réduit de régions. Dans cet espace de conception, les algorithmes de recherche extrêmes en termes d'exploration (resp. d'exploitation) sont la recherche aléatoire (resp. la recherche locale d'amélioration itérative). Dans la recherche aléatoire, à chaque itération, on génère une solution aléatoire dans l'espace de recherche. Aucune mémoire de recherche n'est utilisée. Dans l'algorithme de base de la recherche locale la plus raide, à chaque itération, on sélectionne la meilleure solution de voisinage qui améliore la solution actuelle. De nombreux critères de classification peuvent être utilisés pour la méta-heuristique :

— **Algorithme inspiré de la nature versus algorithme non-inspiré de la nature** : de nombreuses méta-heuristiques sont inspirées par des processus naturels : algorithmes évolutifs et systèmes immunitaires artificiels issus de la biologie ; colonies de fourmis, d'abeilles et optimisation des essaims de particules à partir de l'intelligence des essaims dans différentes espèces (sciences sociales) ; et recuit simulé issu de la physique.

— **Algorithme utilisant de la mémoire versus algorithme sans mémoire** : certains algorithmes méta-heuristiques sont sans mémoire, c'est-à-dire qu'aucune information extraite dynamiquement n'est utilisée pendant la recherche. Les représentants de cette classe sont la recherche locale et le recuit simulé. D'autres méta-heuristiques, par contre, utilisent une mémoire qui contient certaines informations extraites pendant la recherche. Par exemple, les mémoires à court et long terme dans la recherche tabu.

— **Algorithme déterministe versus algorithme stochastique** : une méta-heuristique déterministe résout un problème d'optimisation en prenant des décisions déterministes (par exemple, recherche locale, recherche tabou). Dans la méta-heuristique stochastique, certaines règles aléatoires sont appliquées pendant la recherche (par exemple , le recuit simulé, les algorithmes évolutifs). Dans les algorithmes déterministes, l'utilisation de la même solution initiale conduira à la même solution finale, alors que dans la méta-heuristique stochastique, différentes solutions finales peuvent être obtenues à partir de la même solution initiale. Cette caractéristique doit être prise en compte dans l'évaluation des performances des algorithmes méta-heuristiques.

— **Algorithme de recherche basé sur une population de solution versus algorithme de recherche basé sur une solution unique** : les algorithmes basés sur une solution unique (par exemple , la recherche locale, le recuit simulé) manipulent et transforment une solution unique pendant la recherche, tandis que dans les algorithmes basés sur la population (par exemple , l'essaim de particules, les algorithmes évolutifs), une population entière de solutions est développée. Ces deux familles ont des caractéristiques complémentaires : les méta-heuristiques basées sur une solution

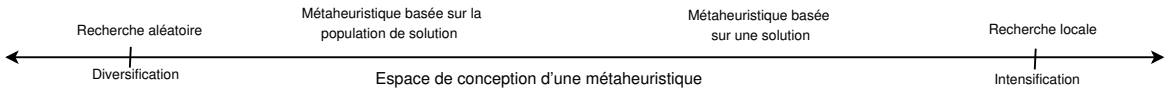



FIGURE II.1 – Critères contradictoires de conception d'une méta-heuristique.

unique sont orientées vers l'exploitation ; elles ont le pouvoir d'intensifier la recherche dans des régions locales. Les méta-heuristiques basées sur une population de solution sont orientées vers l'exploration ; elles permettent une meilleure diversification dans l'ensemble de l'espace de recherche.

— **Algorithme itératif versus algorithme *glouton*** : dans les algorithmes itératifs, nous partons d'une solution complète (ou population de solutions) et la transformons à chaque itération à l'aide de quelques opérateurs de recherche. Les algorithmes *glouton* partent d'une solution vide, et à chaque étape une variable de décision du problème est affectée jusqu'à l'obtention d'une solution complète. La plupart des méta-heuristiques sont des algorithmes itératifs.

#### II.2.1.3 Pourquoi utiliser une méta-heuristique

Il est inutile d'utiliser une méta-heuristique pour résoudre des problèmes lorsque des algorithmes exacts efficaces sont disponibles. Un exemple de cette classe de problème est la classe $P$ des problèmes d'optimisation. Par exemple, il n'est pas nécessaire d'utiliser une méta-heuristique pour trouver un arbre de recouvrement minimum ou un chemin le plus court dans un graphe. Il existe des algorithmes connus d'exactitude du temps polynomial pour ces problèmes.

Par conséquent, pour des problèmes d'optimisation faciles, les méta-heuristiques sont rarement utilisées. Malheureusement, on peut voir de nombreux ingénieurs et même des chercheurs résoudre des problèmes d'optimisation polynomiale avec une méta-heuristique. La première ligne directrice pour résoudre un problème est donc d'analyser d'abord sa complexité.
De nombreux problèmes d'optimisation combinatoire appartiennent à la classe de problèmes NP-difficile. Cette classe de problèmes d'optimisation complexes et de grandes dimensions se pose dans de nombreux domaines d'intérêt industriel : télécommunications, biologie computationnelle, transport et logistique, planification et fabrication, ordonnancement de tâche, etc. De plus, la plupart des problèmes classiques d'optimisation sont NP-difficile dans leur formulation générale [95].

Pour un problème NP-difficile où les algorithmes exacts ne peuvent pas résoudre le problème (compte tenu de la taille et de la structure) dans le temps de recherche requis, l'uti-

lisation de la méta-heuristique est justifiée. Pour cette classe de problèmes, les algorithmes exacts nécessitent (dans le pire des cas) un temps exponentiel. La notion de “temps nécessaire” dépend du problème d’optimisation de la cible. Pour certains problèmes, un temps ”acceptable” peut être équivalent à quelques secondes alors que pour d’autres problèmes, il est égal à quelques mois. Le fait qu’un problème ne se situe pas dans la classe $P$ n’implique pas que tous les grands exemples du problème sont difficiles ou même que la plupart d’entre eux le sont. L’exhaustivité de caractère $NP$ d’un problème n’implique rien quant à la complexité d’une catégorie particulière d’instances qui doivent être résolues.

### II.2.2 Algorithmes d’ordonnancement basés sur l’Algorithme Génétique

Sachant que l’ordonnancement de workflows est un problème NP-difficile, il est quasiment impossible de résoudre un tel problème avec des méthodes exactes. L’utilisation des algorithmes génétiques (AG) peut s’avérer efficace pour résoudre ce problème.
La Figure II.2 représente les différentes étapes d’un algorithme génétique et le plus important est de savoir comment adapter le vocabulaire à un problème spécifique. Dans cette thèse, il est question d’ordonnancer les tâches d’une application scientifique sur des ressources du cloud computing. Pour ce faire, proposer un algorithme génétique dans ce domaine requiert la définition de la population initiale, des chromosomes, de la mutation, et bien évidemment de la fonction objectif.

Plusieurs études de la littérature se sont basées sur ce type d’algorithme pour proposer une (ou plusieurs) solution(s) au problème d’ordonnancement des tâches d’un workflow sur des ressources du cloud computing approvisionnées à la demande.

En général, un utilisateur qui souhaite exécuter une application en temps réel sur une plateforme où il doit payer les ressources utilisées, veut obtenir le résultat de l’application dans la limite de son budget. Par contre, pour les entreprises, le plus important est le caractère urgent d’obtention des résultats et ne peuvent se permettre un budget restreint pour exécuter leur application. De nombreuses études ont été réalisées pour résoudre efficacement le problème d’ordonnancement de workflow sous une fonction mono-objectif qui consiste à minimiser soit le makespan [99, 100], soit le coût des ressources [101, 102] ou par une approche multi-objectif qui consiste à minimiser les deux (makespan et coût) en même temps [98, 103].

Les travaux de Verma et Kaushal [99] et Singh et Singh [100] ont permis de proposer un ordonnancement dont l’objectif est de minimiser le makespan sous une contrainte budgétaire. L’approche proposée par Verma et Kaushal est basée sur la priorité des tâches pour diversifier la population initiale. Cette priorité est calculée par le principe de $bottom\ level$

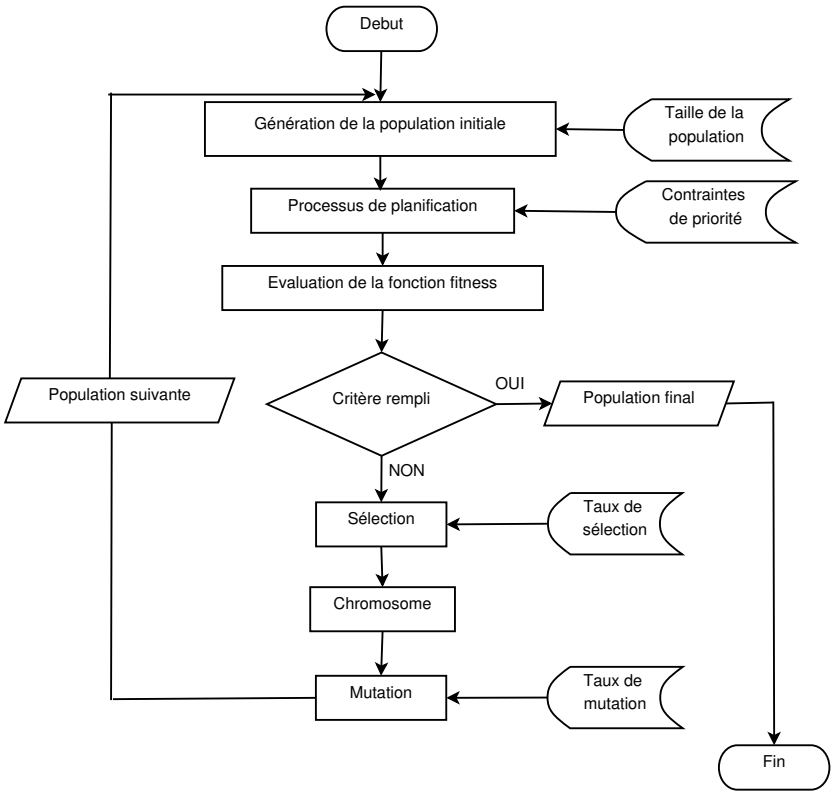



FIGURE II.2 – Schéma général de l'algorithme génétique.

et *top level* basée respectivement sur les études de Topcuoglu *et al.* [113] que nous verrons plus en détail au chapitre III.

Après avoir calculé les *bottom level* et *top level* de chaque tâche et connaissant les différentes ressources du cloud, l'algorithme estime le temps de calcul de chaque tâche sur les différentes machines virtuelles, puis chaque tâche (dans l'ordre de la priorité) est affectée à l'une des machines virtuelles disponibles qui la terminera au plutôt. Tant que le critère d'arrêt défini dans la fonction objectif n'est pas atteint, l'algorithme évalue l'adéquation de l'individu (un ordonnancement) dans la population (ensemble de solution : ordonnancements), puis applique l'opérateur de sélection pour sélectionner le parent (meilleur ordonnancement). Ensuite, l'algorithme applique l'opérateur de croisement (modification aléatoire du mapping) sur le parent sélectionné en utilisant la probabilité de croisement pour créer les enfants (nouveaux ordonnancements) et applique l'opérateur de mutation (mise à jour du mapping) sur les enfants nouvellement créés. Ce nouvel ordonnancement est par la suite validé en fonction de la fonction objectif pour en créer une nouvelle population. Tout comme l'étude menée dans [99], l'approche proposée par [100] est basée sur les algorithmes génétiques et vise la réduction du taux d'échec des ordonnancements (sur des ressources peu fiables du cloud) afin d'offrir un meilleur makespan sous une contrainte budgétaire. Pour pallier ce problème, les auteurs proposent la réplication des tâches. Pour ce faire, ils essaient de trouver un compromis entre le facteur de duplication et le facteur

de resoumission, de sorte que pour un ordonnancement si une tâche a été répliquée sur une autre machine que là où elle a été prévue et qu'elle doit s'exécuter sur cette dernière, le délais de resoumission ne doit pas être élevé. La réplication des tâches est effectuée dans la phase de planification (ordonnancement hors ligne), tandis que la resoumission est effectuée dans la phase d'exécution.

Le cloud IaaS dispose d'une panoplie de ressources pour exécuter les différentes tâches d'une application parallèle. Certaines études de la littérature ont permis de minimiser le coût d'exploitattion des ressources cloud sous contrainte de délais, car celles-ci sont accessibles à la demande et facturées à l'usage. Selon Meena *et al.* [102], en raison des différentes caractéristiques du cloud, les opérateurs génétiques existants de l'algorithme génétique, c'est-à-dire encodage binaire, encodage à valeur réelle ne peuvent pas être directement appliqués au problème d'ordonnancement du workflow dans le cloud. En considérant toutes les caractéristiques du problème d'ordonnancement du workflow dans le cloud, ces auteurs ont présenté un ensemble complet des opérations de l'algorithme génétique, y compris l'encodage, l'initialisation de la population, la mutation et le croisement, l'objectif est de proposer un algorithme générique qui minimise le coût d'utilisation des ressources cloud avec la contrainte de délai. Ils ont utilisé la stratégie de gestion de délais proposé par Deb *et al.* [114], qui consiste à choisir parmi les solutions réalisables, celles qui ont le plus petit coût d'exécution. Cependant, si une solution est faisable et qu'une autre ne l'est pas, la deuxième est ignorée au profit de la première. Si toutes les solutions sont irréalisables, les solutions sélectionnées sont celles qui ont le plut petit temps d'exécution et inférieur au délais. Pour calculer le temps d'exécution de l'application, l'approche proposée prend en compte le temps d'exécution de chaque tâche et le temps de transfert de données d'une tâche à une autre pour les tâches dépendantes.

Minimiser à la fois le makespan et le coût qui sont deux objectifs contradictoires pour le problème d'ordonnancement dans le cloud, est un défi. Pour relever ce défi, certains auteurs proposent généralement une approche multi-objectif. Pour résoudre ce problème, la fonction objectif définie permet d'évaluer la qualité des solutions réalisables. Ainsi la fonction objectif définie par Natesan et Chokkalingam [115] illustrée par l'Eq. II.1 permet de trouver des solutions presque optimales, où $0 < \alpha, \beta \leq 1$. L'estimation du makespan est fonction du temps d'achèvement de la dernière tâche du workflow et le coût est fonction du coût de la ressource de calcul, du service de stockage et de la bande passante. A chaque itération de l'algorithme, les solutions réalisables sont celles dont les valeurs des fonctions objectifs sont inférieurs aux solutions à l'itération précédente, sachant qu'une solution est le mapping obtenu pour chaque tâche du workflow et le coût correspondant à l'utilisation des ressources nécessaires à l'exécution de chacune des tâches. Natesan et Chokkalingam

ont évalués leur approche aux algorithmes FCFS et Min-Min qui sont des algorithmes bien connus dans le domaine de l'ordonnancement, mais ne sont pas les plus adaptés pour l'ordonnancement des tâches dépendantes.

$$FitnessFunction(x_i^t) = \alpha \times Makespan + \beta \times TotalCost \quad \text{(II.1)}$$

L'utilisation de l'algorithme génétique dans le problème de l'ordonnancement, permet d'avoir un ensemble de solutions diversifiées. Cette méthode basée sur un phénomène aléatoire ne permet pas toujours d'avoir une solution optimale, mais améliore les solutions obtenues d'une génération à une autre. La complexité algorithme de cette méthode peut exploser compte tenu du nombre de tâches du workflow (des milliers) et de la diversité des ressources du cloud à utiliser.

### II.2.3 Algorithmes d'ordonnancement basés sur l'algorithme de colonie de fourmis

L'algorithme de colonie de fourmis (*ACO : Ant Colony Optimization*) est une méta-heuristique basée sur le comportement en matière de recherche de nourriture des colonies de fourmis. Au début, les fourmis vont chercher leur nourriture dans des directions aléatoires. Lorsqu'une fourmi trouvera le chemin qui mène à la source de nourriture, une substance chimique appelée phéromone sera laissée par celle-ci en retournant dans son nid. La densité de la phéromone s'évaporera si le chemin menant à la source de nourriture est beaucoup plus long. Si la densité de phéromones est élevée, cela indique que de nombreuses fourmis ont utilisé ce chemin et que, par conséquent, les fourmis suivantes utiliseront ce chemin particulier. Par contre, si la densité de phéromones est faible, cela signifie que le chemin menant à la source de nourriture est beaucoup plus long et que peu de fourmis ont utilisé ce chemin et que ce dernier sera rejeté. Le chemin avec la plus haute densité de phéromones sera sélectionné comme solution optimale.

La méthode ACO est utile pour résoudre des problèmes d'optimisations discrètes qui doivent trouver des chemins vers des objectifs. Elle a été appliquée avec succès pour résoudre le problème du voyageur de commerce, le problème multidimensionnel du sac à dos, l'ordonnancement des tâches en grille de calcul et en cloud. La première étape pour résoudre un problème à l'aide de la méthode ACO est de faire correspondre le système de fourmi au problème donné.

Pour l'ordonnancement de tâches dépendantes sur les ressources cloud, la difficulté est de faire correspondre le vocabulaire de l'ACO et les éléments de l'ordonnancement (tâches, ressources, etc.).

Un bon ordonnanceur de tâches doit adapter sa stratégie d'ordonnancement à l'environnement d'exécution changeant et aux tâches. Par conséquent, un algorithme d'ordonnancement de tâches, tel que l'ACO est approprié pour les ressources cloud. Li *et al.* [104] utilisent dans leur étude les caractéristiques de l'ACO mentionnés ci-dessus pour ordonnancer les tâches du workflow. Le principe de leur approche est d'améliorer les ordonnancements à partir de ceux obtenus précédemment, dans le but de minimiser le temps d'exécution du workflow. Initialement, pour construire une solution (ordonnancement), les tâches (fourmis) sont réparties sur les VMs de façon aléatoire. La valeur de la phéromone est mise à jour en fonction du taux d'utilisation de chaque VM (*load balancing*). La probabilité qu'une tâche puisse être mappée sur une VM est fonction de la valeur de phéromone de cette VM et de son *load balancing*. Contrairement à Li *et al.* [104] qui distribue de façon aléatoire les tâches sur les machines, Lu et Gu [105] proposent une approche qui repartie de façon équitable le nombre de tâche sur les machines disponibles. La mise à jour de la valeur de la phéromone est faite en fonction du temps d'achèvement des tâches mappées sur chaque VM. Ainsi, les VMs prioritaires d'une itération à une autre, sont celles dont le temps d'achèvement est le plus petit et ayant un *load balancing* plus élevé. Cette stratégie permet au processus de recherche de converger rapidement vers les "bonnes" ressources et d'atteindre rapidement l'objectif recherché. L'approche proposée par Nishant *et al.* [106] est une version améliorée de l'algorithme présenté dans [116]. Les deux algorithmes utilisent le comportement des fourmis pour collecter des informations sur les VMs afin d'affecter chaque tâche à une VM spécifique. Cependant, l'algorithme de [116] a le problème de synchronisation des fourmis et l'auteur de [106] essaie de résoudre ce problème en ajoutant la fonction "suicide" aux fourmis. Les deux algorithmes fonctionnent de la manière suivante, une fois qu'une recherche de solution (ordonnancement) est lancée, les fourmis (tâches) commencent par chercher une solution à partir d'une VM (nœud) "maître" en effectuant toutes les combinaisons possibles sur la base d'une liste de VM disponible. La valeur des phéromones est mise à jour en tenant compte du *load balancing* de la VM et du temps d'achèvement des tâches qui lui sont affectées. Le but de l'étude de Nishant *et al.* [106] est la minimisation du makespan par la recherche optimale de ressources cloud pour l'exécution de l'ensemble des tâches d'une application workflow en tenant compte du *load balancing* de chaque VM afin d'avoir un bon taux d'utilisation des ressources. Mod et Bhatt. [107] ont proposé un algorithme d'ordonnancement basé sur l'ACO pour l'ordonnancement et la gestion des ressources cloud à un seul niveau. Dans la phase d'initialisation, des ressources de types différents sont générées de manière aléatoire et la matrice de tâches totale et de ressources est déterminée. Dans la phase d'amélioration de la solution, la valeur de la fonction obectif pour chaque ressource est calculée et finalement la meilleure d'entre elles est sélectionnée.

Kushwah et Goyal [108] ont proposé une approche basée sur l'ACO qui permet d'avoir

pour une tâche donnée, plusieurs mappings possibles. Le but de ce principe est de résoudre le problème de tolérance aux fautes qui peut subvenir pendant l'exécution du workflow. Afin d'augmenter la performance de leur algorithme, les auteurs ont ajoutés une fonction d'équilibrage de charge (*load balancing*).

Contrairement aux algorithmes génétiques qui adoptent un processus aléatoire pour proposer des solutions d'ordonnancement, les algorithmes de colonie de fourmis utilisent un processus probabiliste pour résoudre le problème d'ordonnancement. Ce processus probabiliste peut entraîner un temps non polynomial lors de la recherche de solutions. Les algorithmes de colonie de fourmis sont très peu exploités dans le domaine d'ordonnancement de tâches sur des ressources du cloud computing.

### II.2.4 Algorithmes d'ordonnancement basés sur l'organisation par essaim de particule

L'organisation par essaim de particule (PSO : *Particules Swarm Organisation*) est une classe d'algorithme méta-heuristique adaptée à l'ordonnancement des applications parallèles dans le cloud, dont l'objectif est de planifier les tâches du workflow sur les ressources du cloud computing afin de réduire le temps de calcul et le coût d'utilisation des ressources. Avec les algorithmes PSO, chaque particule a sa propre *valeur de fitness* et est évaluée par la *fonction fitness* pour obtenir un résultat optimisé.

Ce type d'algorithme est auto-adaptatif basé sur une population de recherche globale sans aucune recombinaison directe [117]. L'algorithme PSO générera un certain nombre de solutions candidates connues sous le nom de particules, lesquelles seront ensuite déplacées dans un espace fini. Dans un premier temps, les particules se déplacent dans leur position locale la plus connue, puis dans la meilleure position globale de l'ensemble de la population. Plus tard, ces mouvements sont trouvés par les autres particules et sont appelées essaim (*swarm*) vers les meilleures solutions.

L'algorithme PSO est lancé avec un ensemble de particules (solution : mapping) aléatoires, puis la recherche de l'optima est effectuée en mettant à jour les solutions générées. À chaque itération, chaque particule est mise à jour avec deux meilleures valeurs. La première valeur est la valeur de fitness atteinte jusqu'à présent et elle est qualifiée de $pbes$t. La deuxième valeur est la meilleure valeur de toute particule dans le troupeau et est appelée $gbest$. Après avoir trouvé le $pbest$ et le $gbest$, la vitesse de la particule ($v$) est ajustée à l'aide de l'équation suivante :

$$v(t) = v(t-1) + c1 \times rand \times (pbest - p) + c2 \times rand \times (gbest - p) \qquad \text{(II.2)}$$

où
$p$ = position de la particule
$v$ = vitesse de la particule
$c1$ = poids de l'information locale
$c2$ = poids de l'information globale
$pbest$ = meilleure position des particules
$gbest$ = meilleure position de l'essaim
$rand$ = variable aléatoire

*La fonction fitness* évaluera chaque particule (solution). L'initialisation aléatoire de la position et de la vitesse est effectuée pour les particules. Ensuite, la *valeur fitness* est calculée à l'aide des *fonctions fitness* et la meilleure position locale est trouvée, suivie de la mise à jour. Après avoir calculé les mappings (placements) à l'aide du PSO, l'algorithme d'ordonnancement répartit les tâches prêtes sur les ressources correspondantes. Une tâche prête est une tâche dont les prédécesseurs ont terminé leur exécution et qui a complètement reçu toutes les données de celles-ci. Utiliser les algorithmes PSO pour l'ordonnancement des tâches d'un workflow permettent de réduire le coût d'utilisation des ressources et le temps d'exécution du workflow (makespan). Plusieurs recherches sont portées sur cette approche.

Pandey *et al.* [118] ont proposé une approche basée sur l'optimisation par essaim de particules (PSO) pour l'ordonnancement de workflow sur les ressources du cloud computing. Cette approche tente de minimiser le temps d'exécution de l'application sous une contrainte qui prend en compte à la fois le coût des ressources de calcul et les coûts de transmission de données. Les résultats de l'évaluation montrent que le PSO peut réduire le makespan tout en fournissant un bon équilibrage de charge (workload), c'est-à-dire réduire le temps d'inactivité des ressources sollicitées.

Netjinda *et al.* [119] ont utilisé cette même méta-heuristique pour l'optimisation du coût d'utilisation des ressources. L'algorithme utilise un schéma de décodage pour convertir les valeurs réelles d'une particule en des valeurs entières distinctes représentant une solution. Les premiers résultats montrent une performance prometteuse, au point de vue du coût total et de la convergence de la fonction objectif.

Guo *et al.* [120] ont formulé un modèle pour l'ordonnancement multiobjectif de tâches et ont proposé un algorithme PSO qui optimise le temps d'exécution et le coût d'utilisation des ressources cloud.

Les algorithmes PSO sont les moins onéreux. Ils sont plus simples et les plus faciles à appliquer, avec un très bon taux de convergence par rapport aux algorithmes génétiques et colonie de fourmis. Ils contribuent grandement à améliorer l'efficacité de la planification

des flux de travail dans les environnements cloud.

Les méta-heuristiques étant très généralistes, elles peuvent être adaptées à tous types de problème d'optimisation, mais souvent moins efficaces que les méthodes exactes sur certains types de problèmes d'optimisation et ne garantissent pas la découverte de la (ou des) solution(s) optimale(s) à un temps fini. Cependant, plusieurs problèmes du monde réel ne sont pas de façon efficace par des approches méta-heuristiques. La notion d'efficacité se rapproche très souvent de deux objectifs contradictoires qui sont : (i) la vitesse à trouver une solution optimale, souvent mesurée en nombre d'évaluations de la fonction objectif qui est la plupart du temps la partie la plus gourmande en temps de calcul et (ii) la précision de cette solution, se rapportant à la distance entre la solution optimale trouvée par la méta-heuristique et la solution optimale réelle.

Généralement, un choix doit être fait quant au critère d'arrêt adéquat qui influencera la qualité de la (des) solution(s), et aucune approche méta-heuristique ne peut prétendre être plus meilleure qu'une autre. Toutes les approches ignorent un élément fondamental dans le processus d'ordonnancement de workflows qui est le temps de transfert de données. Pour l'ordonnancement de workflows "data-intensives", un problème est le manque de précision dans la prédiction de la localisation des données et de leur temps de transfert, malgré un temps de calcul long [121]. Ainsi, les heuristiques spécifiques exploitent mieux les dépendances entre les tâches. Dans les sections suivantes, nous exposons les différentes approches existantes dans les heuristiques spécifiques et positionnons notre thèse par rapport à celles-ci.

## II.3 Heuristiques spécifiques

Une méthode heuristique consiste à trouver une ou des solutions optimales d'un problème. Un raisonnement ou une méthode heuristique est une méthode de résolution de problème qui ne s'appuie pas sur une analyse détaillée ou exhaustive du problème [122]. Elle procède par approches successives en éliminant progressivement les solutions alternatives et en ne conservant qu'une gamme restreinte de solutions tendant vers celle qui est optimale.

L'heuristique trouve de "bonnes" solutions sur des problèmes de grandes tailles. Elle permet d'obtenir des performances acceptables dans un temps raisonnable pour un large éventail de problèmes. En général, les heuristiques n'ont pas de garantie d'approximation sur les solutions obtenues.

Contrairement aux méta-heuristiques où une approche permet de résoudre plusieurs problèmes à un problème donné, il peut y avoir plusieurs heuristiques qui proposent des

solutions satisfaisantes c'est-à-dire plusieurs heuristiques différentes permettent de trouver une solution "optimale" pour le même problème. Quant aux problèmes d'ordonnancement qui font l'objet de cette thèse, nous avons étudié plusieurs algorithmes d'ordonnancement basés sur la méthode heuristique que nous avons classée en deux catégories et répertorié dans le Tableau II.2.

Tableau II.2 – Approches et références basées sur les heuristiques spécifiques.

| Approches | Références |
|---|---|
| Clustering | Abrishami *et al.* [123] |
| | Lin *et al.* [124] |
| | Arabnejad *et al.* [125] |
| | Arabnejad *et al.* [126] |
| | Keahey *et al.* [127] |
| | Lin *et al.* [128] |
| | Almi'Ani et Lee [129] |
| List Scheduling | Lin et Lu [130] |
| | El-kenawy *et al.* [57] |
| | Thaman et Singh [131] |
| | Rimal et Maier [132] |
| | Almi'ani *et al.* [133] |
| | Alkhanak et Lee [134] |
| | de Oliveira *et al.* [135] |
| | Fard *et al.* [136] |
| | Durillo et Prodan [121] |
| | Rodriguez et Buyya [137] |
| | Zhou *et al.* [138] |

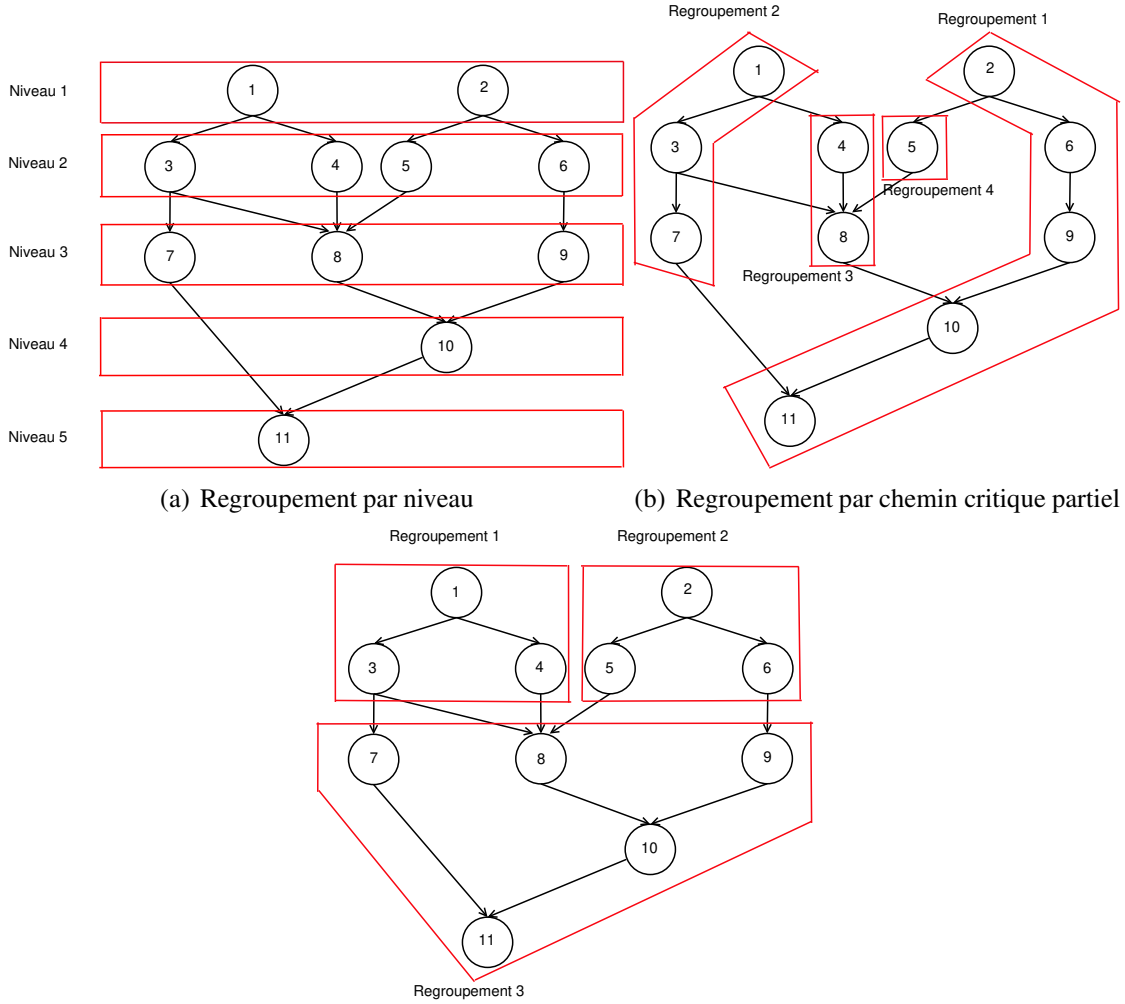
### II.3.1 Définition et principe

En optimisation combinatoire, en théorie des graphes, en théorie de la complexité des algorithmes et en intelligence artificielle, une heuristique est une méthode de calcul qui fournit rapidement c'est-à-dire en temps polynomial une solution réalisable mais pas nécessairement optimale, pour un problème d'optimisation NP-difficile.

Dans le cas du problème d'ordonnancement de tâches de workflow sur des ressources de calcul (qui est une optimisation combinatoire), deux grandes approches ont été proposées dans la littérature : les heuristiques de *clustering* et de *list scheduling*. Les algorithmes de *clustering* sont basés sur des techniques de regroupement comme l'illustrent les Figures II.3(a) - II.3(c) présentant respectivement un exemple de regroupement par niveau, un cas de regroupement par chemin critique partiel et un cas de regroupement selon un ensemble de critères que les auteurs définissent.

Quant aux algorithmes de *list scheduling*, le principe est de maintenir dans une liste triée soit par temps d'exécution, soit par priorité, les différentes tâches du workflow avant de les affecter à des ressources de calcul.

Cette section permet de mettre en exergue quelques heuristiques d'ordonnancement des tâches du workflow scientifique sur des ressources du cloud computing.




(a) Regroupement par niveau

(b) Regroupement par chemin critique partiel

(c) Regroupement quelconque

FIGURE II.3 – Différents types de regroupement.

## II.3.2 Algorithmes d'ordonnancement par "clustering"

Les algorithmes de *clustering* consistent à effectuer un regroupement de tâche selon un certain critère. Dans la littérature, il existe plusieurs types de regroupement, entre autre le regroupement par chemin critique, le regroupement par niveau, etc. La phase de regroupement est l'une des étapes importantes dans le processus d'ordonnancement des tâches. Ainsi, plusieurs auteurs ont basé leur étude sur le principe de *clustering* afin de proposer

des algorithmes d'ordonnancement dans le cloud.

Abrishami *et al.* [139] ont conçu un algorithme d'ordonnancement dénommé *Partial Critical Path* (PCP) initialement prévu pour exécuter des tâches sur des grilles de calcul dans l'optique de réduire le temps d'exécution d'un workflow. Ces mêmes auteurs ont par la suite adapté le PCP pour des ressources IaaS du cloud [123] en proposant deux algorithmes IC-PCP et IC-PCPD2 dont l'objectif est de minimiser le coût d'exploitation des ressources du cloud sous une contrainte de délai. Dans cette étude, après la détermination des différents chemins critiques partiels, les tâches (du même PCP) sont exécutées sur une même ressource cloud. La différence entre ces deux algorithmes est qu'avec IC-PCPD2, un délai est affecté à chaque chemin critique partiel en fonction du délai défini par l'utilisateur.

Lin *et al.* [124], Arabnejad *et al.* [125] ont adapté l'algorithme PCP [139] pour la planification des tâches de workflows scientifiques dans l'environnement cloud. Lin *et al.* [124] ont proposé une stratégie d'ordonnancement utilisant des services du cloud chez différents fournisseurs. L'approche prend en compte certaines caractéristiques essentielles sur les différents clouds considérés, telles que divers types de machines virtuelles, la charge de travail par intervalle de temps de différents fournisseurs de cloud ainsi qu'une bande passante intra-bande (homogène chez chaque fournisseur) et une bande passante inter-bande hétérogène (entre différents fournisseurs). Par rapport au PCP traditionnel, l'approche d'ordonnancement de Lin *et al.* peut compresser et réduire le temps de transfert de données le long du chemin critique partiel, ce qui réduit le temps d'exécution du workflow scientifique.

Quant à Arabnejad *et al.* [125], dans leur étude, proposent de minimiser le coût d'utilisation des ressources cloud sous la contrainte d'un délai. Pour ce faire, les auteurs proposent deux approches : PDC (*Proportional Deadline Constrained*) et DCCP (*Deadline Constrained Critical Path*), qui passent tous deux par une étape de prétraitement consistant d'abord à partitionner le DAG en plusieurs niveaux en fonction des dépendances respectives qui existent entre les tâches. Ensuite, le délai ($T_D$) défini par l'utilisateur est réparti sur les différents niveaux préétablis. Enfin, chaque niveau reçoit son propre délai qui est affecté à toutes les tâches du niveau. Les auteurs utilisent le principe de priorité de HEFT [113] pour construire la liste des tâches critiques et appliquent l'approche DCCP pour l'ordonnancement des tâches appartenant au même chemin critique. Quant aux tâches individuelles, c'est-à-dire qui n'appartiennent à aucune liste de tâches critiques, l'approche PDC est utilisée pour leur ordonnancement. Avec l'approche DCCP, pour chaque liste, une instance (la moins coûteuse) est sélectionnée afin d'exécuter toutes les tâches d'un même chemin critique, favorisant ainsi la réduction des temps de communication généré lors des transferts de données dans le réseau. Selon les auteurs de [125], le modèle de facturation des fournisseurs tels qu'Amazon se fait par tranche d'une heure (60 min), c'est-à-dire lorsqu'une tâche s'exécute sur une instance, l'intégralité de l'intervalle de facturation est facturée, quelle que soit l'utilisation de cette instance. Par conséquent, si d'autres tâches peuvent être exécutées

sur la même instance au cours de cet intervalle facturé, leur coût d'exécution est nul. Pour trouver la meilleure instance dans l'approche DCCP, la priorité est de sélectionner une instance qui a suffisamment de temps libre sur les instances déjà facturées et qui peut exécuter toutes les tâches du chemin critique.

Arabnead *et al.* [126] ont proposé une approche améliorée de l'approche IC-PCP [123]. Pour ces auteurs, Abrishami *et al.* , dans leur approche, ne tiennent pas compte du temps de démarrage et de déploiement des machines virtuelles sur les machines physiques, ce qui peut violer le délai défini par l'utilisateur. Pour pallier ce problème, Arabnead *et al.* ont proposé dans la phase d'assignation de ressources, d'assigner à une instance déjà louée les tâches du PCP si celle-ci peut exécuter toutes les tâches sans violation de délais. Cette proposition permet d'éviter d'assigner les tâches d'un PCP à des instances différentes et donc d'éviter le temps de démarrage et de déploiement de certaines machines virtuelles. Cela a pour avantage de louer moins de machines virtuelles que de PCP effectué, permettant ainsi de minimiser le coût d'exploitation des ressources cloud. Pour Arabnead *et al.* , en plus d'effectuer des regroupements, l'ordre d'exécution des tâches de chaque regroupement est également important, ainsi ils se sont basés sur le principe de calcul de priorité de HEFT pour assigner à chaque tâche un ordre d'exécution.

Le concept de multi-cloud a été introduit pour la première fois par Keahey *et al.* [127] et est relativement naissante dans le domaine de l'ordonnance de workflow. A cet effet, Lin *et al.* [128] ont proposé une toute nouvelle approche dont l'objectif est de minimiser le coût des ressources utilisées chez les fournisseurs différents. Pour ce faire, après avoir obtenu tous les chemins critiques du workflow, la phase d'assignation des ressources consiste à trouver pour chaque PCP la ressource la moins coûteuse pour exécuter les tâches du PCP. Ces ressources de calcul sont obtenues chez les fournisseurs cloud différents c'est-à-dire le fournisseur qui offre le service le moins coûteux pour exécuter les tâches du PCP.

Almi'Ani et Lee [129] ont proposé l'algorithme *Partitioning-Based Workflow Scheduling (PBWS)* qui s'efforce d'établir un mapping efficace en divisant le workflow en groupes de tâches. Cette répartition vise à simplifier le processus d'affectation des ressources aux tâches. Ensuite, sur la base des dépendances des données entre les partitions, la capacité des ressources à affecter à chaque partition est déterminée pour garantir que le temps d'achèvement et le coût d'exploitation des ressources soient minimisés. L'algorithme présenté se compose de trois étapes : (1) l'étape de partitionnement, (2) l'étape d'ajustement de partition, (3) l'étape d'affectation des ressources. Dans l'étape de partitionnement, l'objectif est de partitionner un workflow donné de sorte que la dépendance entre les partitions résultantes assure que ces partitions peuvent être représentées comme un DAG. Cela vise à simplifier le processus d'attribution des ressources car chaque partition sera traitée comme un nœud unique à des étapes ultérieures. Dans l'étape d'ajustement de la partition, certaines tâches sont réorganisées et déplacées vers différentes partitions afin de réduire

encore le temps d'achèvement de l'application. L'étape d'affectation des ressources détermine les correspondances les plus rentables c'est-à-dire moins coûteuse entre les tâches et les ressources garantissant la minimisation du temps d'inactivité totale des ressources.

Avec l'approche par regroupement, il existe toujours des dépendances entre les regroupements constitués pouvant dégrader le temps d'exécution du workflow. L'approche de list scheduling permet de résoudre ce problème en tenant compte de toutes les dépendances qui lient une tâche donnée. Cette approches est la plus adaptée aux problèmes d'ordonnancement résolus dans cette thèse.

### II.3.3 Algorithmes par *"list scheduling"*

Il est question ici de créer une liste d'ordonnancement en attribuant une priorité et en triant les tâches en fonction de leur priorité, puis de sélectionner une tâche et une ressource de manière répétitive jusqu'à ce que toutes les tâches du DAG soient ordonnancées. Cette approche de sélection des ressources et des attributs de priorité est nécessaire pour décider des priorités des tâches et de la ressource optimale pour chaque tâche.

En effet, Topcuoglu *et al.* [113] ont mis au point algorithme dénommé HEFT (*Heterogeneous Earliest-Finish-Time*) ; initialement développé pour les grilles de calculs sur des processeurs hétérogènes non uniformes entièrement connectés dont l'objectif est de réduire le temps d'achèvement (makespan) d'une application parallèle modélisée sous forme de DAG. Le principe de leur étude peut se résumer en trois étapes :

1. l'attribution de priorité à chaque tâche ;
2. le tri par ordre croissant ;
3. l'allocation de ressource.

La première étape consiste à calculer le niveau de priorité de chaque tâche. Cette priorité se calcule en tenant compte du temps d'exécution que met une tâche sur une ressource et du temps de transfert des données que celle-ci met pour transférer l'ensemble des données qu'elle produit en sortie à une autre tâche, si celles-ci s'exécutent sur des ressources différentes. L'estimation du temps de transfert de données d'une tâche vers une autre tâche, si celles-ci s'exécutent sur des machines différentes, est fonction de la moyenne de la bande passante qui lie toutes les machines entre elles. Si deux tâches s'exécutent sur la même machine, le temps de transfert de données est négligeable donc supposé nul. La valeur de priorité de chaque tâche est calculée de manière récursive en parcourant les tâches du DAG du bas vers le haut, en commençant par la tâche de sortie (dernière tâche). Pour la tâche de sortie, la valeur de priorité est égale à son temps d'exécution. Fondamentalement, la valeur de priorité est la longueur du chemin critique d'une tâche à la dernière tâche. La deuxième

étape consiste à générer une liste de tâches triées par priorité décroissante. Il peut exister d'autres stratégies de sélection telle que la sélection aléatoire. On peut facilement montrer que l'ordre décroissant des valeurs de priorité fournit un ordre topologique des tâches qui est un ordre linéaire et préserve les contraintes de priorité et de dépendance. Quant à la troisième étape, elle consiste à exécuter chaque tâche sur le processeur qui peut la terminer au plus tôt. Pour la plupart des algorithmes d'ordonnancement de tâches, la date au plus tôt disponible d'un processeur pour l'exécution d'une tâche est l'instant à laquelle le processeur termine l'exécution de la dernière tâche qui lui est affectée. La particularité de HEFT est d'utiliser des ressources hétérogènes, c'est-à-dire des processeurs ayant des vitesses de calcul différentes. Dans HEFT, la recherche d'un créneau temporel d'inactivité approprié pour l'exécution d'une tâche sur un processeur débute à l'instant auquel cette tâche est prête : c'est-à-dire au moment où toutes les données en entrée ont été envoyées par l'ensemble des tâches immédiatement prédécesseurs à celle-ci sont parvenues au processeur. La recherche se poursuit jusqu'à obtenir un intervalle de temps inactif sur le processeur qui peut terminer la tâche au plus tôt. L'algorithme HEFT [113] est l'un des meilleurs algorithmes qui minimise le makespan sur des ressources de grilles de calcul et a été longtemps étudié et adapté sur plusieurs environnement d'exécution et aujourd'hui sur le cloud computing.

Pour l'ordonnancement d'une application parallèle sur les des ressources du cloud, les ordonnanceurs peuvent prendre en compte différents objectifs et paramètres tels que le temps d'exécution, le coût d'utilisation des ressources, etc. En fonction des objectifs considérés, les algorithmes d'ordonnancement ont été catégorisés dans les sections suivantes.

#### II.3.3.1 Approche mono-objectif

Les algorithmes d'ordonnancement *Best-Effort* tentent d'optimiser un objectif sous la contrainte d'un ensemble de critère. Cette optimisation peut être la minimisation du makespan ou du coût d'utilisation des ressources cloud. L'étude de Rezaeian *et al.* [140] a pour objectif de minimiser le makespan sous la contrainte de budget défini par l'utilisateur sur un environnement cloud hybride, c'est-à-dire utilisant à la fois les ressources du cloud privé et public pour l'exécution du workflow. Cette étude [140] est une adaptation de HEFT [113] sur les ressources cloud et comprend trois étapes principales :

1. l'attribution de priorité et sélection d'instance ;
2. l'attribution de sous-budget ;
3. l'amélioration du makespan tout en respectant le sous-budget.

Dans la première étape, une liste de tâches est constituée en se basant sur la méthode de HEFT [113]. Dans cette étape, les tâches dites "tâches sensibles", c'est-à-dire critiques

(ayant un niveau de priorité élevé), sont affectées aux ressources qui peuvent les terminer le plus rapidement et, par conséquent, les plus chères du cloud public. Cette étape est appelée *fastest scheduling*. Le budget d'exécution du workflow de cet ordonnancement a pour objectif de fournir une vision finale, quel que soit le budget défini par l'utilisateur. Le processus de la première étape est très similaire à la phase de sélection de ressources dans l'algorithme HEFT. Cela signifie qu'à chaque étape, une tâche non mappée (planifiée) ayant la priorité la plus élevée est sélectionnée et est affectée à la ressource sur laquelle elle se termine plus tôt que sur n'importe quelle autres ressources. La seule différence entre cette phase de l'algorithme proposé et l'approche HEFT consiste à prendre en compte les tâches sensibles. Pour mapper les tâches sensibles, l'algorithme proposé sélectionne des ressources uniquement dans le cloud privé. Si l'ordonnancement respecte le budget défini par l'utilisateur, le plan d'ordonnancement est renvoyé comme solution au problème. Sinon, la deuxième phase de l'algorithme démarre et estime les sous-budgets pour les tâches dites "tâches non-sensibles" et les remappe en fonction de ces sous-budgets estimés. Dans la deuxième étape, les auteurs affectent un sous-budget à chaque niveau du DAG. Ces sous-budgets sont calculés en fonction du budget global défini par l'utilisateur et par rapport au budget obtenu dans la première étape, c'est-à-dire sur les ressources les plus rapides. Cependant, ces sous-budgets risquent de fournir un mauvais makespan (plus long). Par conséquent, la troisième étape a pour objectif de réduire le makespan tout en respectant le budget défini par l'utilisateur. Pour ce faire, chaque tâche du DAG est sélectionnée niveau par niveau en commençant par le premier niveau. Pour un niveau donné, les ressources du cloud les plus puissantes sont louées pour exécuter chaque tâche tout en respectant le sous-budget défini pour ce niveau. Pour ce faire, une méthode récursive est proposée, qui vise à séparer les tâches "sensibles" des tâches "non-sensibles". Les tâches sensibles sont par la suite assignées aux ressources d'un cloud privé avec un temps minimal d'achèvement et simultanément, assigner les tâches non-sensibles aux ressources privées s'il y a encore des ressources disponibles, sinon les assigner aux ressources d'un cloud public tout en respectant le sous-budget défini pour ce niveau. Cette étape est effectuée à chaque niveau du DAG et elle garantit que le budget d'ordonnancement est inférieur au budget défini par l'utilisateur.

Lin et Lu [130] ont proposé l'heuristique SHEFT (*Scalable HEFT*) pour ordonnancer un workflow d'une façon élastique sur un environnement de cloud computing. Cet algorithme tente de minimiser le makespan du workflow et gère la scalabilité des ressources à l'exécution. Cette approche ignore les dépendances de données entre tâches et se concentre particulièrement aux données transférées entre les différents regroupement de ressources. El-kenawy *et al.* [57] ont proposé une version améliorée de l'algorithme Max-Min. Cet algorithme utilise le temps d'exécution prévu, au lieu du temps complet, comme base de sélection. Ils ont utilisé les réseaux de Petri qui sont bien adaptés pour la modélisation du

comportement concurrentiel des systèmes distribués. Le résultat montre que cet algorithme réalise l'ordonnancement avec un makespan inférieur à celui de l'algorithme Max-Min original. L'algorithme Max-Min n'est pas adapté à l'exécution de workflows "data-intensives" car il ne prend en compte que le temps de calcul des tâches et ignore le temps de transfert de données entre tâches, or celui-ci est important dans le processus d'ordonnancement de workflows "data-intensives".

Thaman et Singh [131] ont également basé leur étude sur l'algorithme HEFT, pour proposer une approche hybridée à une nouvelle façon de mettre en correspondance un ensemble de tâche avec les machines disponibles dénommé *Interior scheduling*. Pour ces auteurs, l'avantage de HEFT est qu'un graphe de tâches non linéaire est converti en une liste linéaire de tâches et permet d'attribuer une priorité aux tâches du workflow.

L'une des particularités du cloud est de fournir aux utilisateurs des ressources de calcul hétérogènes, exploitables pour les applications parallèles à forte intensité de calcul (*compute-intensive*), c'est-à-dire nécessitant plus de ressources de calcul différentes. A cet effet, Rimal *et al.* [132] ont proposé une approche dont l'objectif est de minimiser le temps d'achèvement de l'application, le retard de lancement de certaines tâches et le temps d'inactivité des machines déjà reversées chez le fournisseur. Ainsi les tâches nécessitant plus de calcul seront affectées aux machines plus puissantes, selon la disponibilité de celles-ci. Dans cette approche, une machine peut être dotée de plusieurs cœurs de calcul qui ne peuvent exécuter qu'une seule tâche à un instant donné.

Outre le makespan, le coût est une métrique de plus en plus importante à prendre en compte dans les environnements de cloud computing . En effet, avec le succès des services cloud, les fournisseurs déploient de plus en plus des infrastructures physiques (Datacenters) qu'ils doivent amortir et même réaliser des bénéfices. Cependant, les stratégies d'ordonnancement doivent adopter des mesures, non seulement pour répondre aux exigences de QoS définies par l'utilisateur via les SLA, mais aussi pour veiller à ce que le profit des fournisseurs ne soit pas considérablement réduit. Par conséquent, des travaux plus récents ont porté sur le développement d'algorithmes d'ordonnancement prenant en compte la consommation énergétique, la tolérance aux fautes, la sécurité du système, etc.

*Dyna* est un framework d'ordonnancement proposé par Zhou *et al.* [141] et prend en compte la nature dynamique des environnements cloud du point de vue des performances et des coûts des ressources. Il est basé sur un modèle de ressources similaire à celui d'Amazon EC2, car il prend en compte à la fois les instances réservées d'avance et les instances obtenues à la demande. Le but est de minimiser le coût d'utilisation des ressources cloud tout en offrant des garanties d'échéances probabilistes qui reflètent la variabilité des performances des ressources et la dynamique des coûts des instances. Les instances réservées d'avance sont utilisées pour réduire le coût de l'infrastructure et les instances à la demande pour respecter les contraintes de délais lorsque les instances réservées ne sont pas capables

de terminer les tâches à temps. Pour ce faire, un plan de configuration hybride statique des instances (une combinaison d'instances réservées et à la demande) est généré pour chaque tâche. Chaque plan de configuration implique un ensemble d'instances réservées et un type d'instance à la demande qui doit être utilisé en cas d'échec de l'exécution sur chacune des instances réservées de l'ensemble de configuration. Au moment de l'exécution, des techniques de consolidation et de réutilisation des instances sont utilisées pour mapper les tâches. Contrairement à la plupart des algorithmes, le framework *Dyna* reconnaît que les estimations statiques du temps d'exécution des tâches et les garanties de performances déterministes ne sont pas adaptées aux environnements cloud. Les auteurs proposent plutôt d'offrir aux utilisateurs une garantie de délai plus réaliste et probabiliste qui reflète la dynamique du cloud. Leurs modèles probabilistes réussissent à saisir la variabilité des performances des entrées/sorties et du réseau.

La majorité des travaux mentionnés ci-dessus se concentre uniquement sur l'optimisation d'une seule métrique à savoir le temps d'achevement du workflow scientifique ou du coût d'exploitation des ressources du cloud computing. Toutefois, avec l'avènement du cloud computing, les fournisseurs de services clouds commerciaux offrent, à la demande, des ressources de calcul ayant des fréquences différentes et des services de stockage, proposés à des prix variés. Les fournisseurs facturent les utilisateurs selon leurs consommations. Par exemple, pour Amazon EC2, qui est un fournisseur de services IaaS du cloud, les utilisateurs sont facturés pour la consommation des ressources de calcul et de stockage. Les utilisateurs peuvent décider de payer un peu plus cher en demandant plus de ressources afin de réduire le temps d'achèvement de leurs workflows scientifiques ou réduire le coût d'exploitation des ressources en autorisant un temps d'achèvement plus long, tant que le délai est respecté. Par conséquent, les algorithmes d'ordonnancement doivent impérativement considérer le budget prévisionnel et le délai de l'utilisateur selon le caractère urgent de la réception des résultats de l'application.

#### II.3.3.2 Approche multi-objectif

De manière générale, les algorithmes présentés ci-dessus ont été conçus pour l'ordonnancement de workflows sur une infrastructure distribuée et hétérogène de cloud qui se focalise sur l'optimisation du makespan ou du coût, avec ou sans contraintes respectives de budget ou de délai. Très peu d'études sont focalisées sur l'approche multi-objectif du problème c'est-à-dire minimiser à la fois le makespan et le coût d'utilisation des ressources cloud, car ces deux problèmes sont contradictoires, c'est-à-dire minimiser le makespan conduit à un coût plus élevé. Cela s'explique par le fait que pour minimiser le makespan, il est indispensable d'avoir plusieurs ressources de calcul pour éviter de trop retarder les tâches prêtes. Par contre, pour minimiser le coût d'exploitation des ressources cloud, une

seule instance de calcul est suffisante, mais conduisant au plus grand (mauvais) makespan, car les tâches s'exécuteront de façon séquentielle sur cette unique ressource de calcul.

Des heuristiques classiques ont été utilisées dans ces algorithmes d'ordonnancement, tels que HEFT [113], Min-Min [142], Max-Min [143], etc. mais elles ne s'intéressent qu'à un seul objectif. En outre, ces heuristiques sont conçus pour des ressources statiques de calcul dans des environnements de grille ou de grappe de calcul. En revanche, très peu d'études gèrent des objectifs multiples c'est-à-dire qui réduisent le temps d'exécution et le coût, sur des ressources IaaS du cloud. Il n'est pas toujours évident de concevoir un algorithme pour chaque problème d'optimisation et il n'est pas trivial de configurer les paramètres pour le problème.

Certaines approches d'ordonnancement multi-objectif [121, 135–137] ont été proposées. Cependant, elles ne tiennent pas compte de la localisation des données qui est un élément indispensable à l'ordonnancement d'applications *data-intensive* sur des ressources du cloud, car l'on ne sait pas d'avance si les machines virtuelles réservées sont sur un même *data-center*, c'est-à-dire sur un seul site ou sur plusieurs *data-centers* c'est-à-dire sur plusieurs sites distincts géographiquement et ne sont donc pas adaptées à un environnement cloud, c'est-à-dire multi-site.

De Oliveira *et al.* [135] proposent une approche multi-objectif qui vise la réduction du makespan et du nombre de ressources utilisées en termes de cœurs virtuels de calcul. L'approche proposée par ces auteurs est particulièrement adaptée aux applications *compute-intensive* c'est-à-dire nécessitant plus de calcul que de données. Les ressources de calcul sont réservées dans la limite des ressources disponibles sur un *data-centers*, car leur approche s'applique aux ressources cloud mono-site c'est-à-dire les ressources ne sont pas géographiquement réparties. Cependant, cette approche n'est pas appropriée pour l'exécution de workflows scientifiques dotés de plusieurs milliers de tâches sur les ressources IaaS du cloud, car l'on ne sait pas d'avance où celles-ci sont localisées et n'est donc pas adaptée pour la prise en compte du transfert de données volumineuses dans le cas des applications *data-intensive*.

Tout comme De Oliveira *et al.* [135], Rodriguez et Buyya [137] proposent une stratégie d'ordonnancement multi-objectif en se basant sur le principe de *clustering*. Chaque regroupement effectué est par la suite mappé sur la VM qui pourrait à la fois vite terminer l'exécution des tâches et qui serait la moins coûteuse. Ces auteurs ont exploité l'hétérogénéité des ressources cloud pour le mapping des tâches de chaque regroupement c'est-à-dire les VMs sont différentes les unes des autres en termes de leur puissance de calcul, mais ne tiennent pas compte du temps de transfert de données d'un regroupent à un autre. Ce qui a un impacte considérable sur le makespan pour les applications *data-intensives*.

Certaines approches multi-sites ont été proposées pour résoudre le problème multi-objectif en se basant sur le front de Pareto pour proposer un ensemble de solutions [121, 136, 138]. Le choix des meilleures solutions effectué par l'utilisateur est fait en tenant compte de certains critères selon ses besoins. Ces critères sont portés soit sur le temps d'exécution, soit sur le coût d'utilisation des ressources. Cependant, cette méthode exige que les utilisateurs choisissent la solution la plus appropriée.

Fard *et al.* [136] ont proposé une méthode d'ordonnancement multi-objectifs pour l'exécution de workflows scientifiques dans les infrastructures cloud distribuées. Leur approche produit un ordonnancement basé sur HEFT et pour chaque tâche, trouver la ressource qui minimise les quatre objectifs qui sont : le makespan, le coût d'utilisation des ressources, la consommation énergétique et la fiabilité des ressources utilisées. Ces auteurs proposent un front de Pareto pour la représentation des solutions optimales. Néanmoins, cette approche ne prend pas en compte la distribution des données dans un environnement multi-site.

Durillo et Prodan [121] ont proposé l'algorithme MOHEFT (*Multi-Objectifs Heterogeneous Earliest Finish Time*) qui est une extension de l'algorithme d'ordonnancement de workflow bien connu : HEFT [113]. La méthode heuristique calcule un ensemble de solutions se trouvant sur le fron de Pareto parmi lesquelles les utilisateurs peuvent choisir la meilleure solution possible. MOHEFT construit plusieurs ordonnancements (ou solutions) de workflow, au lieu d'un seul comme le fait HEFT. La qualité des solutions est assurée par l'utilisation de relations de dominance, tandis que leur diversité est garantie par l'utilisation d'une mesure connue sous le nom de "distance d'encombrement" (*crowding distance*) [114]. Ces solutions (définitives) sont obtenues via des solutions intermédiaires qui représentent l'ordonnancement de workflows partiels obtenus par l'ajout itérative d'une nouvelle tâche jusqu'à obtenir toutes les tâches du workflow. L'algorithme est générique dans le nombre et le type d'objectifs qu'il est capable de traiter ; toutefois, le makespan et le coût d'utilisation des ressources sont optimisés pour l'exécution du workflow scientifique dans un cloud IaaS. La flexibilité offerte par MOHEFT en tant qu'algorithme générique multi-objectifs est très attrayant. En outre, le front de Pareto est un outil efficace d'aide à la décision, car il permet aux utilisateurs de sélectionner la solution de compromis la plus appropriée en fonction de leurs besoins. Par exemple, leurs expériences ont démontré que dans certains cas, le coût pouvait être réduit de moitié avec une augmentation de 5 % du makespan. MOHEFT offre une complexité temporelle approximative de $O(n \times m)$ où $n$ est le nombre de tâches et $m$ le nombre de ressources.

Zhou *et al.* [138] ont proposé FDHEFT (*Fuzzy Dominance sort based Heterogeneous Harliest-Finish-Time*), capable de minimiser à la fois le coût et la durée des workflows déployés sur les nuages IaaS.

Très peu d'études ont abordé l'aspect multi-objectif du problème d'ordonnancement

de workflows sur les ressources de cloud computing. Néanmoins cet aspect reste la plus adaptée, car en proposant une heuristique spécifique pour l'ordonnancement des tâches des workflows pour minimiser le makespan, on peut également minimiser une autre métrique, soit l'impacte énergétique, soit le coût d'exploitation des ressources. Ainsi dans cette thèse, nous cherchons à minimiser à la fois le makespan et le coût monétaire d'exploitation des ressources du cloud. Ces deux métriques, comme le mentionnent certaines études de la littérature sont des objectifs contradictoires à minimiser.

Deux questions découlent de l'étude bibliographique précédente. D'une part, laquelle de ces approches est-elle la mieux adaptée à l'ordonnancement de workflows scientifiques sur les ressources IaaS du cloud ? D'autre part, comment les différentes tâches indépendantes du workflow scientifique peuvent elles être réparties sur les ressources IaaS du cloud afin de répondre à des objectifs spécifiques ?

Compte tenu des considérations de conception des méta-heuristiques pour l'ordonnancement de workflows scientifiques, on s'attend à ce que les algorithmes proposés fournissent de bonnes solutions d'ordonnancement. Cependant les algorithmes d'ordonnancement qui utilisent les méta-heuristiques exploitent un processus aléatoire. Ce processus aléatoire peut augmenter la complexité en temps de recherche de solution optimale et peut faire perdre une solution non optimale à une itération, mais pouvant conduire à une bonne solution plus tard. Pour résoudre le problème d'ordonnancement sur les ressources du cloud où les tâches sont liées par des dépendances de données, les méta-heuristiques telles que les algorithmes génétiques, l'optimisation de colonie de fourmis, etc. ne sont pas totalement adaptées. C'est pour ces raisons que nous explorons les heuristiques.

## II.4 Conclusion partielle

Il existe deux grandes approches heuristiques pour résoudre le problème d'ordonnancement de workflows scientifiques sur les ressources IaaS du cloud, qui sont l'approche de clustering et l'approche de list scheduling. En générale, les workflows scientifiques sont conçus pour s'exécuter sur des ressources distribuées. L'approche de clustering consiste à faire des regroupements de tâches et mapper chaque regroupement sur les ressources appropriées (distribuées) du cloud computing. Cependant, il existe des dépendances de données entre les différents regroupements constitués. Dans un tel scénario, il ne faut pas négliger ces dépendances car cela peut entraîner des retards considérable sur la date de début de certaines tâches. Par conséquent, nous pouvons dire que cette approche n'est pas complètement adaptée au problème d'ordonnancement que nous résolvons dans cette thèse. Quant à l'approche de list scheduling, elle consiste à maintenir dans une liste triée selon un certain ordre l'ensemble des tâches du workflow scientifique. Cette approche tient

compte en plus du temps de traitement de chaque tâche, le temps moyen du transfert de données entre deux tâches dépendantes si celles-ci doivent s'exécuter sur des ressources distinctes. A cet effet, nous pouvons affirmer que cette approche est la mieux adaptée au problème d'ordonnancement de workflows scientifiques sur les ressources IaaS du cloud.

Dans la suite de cette thèse, nous nous fondons sur l'approche du *list scheduling* pour proposer, d'une part un nouvel algorithme d'ordonnancement qui vise à minimiser le temps d'achèvement (makespan) de workflows scientifiques par la réduction de transfert de données sur le réseau ; et de l'autre, un outil d'aide à la décision proposant à un utilisateur un ensemble d'infrastructure qui permet d'avoir un bon compromis entre le makespan et le coût d'exploitation des ressources IaaS du cloud.

Chapitre III

# Heuristique d'optimisation du makespan par la réduction des transferts de fichiers

## Sommaire

## III.1 Introduction partielle

Le cloud computing englobe des services fiables à la demande fournis sur Internet avec un accès facile à des ressources de calcul, de stockage et de réseaux virtuelles pratiquement illimités. Ce pool de ressources est généralement exploité par un modèle de paiement à l'usage. Dans ce modèle de paiement, des garanties sont proposées par le fournisseur de service cloud utilisant des accords sur les niveaux de service (SLA) personnalisés. Le SLA fait partie d'un contrat de service dans lequel un service est défini formellement.

Une des différences significatives entre la grille de calcul et le cloud réside dans le modèle commercial. En effet, le cloud propose une méthode de paiement particulière en offrant des ressources dont l'utilisateur à l'illusion de l'infinité de celles-ci. Cependant, les services cloud peuvent être divisés en trois grandes catégories : logiciel en tant que service (*Software as a Service* : SaaS), plate-forme en tant que service (*Platform as a Service* : PaaS) et infrastructure en tant que service (*Infrastructure as a Service* : IaaS). Le cloud IaaS est la fourniture d'une infrastructure informatique (ressources de calcul, de réseau, de stockage, etc.) sous forme de service tel qu'Amazon Elastic Compute Cloud (EC2). Le cloud IaaS est particulièrement adapté à l'exécution de workflows scientifiques et le challenge est d'effectuer un bon "mapping" en affectant les différentes tâches et données aux ressources cloud.

Dans ce chapitre, il s'agit de proposer une heuristique spécifique dont l'objectif est de donner de meilleures performances par rapport aux algorithmes de la littérature. Ainsi, nous commencerons par présenter une description des éléments de la plateforme utilisée pour implémenter notre algorithme à la section III.2. La section III.3 illustre le modèle de plateforme utilisé et présente les différents workflows scientifiques synthétiques pour valider notre approche. L'heuristique spécifique proposée est abordée à la section III.4, suivi de l'évaluation des performances à la section III.5. Nous terminons ce chapitre par la présentation de nos résultats en les comparant avec des algorithmes de la littérature à la section IV.5.

## III.2 Description des éléments de la plateforme

L'infrastructure qui exécutera les différentes tâches est constituée d'un ensemble d'élément pour garantir un meilleur temps d'achèvement (makespan) de l'application à forte intensité de données.

Les machines possèdent de nombreux cœurs de calcul. Le modèle de plateforme utilisé

dans cette thèse est inspiré de l'offre m5d du fournisseur cloud Amazon [51]. Ainsi dans cette thèse nous avons supposé que les machines virtuelles utilisées possèdent entre deux cœurs et quatre-vingt-seize cœurs. Cette supposition favorise l'exécution simultanée de plusieurs tâches de l'application parallèle. Les instances utilisées dans la littérature sont généralement des instances mono-cœur, et même lorsque ces instances sont multi-cœurs, elles utilisent tous les cœurs pour exécuter une unique tâche au même instant donnant ainsi un caractère parallèle à cette tâche, et celle-ci peut être exécutée sur tous les cœurs. Ce qui n'est pas réaliste selon les études de Juve *et al.* [58] qui estiment que les tâches des workflows scientifiques sont séquentielles. Chaque machine possède un disque dur rapide (SSD) pour le stockage de certains fichiers de sortie. La distribution des différents fichiers (entrée et sortie) est importante pour la réduction du temps d'exécution d'un workflow scientifique. Utiliser un unique service de stockage pour tous les fichiers comme c'est le cas dans les études de la littérature, engendre des contentions sur le réseau car cet unique service de stockage est en permanence sollicité par toutes les machines virtuelles auxquelles elles sont liées par un lien qui sera partagé entre celles-ci. C'est pour ces raisons que nous avons supposé que chaque machine virtuelle est dotée d'un service de stockage (en plus des cœurs) qui permettra à chaque tâche qui s'exécutera sur ce nœud de stocker tous les fichiers de sortie générés et indispensables à l'exécution d'une autre tâche (tâche successeur). Une tâche récupère tous ses fichiers d'entrée là où son (ses) prédécesseur(s) l'a (ont) stocké, si celle-ci s'exécute sur une autre machine virtuelle différente de son (ses) prédécesseur(s), sinon elle récupère les données localement, c'est-à-dire sur la même machine virtuelle. Cette hypothèse permettra de limiter les demandes de connexion à l'unique service de stockage, ce qui permet d'éviter les contentions sur le réseau et avoir un meilleur temps d'achèvement de l'application parallèle.

Au démarrage de l'exécution du workflow scientifique, les tâches de début et même certaines tâches du workflow peuvent nécessiter des fichiers en entrée qui ne sont produits par aucune tâche. Ces fichiers sont stockés sur un service de stockage spécial : le service de stockage par défaut ( *Amazon Elastic Block Storage* : EBS). Ce service de stockage permet également de stocker tous les fichiers de sortie qui ne seront pas utilisés par une autre tâche, ainsi les résultats finaux sont récupérés sur le service de stockage EBS. Le service de stockage EBS est accessible par toutes les machines virtuelles EC2 à travers une liaison spécialisée dont la bande passante est proportionnelle au nombre de cœurs de la machine virtuelle EC2.

Pour exécuter un workflow scientifique, nous avons recours à plusieurs machines virtuelles qui doivent être interconnectées. Ces machines virtuelles sont instanciées sur des serveurs physiques liés entre eux par des liens réseaux dont les débits maximum sont fixés à 25 Gbps comme il est mentionné dans le Tableau III.1. Puisque chaque serveur physique peut être partagé entre plusieurs machines virtuelles, nous avons supposé que le débit du

lien qui relie une machine au reste du réseau est proportionnel au nombre de cœur logique de cette machine virtuelle.

## III.3 Modèle de plateformes et de workflows scientifiques synthétiques

### III.3.1 Modèle de plateformes

Dans cette thèse, nous considérons des machines virtuelles homogènes, c'est-à-dire qu'elles ont toutes la même vitesse de calcul et elles sont dotées de plusieurs cœurs de calcul. Le tableau III.1 illustre les différentes machines virtuelles utilisées dans cette thèse. Contrairement aux études de la littérature, nous montrons l'intérêt d'utiliser des machines multi-cœurs pour l'ordonnancement des tâches du workflow et de l'impact de ces machines sur l'ordonnancement et le makespan résultant.

Dans ce chapitre, les machines virtuelles considérées sont homogènes (c'est-à-dire les processeurs virtuelles ont la même vitesse de calcul). Pour les expérimentations, les machines virtuelles réservées chez le fournisseur cloud seront reliés entre elles à travers un commutateur (switch). Dans cette topologie en étoile, nous ferons l'hypothèse que le commutateur ne subit pas de contentions. En effet, nous supposons que le commutateur est non bloquant et que la bande passante de chacun de ces ports est suffisamment grande par rapport à toute la bande passante autorisée par le fournisseur (jusqu'à 25 Gbps). Ainsi le commutateur peut gérer simultanément tous les flux entre les liens qui le connectent aux machines virtuelles louées sans contention. En revanche, les liens reliant chaque machine virtuelle au commutateur ont une bande passante limitée qui est proportionnelle au nombre de cœurs de la machine virtuelle et peuvent subir des contentions. Une tâche qui s'exécute sur une machine virtuelle peut nécessiter des données stockées sur une autre machine virtuelle et des données stockées sur le service de stockage par défaut. Dans le cas où une tâche nécessite des données stockées sur une machine autre que la machine où elle est prévue s'exécuter, il faudra passer par le commutateur via le lien réseau. Chaque machine virtuelle est directement reliée au service de stockage par défaut. La bande passante de ce lien est également proportionnelle au nombre de cœurs de la machine virtuelle et peut aller jusqu'à quatorze mille (14000) Mbps. Le Tableau III.1 présente un récapitulatif des toutes ces bandes passantes. A cet effet, une tâche qui nécessite des données stockées sur le service de stockage par défaut passe par ce lien.

L'architecture illustrée par la Figure III.1 représente le déploiement de la plateforme utilisée lors de la phase expérimentale. Cette architecture est le modèle de déploiement de l'infrastructure du cloud IaaS proposé par Amazon EC2.

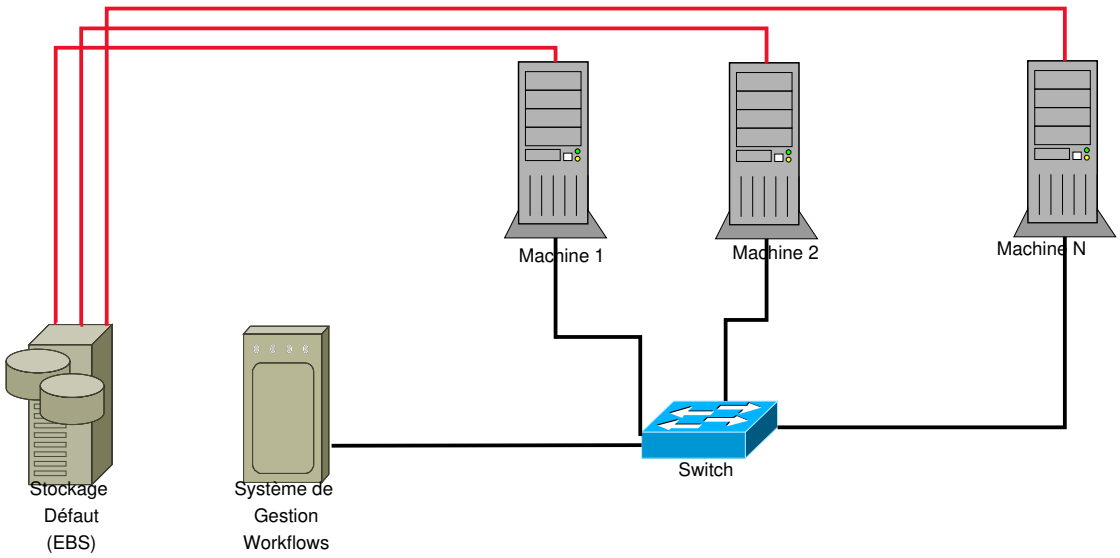



FIGURE III.1 – Architecture de déploiement.

Tableau III.1 – Caractéristiques des types d'instance m5d d'Amazon.

| Modèle | cœur | Mémoire (Go) | Instance de stockage (Go) | Bande passante réseaux (Gbps) | Bande passante EBS (Mbps) | Coût ($ par Heure) |
|---|---|---|---|---|---|---|
| m5d.large | 2 | 8 | 1 x 75 NVMe SSD | Jusqu'à 10 | Jusqu'à 3,500 | 0.113 |
| m5d.xlarge | 4 | 16 | 1 x 150 NVMe SSD | Jusqu'à 10 | Jusqu'à 3,500 | 0.226 |
| m5d.2xlarge | 8 | 32 | 1 x 300 NVMe SSD | Jusqu'à 10 | Jusqu'à 3,500 | 0.452 |
| m5d.4xlarge | 16 | 64 | 2 x 300 NVMe SSD | Jusqu'à 10 | 3,500 | 0.904 |
| m5d.8xlarge | 32 | 128 | 2 x 600 NVMe SSD | 10 | 5,000 | 1.808 |
| m5d.12xlarge | 48 | 192 | 2 x 900 NVMe SSD | 10 | 7,000 | 2.712 |
| m5d.16xlarge | 64 | 256 | 4 x 600 NVMe SSD | 20 | 10,000 | 3.616 |
| m5d.24xlarge | 96 | 384 | 4 x 900 NVMe SSD | 25 | 14,000 | 5.424 |

#### III.3.1.1 Intérêt d'utiliser des machines multi-cœurs du cloud

Traditionnellement, les machines multiprocesseurs sont une extension de l'ordinateur personnel mono-processeur, mais qui est constituées cette fois-ci de plusieurs processeurs (CPU). En tant que tels, ils conviennent à l'exécution d'applications parallèles. Fondamentalement, le parallélisme disponible dans les machines multi-processeurs est utilisé pour exécuter plusieurs processus non liés en parallèle (par exemple, exécuter deux compilations et une recherche Web sur différents processeurs). Ce même principe est utilisé dans le cloud computing afin d'offrir à l'utilisateur plusieurs gammes de machines virtuelles qui possèdent des processeurs virtuels (cœur) pour effectuer différentes instructions. Dans

cette thèse, la notion de “cœur” est utilisée au lieu de processeur virtuel pour indiquer que plusieurs processeurs virtuels sont assemblés dans une machine pour effectuer plusieurs instructions simultanément.

Généralement, une tâche d'une application parallèle qui termine son exécution peut libérer plusieurs autres qui deviennent à leur tour prêtes et susceptibles de s'exécuter. Les tâches prêtes qui proviennent de la même tâche précédente sont liées par une contrainte de dépendance dans un workflow scientifique (comme l'illustre la Figure III.2(a) à travers les tâches 2, 3 et 4) et peuvent être exécutées (toutes) de façon simultanée dans la limite des cœurs disponibles. Les workflows scientifiques issues de plusieurs domaines de recherche scientifique possèdent des milliers de tâches et certaines tâches indépendantes peuvent s'exécuter simultanément, d'où l'appellation application parallèle. Les ressources du cloud computing telles que mentionnées au Tableau III.1 permettent d'exécuter les tâches d'une telle application.

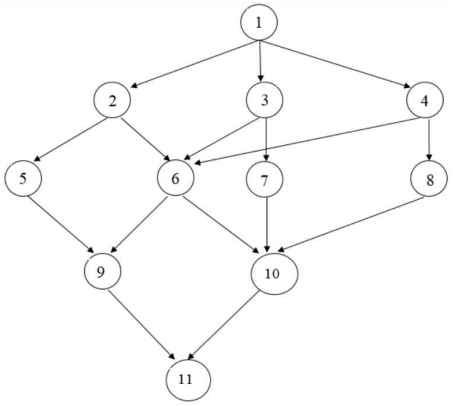
#### III.3.1.2 Impact des machines multi-cœurs sur l'ordonnancement et le makespan

Utiliser une machine mono-cœur pour exécuter toutes les tâches du workflow entraîne une exécution séquentielle, ce qui conduit au plus mauvais temps d'achèvement comme l'illustre la Figure III.2(b). Par contre, l'utilisation des machines multi-cœurs favorise l'exécution simultanée de plusieurs tâches telles que sus-mentionnées. Laisser plusieurs tâches s'exécuter simultanément permet d'avoir un gain sur le makespan comme l'illustre la Figure III.2(c).

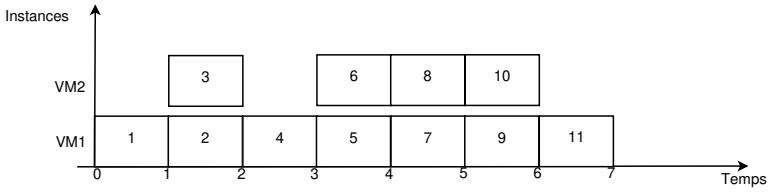
Il est trivial de dire que l'exécution d'une application parallèle sur des ressources multi-cœurs permet d'avoir un gain sur le temps d'achèvement de l'application. Mais le problème qui se pose, ici, est de savoir la location des données nécessaires en entrée pour une tâche $v_i$, afin de faire un bon mapping et d'obtenir le meilleur temps d'achèvement de l'application dû en partie aux temps de transfert des données sur le réseau.

### III.3.2 Workflows scientifiques synthétiques

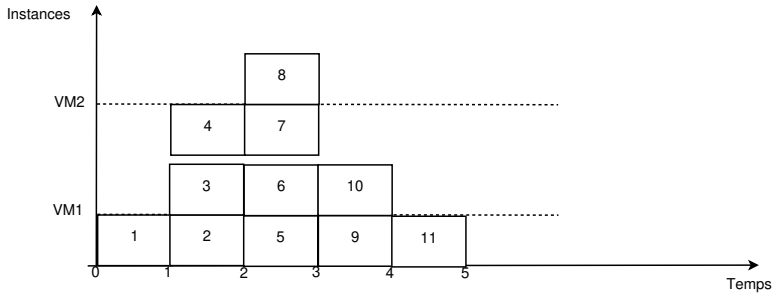
Un workflow est l'automatisation d'un processus au cours duquel les données sont traitées par différentes tâches selon un ensemble de règles. Un workflow est l'assemblage d'ensembles complexes de tâches avec des dépendances de données entre elles [144]. La représentation la plus générale est un graphe orienté, dans lequel les nœuds correspondent aux tâches à exécuter et les arêtes représentent les dépendances de données. Mais le plus souvent, un workflow est représenté comme un DAG pour de nombreuses applications. Les workflows synthétiques tirent parti des langages de programmation pour prouver une




(a) Exemple simple de workflow




(b) Diagramme de Gantt sur machines mono-cœurs




(c) Diagramme de Gantt sur machines multi-cœurs

FIGURE III.2 – Intérêt et impact d'utilisation des machines multi-cœurs.

hypothèse scientifique [145]. Un workflow scientifique à forte intensité de données (*data-intensive*) traite, gère ou produit d'énormes quantités de données pendant son exécution. En outre, les workflows scientifiques doivent être entièrement une représentation des applications réelles [145]. Les workflows scientifiques ont été utilisés dans divers domaines scientifiques.

### III.3.2.1 CyberShake

La Figure III.3(a) montre un exemple du workflow scientifique CyberShake [146] qui caractérise l'aléa sismique (sismogramme). Bien que sa structure soit relativement simple, cette application peut être utilisée pour effectuer des calculs sur des ensembles de données extrêmement importants. Le workflow CyberShake contient plus de huit cent mille (800 000) tâches et nécessitant plus de quatre-vingt mille (80 000) fichiers (200 To) [58]. La durée d'exécution de nombreuses tâches étant de l'ordre de quelques millisecondes, la charge des opérations sur les méta-données deviennent très lourdes et le fait d'accéder à ces méta-données centralisées représente un sérieux goulot d'étranglement au niveau des performances.

#### III.3.2.2 Epigenomics

La Figure III.3(b) illustre un workflow scientifique Epigenomics qui représente l'état des connaissances en matière d'épigénétique. Le workflow scientifique Epigenomics [147] est une application très complexe qui utilise plusieurs tâches fonctionnant en parallèle sur des blocs de données indépendants. Ces données sont des séquences d'ADN (acide désoxyribonucléique) obtenues par plusieurs processus d'analyse génétique.

#### III.3.2.3 Ligo

Le workflow scientifique Ligo [147] analyse les données de la coalescence (fusion d'éléments) de systèmes compacts tels que les étoiles à neutrons et les trous noirs. Ce workflow est très complexe et se compose de plusieurs sous-workflows. Une représentation simplifiée de ce workflow est illustrée sur la Figure III.3(c) avec des sous-workflow. Le workflow scientifique réel possède de 25 sous-workflows avec plus de 2000 tâches, jusqu'à 100 sous-workflows avec 200 000 tâches. Étant donné que ces tâches fonctionnent sur des données obtenues à partir de plusieurs tâches précédentes, elles peuvent être considérés comme des tâches d'agrégation de données. Une grande partie du temps d'exécution (jusqu'à 92%) correspond au temps d'écriture et de lecture sur les périphériques de stockages. Un aspect intéressant de l'utilisation du workflow scientifique Ligo est qu'il lit une grande quantité de données (213 Go), mais écrit très peu (50 Mo).

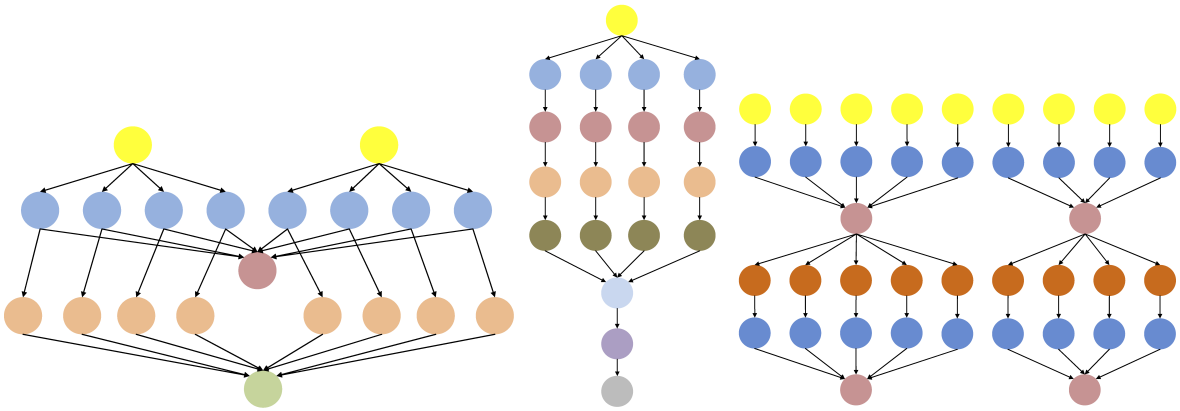
#### III.3.2.4 Montage

La composition de mosaïques d'images est essentielle pour les astronomes. La plupart des instruments astronomiques ont un champ de vision et une plage spectrale limités. Cela signifie que les grandes structures astronomiques ne peuvent pas être observées en une seule prise de vue. La composition d'une image finale d'une grande structure astronomique nécessitera souvent la fusion d'images prises à différents moments, en différents endroits, avec différents instruments. Les structures astronomiques ne changent généralement pas du jour au lendemain et le déplacement de la terre est généralement négligeable, l'agencement d'images provenant de ces différents lieux et moments nécessite des outils spéciaux. Dans le domaine de l'astronomie, le workflow scientifique le plus connu est Montage [148, 149], développé par *California Institute of Technology* (CalTech) grâce à un financement de la NASA et de la *National Science Foundation*. La Figure III.3(d) illustre le workflow scientifique Montage. La taille du workflow scientifique Montage dépend du nombre d'images utilisées pour construire la mosaïque de la structure astronomique souhaitée. Pour générer de telles mosaïques, Montage effectue des calculs scientifiques en se basant sur des données. Le workflow scientifique Montage utilise moins de ressources de

calcul, mais nécessite plus de 1 Go de données en entrée et produit plus de 775 Go de données pendant l'exécution rendant cette application *data-intensive* [58].

#### III.3.2.5 Sipht

L'université de Harvard mène une vaste recherche sur les petits ARN non traduits (*small RNAs* : sARNs) qui régulent plusieurs processus tels que la sécrétion ou la virulence des bactéries. Le workflow scientifique SIPHT [147] est un protocole d'identification des sRNAs pour automatiser la recherche des gènes et de toutes les répliques bactériennes dans la base de données du *National Center for Biotechnology Information* (NCBI). La structure du workflow SIPHT est illustrée par la Figure III.3(e)



(a) CyberShake (b) Epigenomics (c) Ligo

(d) Montage (e) Sipht

FIGURE III.3 – Quelques exemples de workflows scientifiques.

Gérer les méta-données de manière centralisée pour de tels scénarios n'est pas approprié. Car la congestion générée par les opérations simultanées de lecture et d'écriture sur les méta-données et les opérations inter-sites à distance provoquent d'importants retards dans l'exécution. Pour remédier à ce problème, certains systèmes de fichiers reposent sur

l'utilisation de serveurs de méta-données décentralisés [150]. Cette thèse explore ce principe à travers l'utilisation de plusieurs services de stockage, afin d'éviter à certaines tâches d'utiliser la bande passante du réseau pour télécharger certaines données indispensables à son exécution.

## III.4 Heuristique proposée et motivation

### III.4.1 Motivation

Les workflows scientifiques sont généralement utilisés pour modéliser des applications parallèles complexes sous forme de graphe orienté acyclique, dans lequel les sommets représentent les tâches de traitement et les arcs des dépendances de données entre celles-ci. Une tâche du workflow scientifique peut nécessiter plusieurs données en entrée, une donnée peut correspondre à plusieurs tâches pendant l'exécution du workflow. Ainsi, l'exécution efficace de workflow scientifique à forte intensité de données (*data-intensives*), par exemple CyberShake, Epigenomics, Montage [58] devient une question importante.

Certains systèmes de gestion de workflow, par exemple Pegasus [63], prennent généralement en charge la localisation des données, c'est-à-dire les méta-données indispensables à l'exécution du workflow scientifique. Les données généralement distribuées géographiquement sont utilisées pour l'analyse et l'exécution du workflow scientifique et peuvent être très importantes. Pendant l'exécution d'une tâche, il peut avoir de multiples échanges de données entre les différents services de stockage. Toutefois, les données et les ressources (de calcul et de stockage) nécessaires pour gérer chaque tâche du workflow scientifique peuvent très bien être réparties sur différents sites (éventuellement dans différentes régions), car l'on ne sait pas d'avance où les machines virtuelles louées sont localisées.

Pour permettre l'exécution de chaque tâche des workflows sur des ressources multi-cœurs du cloud computing avec des données d'entrée distribuées, celles-ci doivent être mappées sur la bonne ressource permettant de minimiser le makespan. Ensuite, le problème d'ordonnancement consiste à décider sur quelles ressources (sites) exécuter les tâches afin d'atteindre un objectif donné, par exemple réduire le temps d'exécution de l'application parallèle. En outre, comme le transfert de données entre deux sites différents peut prendre beaucoup de temps, la résolution du problème d'ordonnancement doit tenir compte des types de ressources, et par conséquent des différentes largeurs de bande passante.

Dans cette thèse, nous nous focalisons sur le problème d'ordonnancement de tâches pour réduire le makespan, c'est-à-dire le temps d'exécution du workflow , par la réduction des transferts de données à travers le réseau. Nous utilisons une architecture distribuée avec une VM maîtresse qui coordonne l'exécution de chaque tâche et qui sait où est sto-

cker toutes les données nécessaires à l'exécution de l'application parallèle. Dans cette architecture, les données transférées entre les VMs peuvent être des données intermédiaires produites à la sortie de chaque tâche, c'est-à-dire pendant l'exécution de l'application scientifique ou des données fournies en entrée du workflow. Les données intermédiaires sont les données générées après l'exécution des tâches et peuvent également être les données d'entrée pour les tâches suivantes. Dans le cloud computing d'Amazon EC2, la largeur de la bande passante entre deux VMs est fonction du type de la VM, c'est-à-dire du nombre de cœurs que possède cette VM. Pour les workflows à forte intensité de données, il peut y avoir de nombreuses données. Par exemple, des données intermédiaires et des données d'entrées de l'application à transférer entre différentes VMs pour l'exécution d'une tâche alors que le temps d'exécution de la tâche peut être très faible. Par conséquent, le temps de transfert des données intermédiaires et des données d'entrée ne peuvent être ignorés dans le processus d'ordonnancement. Ainsi, nous considérons également le temps de transfert des données intermédiaires et des données d'entrée dans le processus d'ordonnancement afin de mieux réduire le temps d'exécution total du workflow.

Nous apportons les contributions suivantes :
premièrement, nous proposons une infrastructure dotée de VMs multi-cœurs rendant possible l'exécution simultanée de plusieurs tâches en parallèles.

Deuxièmement, nous proposons d'utiliser le profil d'utilisation de VMs afin de minimiser le temps d'inactivité des machines virtuelles sollicitées.

Troisièmement, nous proposons un nouvel algorithme d'ordonnancement de tâches, c'est-à-dire l'ordonnancement de tâches à forte intensité de données sur des ressources multi-cœurs du cloud IaaS pour l'exécution de workflows scientifiques avec prise en charge de la localisation des données géographiquement distribuées.

Quatrièmement, nous effectuons une évaluation expérimentale approfondie basée sur la simulation qui nous permet de mieux implémenter l'architecture du cloud IaaS et le déploiement d'une nouvelle approche d'ordonnancement.

### III.4.2 Rappel de l'algorithme HEFT

L'ordonnancement de tâches nécessite de connaître au préalable l'ordre dans lequel ces dernières doivent être traitées afin de répondre efficacement aux exigences de l'algorithme d'ordonnancement. Ainsi, les algorithmes de *List Scheduling* ont en commun l'établissement d'une liste de tâches triées selon un certain critère (ordre ou priorité).

En effet, Topcuoglu *et al.* [113] ont mis au point un algorithme dénommé HEFT, initialement développé pour les grilles de calculs sur des processeurs hétérogènes non uniformes entièrement connectés, dont l'objectif est de réduire le temps d'achèvement (makespan)

d'une application parallèle modélisée sous forme de DAG. Le principe de leur étude peut se résumer en trois étapes :

1. l'attribution de priorité à chaque tâche ;
2. le tri par ordre croissant de priorité ;
3. l'allocation de ressource.

La première étape consiste à calculer le niveau de priorité de chaque tâche et se fait en tenant compte du temps d'exécution que met chaque tâche sur une ressource et du temps de transfert des données qu'une tâche met pour acheminer l'ensemble des données qu'elle produit en sortie. La deuxième étape consiste à trier les tâches. Quant à la troisième étape, elle permet de trouver le processeur qui termine au plus tôt l'exécution de la tâche.

Nous proposons une approche de *List Scheduling* qui stipule d'utiliser une liste de tâches triées par priorité décroissante. La notion de priorité introduite par Topcuoglu *et al.* [113] à travers l'Équation III.1 prend en compte le temps d'exécution moyen d'une tâche ($t_i$) et le temps moyen qu'une tâche met pour transférer l'ensemble de ses fichiers de sortie ($\mathcal{C}_{i,j}$). Le *bottom-level* d'une tâche $v_i$, noté $bl_i$ représente la longueur du plus long chemin d'une tâche $v_i$ à la dernière tâche $v_{exit}$ (tâche de sortie), incluant son propre temps d'exécution $t_i$. Puisque la dernière tâche n'a pas de successeur, son *bottom-level* est égal à sa durée d'exécution $t_{exit}$ et représenté par l'Équation III.2.

Le *bottom-level* permet de connaître l'ordre d'exécution des tâches du workflow scientifique. Plus le *bottom-level* d'une tâche est grand, plus cette tâche est prioritaire et le traitement de celle-ci ne doit pas être retarder, sinon cela aura un impact négatif sur l'exécution des tâches qui viennent après elle. Pour ce faire, il est important de connaître les ressources de calcul sur chaque machine virtuelle louée. Nous utilisons dans la suite de cette étude la notion *usage profile* proposée par N'takpé et Suter [151] pour la mise à jour régulière du nombre de cœurs disponible dans chaque machine virtuelle pour décider à quel moment le mapping doit se faire sur une VM.

$$bl_i = t_i + \max_{v_j \in succ(v_i)} (\mathcal{C}_{i,j} + bl_j) \tag{III.1}$$

$$bl_{exit} = t_{exit} \tag{III.2}$$

### III.4.3 Description du problème

Le modèle de plateforme est basé sur une configuration de cloud IaaS. Plusieurs instances de machines virtuelles (VMs) sont déployées sur des serveurs physiques au sein de data-centers géographiquement répartis. Plus précisément, nous considérons un ensemble

de machines virtuelles similaires aux instances M5 d'Amazon EC2 [51]. Ces machines virtuelles sont des instances M5d fournies avec un espace de stockage local sur les disques SSD NVMe tandis que les instances M5 normales doivent s'appuyer sur le service Amazon Elastic Block Storage (EBS) pour stocker les données. Le Tableau III.1 détaille les caractéristiques des instances M5d disponibles. Les coûts indiqués en dollars par heure correspondent aux instances Linux à la demande dans la région USA-Est (Ohio) au moment de la rédaction de cette thèse.

Le nombre de processeurs virtuels (cœurs) dans cette série d'instances varie de deux (2) à quatre-vingt seize (96), avec une quantité constante de mémoire par cœur de quatre (4) Go. Ces instances sont généralement déployées par Amazon sur des nœuds dotés d'un processeur *Intel Xeon Platinum 8000*. La particularité des instances M5d est leur facilité de connecter un stockage rapide SSD qui est lié à la durée de vie de l'instance. Dans cette étude, nous visons à exploiter ce stockage rapide qui est partagé par les cœurs d'une instance mais qui leur est dédié pour stocker les fichiers intermédiaires produits lors de l'exécution de l'application parallèle, réduisant ainsi le nombre de transferts de données sur le réseau pour les tâches planifiées sur la même machine virtuelle. Seuls les fichiers d'entrée et de sortie du workflow seront stockés sur un nœud de stockage externe appelé stockage par défaut (EBS).

La bande passante qui relie un commutateur virtuel (switch) à une instance ou à l'EBS dépend de la taille de l'instance, c'est-à-dire qui est proportionnelle au nombre de cœur. Nous supposons que seules les plus grandes instances capables d'exploiter un nœud complet avec quarante-huit (48), soixante-quatre (64) ou quatre-vingt seize (96) cœurs, ont une bande passante réseau garantie de dix (10), vingt (20) et vingt-cinq (25) Gbps respectivement. Pour les instances plus petites, de deux (2) à trente-deux (32) cœurs, la bande passante disponible est proportionnelle au nombre de cœurs et égale à deux cent huit virgule trente-trois (208,33) Mbps par cœur. Toutes les instances de machines virtuelles démarrées pour l'exécution d'un workflow donné sont connectées via un seul commutateur.

Selon la description des instances M5d, la connexion d'une VM à l'EBS passe par une liaison réseau dédiée qui est prise en compte dans notre infrastructure simulée. En ce qui concerne les connexions réseaux entre les machines virtuelles, la bande passante de la connexion dédiée entre une machine virtuelle et l'EBS est proportionnelle au nombre de cœur. Pour les machines virtuelles de moins de trente-deux (32) cœurs, c'est deux cent dix-huit virgule soixante-quinze (218,75) Mbps par cœur.

Les workflows scientifiques sont représentés par des graphes acycliques orientés (DAG) $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, où $\mathcal{V} = \{v_i | i = 1, \ldots, V\}$ est un ensemble de sommets représentant les tâches à exécuter du workflow et $\mathcal{E} = \{e_{i,j} | (i,j) \in \{1, \ldots, V\} \times \{1, \ldots, V\}\}$ est un ensemble

d'arêtes entre les sommets, représentant soit une dépendance de données, soit un transfert de fichier ou une dépendance de flux entre deux tâches. Chacune des tâches composantes du workflow a une durée d'exécution prédéfinie (estimée), nécessite un ensemble de fichier (*input files*) pour démarrer son exécution et produit un ensemble de fichier (*output files*) à la fin de son exécution. Nous notons donc $Input_i^k$ (resp. $Output_i^k$), le $k^{th}$ *input* (resp. *output*) *file* d'une tâche donnée $v_i$. Lorsqu'un *output file* produit par une tâche $v_i$ est utilisé en entrée par une autre $v_j$, cela crée une dépendance des données entre $v_i$ et $v_j$, représentée par l'arête $e_{i,j}$. Les *input files* qui ne sont produits par aucune des tâches du workflow sont appelés fichiers d'entrée (*entry files*) de l'application parallèle. Inversement, les fichiers de sortie qui ne sont utilisés par aucune tâche sont appelés fichiers de sortie (*exit files*).

Nous définissons par la suite deux paramètres associés à chaque tâche du workflow qui seront utilisés lors du processus d'ordonnancement. Nous définissons d'une part le *Local Input Volume* d'une tâche $v_i$ sur une machine $M_j$, ou $LIV_{i,j}$ (voir Équation III.3) comme étant la somme des tailles des fichiers que $v_i$ prend en entrée et qui sont localement stockés sur $M_j$. Et d'autre part, le *Local Output Volume* ou $LOV_{i,j}$ (voir Équation III.4) comme la somme des tailles des fichiers produits par $v_i$ qui sont utilisés par les successeurs de $v_i$ et également stockés sur $M_j$. Si un fichier est utilisé par plusieurs successeurs, celui-ci sera téléchargé autant de fois que nécessaire par les successeurs de $v_i$.

$$LIV_{i,j} = \sum_{k=1}^{m} Input_{i,j}^k \tag{III.3}$$

$$LOV_{i,j} = \sum_{k=1}^{m} Output_{i,j}^k \tag{III.4}$$

Lors de l'exécution du workflow, tous les fichiers intermédiaires, c'est-à-dire ceux produits par une tâche et utilisés par une autre, seront stockés localement sur le stockage SSD d'une ou plusieurs machines. Seuls les fichiers d'entrée et de sortie du workflow seront stockés sur le service EBS accessible à toutes les machines virtuelles. Le temps de transfert d'un fichier d'une machine à une autre comprend le temps de lecture du fichier sur le disque de la machine source, la durée du transfert de données sur le réseau et le temps d'écriture du fichier sur le disque à destination.

### III.4.4 Profil d'utilisation d'une machine

Ordonnancer une tâche $v_i$ sur une machine $M_k$ tel que $M_k \in \mathcal{M}$ où $\mathcal{M}$ est l'ensemble des machines virtuelles louées pour exécuter toutes les tâches du workflow, cela revient à trouver la place de cette tâche dans le profil d'utilisation des machines, c'est-à-dire trouver la place de cette tâche dans une liste des ressources de calcul disponible à un moment

donné. Pour mettre à jour le *usage profile* de chaque machine, il convient de calculer au préalable la date de début au plutôt prévue et la date de fin de chaque tâche. Ainsi la date de début au plutôt ou encore le *start time* ($st_k(v_i)$) d'une tâche $v_i$ sur une machine $M_k$ est la date à laquelle son dernier prédécesseur a terminé son exécution et nous le formalisons à travers l'équation suivante :

$$st_k(v_i) = \max_{v_j \in prec(v_i)} (ft(v_j)) \tag{III.5}$$

où $v_j$ est un prédécesseur de $v_i$, c'est-à-dire une tâche parente à $v_i$. Pour les tâches d'entrée du workflow, c'est-à-dire les tâches qui n'ont pas de prédécesseur, le *start time* est le suivant pour toute machine $M_k$ :

$$st_k(v_{entry}) = 0 \tag{III.6}$$

Toutes les machines ont la même vitesse de calcul, c'est-à-dire qu'elles sont homogènes et la date de fin prévu d'une tâche $v_i$ est ainsi définie par l'équation suivante :

$$ft(v_i) = st_k(v_i) + t_i + \mathcal{C}_{i,j} \tag{III.7}$$

où $t_i$ est le temps d'exécution de la tâche $v_i$, et $\mathcal{C}_{i,j}$ est le temps de transfert de données, c'est-à-dire le temps qu'une tâche $v_i$ met pour récupérer l'ensemble des données produites par ses prédécesseurs si ceux-ci se sont exécutés sur une machine autre que celle qui a été prévue pour exécuter $v_i$. Notre modèle de plateforme prend en compte des machines multi-cœurs du cloud computing et nous proposons l'algorithme 1 qui permet de savoir à tout moment le nombre de cœurs disponibles sur une machine virtuelle.

Ordonnancer une tâche $v_i$ revient à trouver sa place dans le profil d'utilisation des ressources, c'est-à-dire dans l'ensemble de ressources disponibles à un moment donné. Le profil d'utilisation est régulièrement mise à jour par l'ordonnanceur afin de savoir à tout moment le nombre de ressources disponibles. La sélection d'un créneau spécifique pour une tâche détermine la date de début d'exécution de celle-ci sur une machine $M_k$

L'initialisation du profil d'utilisation ($usage\_profile_k$) d'une machine $M_k$ est représentée par le couple $(0; C_k)$, où $C_k$ est le nombre de cœurs disponibles sur la VM $M_k$ à l'instant $t = 0$. Au fur et à mesure qu'une tâche est ordonnancer sur une VM $M_k$, le nombre de cœur $C_k$ disponible est mise à jour en même temps que la date de disponibilité des ressources.

### III.4.5 Algorithme de réduction des fichiers provenant des tâches précédentes

Pour proposer un bon algorithme d'ordonnancement, nous supposons que les $\mathcal{M}$ instances de machines virtuelles sont connues. L'objectif de l'algorithme d'ordonnancement

**Algorithme 1** Profil d'utilisation des machines.

**pour tout** $M_k \in \mathcal{M}$ **faire**
    $usage\_profile_k \leftarrow \{(0; C_k)\}$ ▷ Initialisation du $usage\_profile$
**fin pour**
**pour tout** $v_i \in \mathcal{V}$ **faire**
    **pour tout** $M_k \in \mathcal{M}$ **faire**
        Calculer $st_k(v_i)$ et $ft(v_i)$ ▷ voir Eq. III.5 et III.7
        **si** $t_k^{courant} < ft(v_i)$ ou ($ft(v_i) > t_k^{courant}$ et $ft(v_i) < t_k^{precedent}$) **alors**
            Mapper $v_i$ sur $M_k$
            Mise à jour $usage\_profile_k$
        **fin si**
        **si** $t_k^{courant} \geq st_k(v_i)$ et $t_k^{courant} < ft(v_i)$ **alors**
            Mise à jour $usage\_profile_k$
        **sinon si** $t_k^{courant} < st_k(v_i)$ et $st_k(v_i) < t_k^{precedent}$ **alors**
            Mapper $v_i$ sur $M_k$
            Mise à jour $usage\_profile_k$
        **fin si**
    **fin pour**
**fin pour**

proposé est de mapper l'ensemble $\mathcal{V}$ de $V$ tâches composant le workflow sur $\mathcal{M}$ tout en exploitant deux caractéristiques principales de la plateforme IaaS cloud, à savoir (1) des instances multi-cœurs et (2) un espace de stockage local rapide et distribué, afin de minimiser l'impact des transferts de données sur l'exécution de workflows scientifiques à forte intensité de données.

L'algorithme 2 commence par établir une liste triée qui contient toutes les tâches du workflow (linges 1-2). Cette liste tient compte de toutes les dépendances qui existent entre les tâches. Ensuite, l'algorithme détermine un premier *mapping* pour chaque tâche $v_i$ dans $\mathcal{V}$ (lignes 3-7) qui minimise les transferts de données provenant des tâches précédentes. La machine $M_j$ sélectionnée dans $\mathcal{M}$ est celle qui : (i) minimise la date de début de la tâche $v_i$ ($st_j(v_i)$) ; et (ii) maximise le volume des fichiers d'entrée stockés localement, c'est-à-dire les données qui sont stockées sur la machine $M_j$ pour la tâche $v_i$. Le raisonnement est qu'entre deux machines virtuelles capables d'exécuter $v_i$ au même moment, c'est-à-dire $v_i$ a le même *start time* sur les deux machines virtuelles, celle qui minimise la quantité de transferts de données sur le réseau est privilégiée.

Toutes les instances de machines virtuelles considérées étant dotées de plusieurs cœurs, l'ordonnancement d'une tâche $v_i$ sur une machine $M$ implique alors d'effectuer un planning local à l'intérieur de la machine virtuelle. Afin de maximiser l'utilisation des cœurs au sein d'une machine virtuelle, chaque machine est gérée comme le ferait un gestionnaire de tâches et de ressources. Les informations disponibles sur le temps d'exécution (estimé) de chaque tâche permet de mettre en place un mécanisme de remplissage conser-

vateur (*conservative backfilling*) [152] lors de l'élaboration du planning local. La mise à jour d'un tel profil d'utilisation d'une machine virtuelle est obligatoire pour déterminer le moment auquel une nouvelle tâche peut commencer sur cette machine particulière, c'est-à-dire $st_j(v_i)$. Ensuite, après avoir sélectionné une machine $M$ pour l'exécution de $v_i$ , la mise à jour du profil d'utilisation de $M$ (ligne 6) est faite. Les profils d'utilisation des machines virtuelles sont également utilisés dans la deuxième étape de l'algorithme 2 au cours de laquelle les tâches de l'ordonnancement initial sont réorganisées afin de réduire davantage la quantité de données transférées sur le réseau.

Cette étape de réorganisation ou réarrangement (lignes 8-11) parcourt le workflow scientifique niveau par niveau, de bas en haut. Le principe est que lors de l'ordonnancement initial qui se déroule de haut en bas, seul le volume de données provenant des prédécesseurs directs d'une tâche est pris en compte. Il est en effet impossible de tenir compte de la localité des données nécessaires à un descendant direct d'une tâche lorsque le mapping de celle-ci sur une VM n'est pas encore déterminé. Cela peut conduire à des mouvements de données évitables.

Le niveau 0 (*level* 0) correspond au niveau le plus élevé du DAG et comprend toutes les tâches d'entrée du workflow scientifique (*entry task*). Pour chacune des autres tâches, le niveau est calculé récursivement en fonction du niveau maximum de ses prédécesseurs, plus un. Enfin, $L$ indique le nombre de niveaux dans le workflow. Le principe de l'étape de réarrangement est décrit dans l'algorithme 3.

**Algorithme 2** Mapping des tâches de workflow.

1: Calculer $bl_i$ de chaque tâche $v_i$
2: Trier $\mathcal{V}$ par $bl_i$ décroissant
3: **pour tout** $v_i \in \mathcal{V}$ **faire**
4: $M \leftarrow \{M_j \in \mathcal{M} \mid st_j(v_i)$ est minime **et** $LIV_{i,j}$ est maximal $\}$
5: Mapper $v_i$ sur $M$
6: Mise à jour du *usage profile* de $M$
7: **fin pour**
8: **pour** $l = L$ à 0 **faire**
9: $V_l \leftarrow$ tâches au niveau $l$ triées par $bl$ décroissant
10: Rearrangement($V_l$) ▷ voir Algorithme 3
11: **fin pour**

### III.4.6 Algorithme de réduction des fichiers allant vers les tâches successeurs

#### III.4.6.1 Principe de l'algorithme du réarrangement

Pour améliorer l'ordonnancement obtenu de l'algorithme 2, qui minimise le transfert de données provenant des prédécesseurs d'une tâche $v_i$, l'algorithme 3 proposé a pour objectif de minimiser le transfert de données pour les successeurs d'une tâche $v_i$. Le principe est de se fonder sur l'ordonnancement obtenu de l'algorithme 2 et de trouver une nouvelle machine qui peut réduire certains paramètres que nous verrons plus bas.

Cet algorithme commence par sauvegarder la date de début ($st^c(v_i)$) et le mapping ($M^i$) obtenu de l'ordonnancement précédant à partir de l'algorithme 2 pour chaque tâche $v_i$ de $V_l$ (lignes 2-3) ; $V_l$ étant l'ensemble des tâches du niveau $l$. L'étape suivante consiste à déterminer le volume local de données $LV_{i,j}$ pour une tâche $v_i$ sur une machine $M_j$ (lignes 4-6) ; $LV_{i,j}$ est la somme des données nécessaires en entrée stockées sur une machine $M_j$ et les données produites en sortie par une tâche $v_i$ et stockées localement, c'est-à-dire sur la machine $M_j$. Le volume de donnée local $LV_i^c$ pour le mapping obtenu de l'algorithme 2 est sauvegardé pour chaque tâche $v_i$ (ligne 7) avant d'annuler ce mapping (ligne 8).

L'annulation du mapping de toutes les tâches d'un niveau donné du DAG crée des créneaux libres (trous) dans les profils d'utilisation des différentes machines virtuelles et ceux-ci peuvent être utilisés pour améliorer la localisation des données en "déplaçant" certaines tâches d'une machine virtuelle à une autre. Les conditions pour déplacer une tâche $v_i$ d'une machine $M_j$ à une nouvelle machine $M_k$, doivent être défini de sorte que ce déplacement puisse améliorer la localisation des données ($LV_{i,k} \geq LV_i^c$) et doit réduire la date de début de la tâche, c'est-à-dire $st_k(v_i) \leq st^c(v_i)$.

La boucle principale de l'algorithme 3 (lignes 11-33) vise à améliorer de manière itérative l'ordonnancement des tâches du niveau $V_l$. Chaque itération permet de trouver un meilleur mapping (lignes 15-21) pour chaque tâche en considérant d'abord la machine virtuelle qui a la plus grande valeur du volume local $LV_{i,j}$. Si la tâche peut également commencer plus tôt sur cette machine, elle est sélectionnée pour un nouvel ordonnancement provisoire. Il y a trois cas de sortie de cette boucle : (i) il existe un meilleur mapping pour $v_i$ sur une autre machine $M_k$ ; (ii) $v_i$ a été remappé sur la même machine $M_j$ avec une date de début meilleure ou égale ; ou (iii) aucun meilleur mapping n'a été trouvé. Ce dernier cas signifie qu'une tâche plus prioritaire a été mappée sur $M_j$ et que $st^c(v_i)$ ne peut plus être garantie. Dans les deux autres cas, $v_i$ est ramenée à son mapping d'origine, qui devient définitif (lignes 23-26). Toutefois, cette décision peut invalider certains déplacement (par exemple une tâche avec un niveau de priorité plus élevé mappée sur la machine $M_j$). Toutes les ordonnancement provisoires déterminées à cette étape (lignes 28-32) sont annulés et le réarrangement des tâches restantes se poursuit.

L'algorithme 3 se termine lorsque toutes les décisions de déplacement sont prises au

**Algorithme 3** Réarrangement des tâches du niveau $l$.

**pour tout** $v_i \in V_l$ **faire**
    $st^c(v_i) \leftarrow$ start time de $v_i$ obtenu de l'algorithme 2
    $M^i \leftarrow$ mapping de $v_i$ obtenu de l'algorithme 2
    **pour tout** $M_j \in \mathcal{M}$ **faire**
        $LV_{i,j} \leftarrow LIV_{i,j} + LOVi, j$
    **fin pour**
    $LV_i^c \leftarrow$ volume de données local de $v_i$ obtenu de l'algorithme 2
    Annuler le mapping de $v_i$ obtenu de l'algorithme 2
**fin pour**
niveau_est_rearrangé $\leftarrow$ FAUX
**tant que** $\neg$ niveau_est_rearrangé **faire**
    niveau_est_rearrangé $\leftarrow$ VRAIE
    **pour tout** $v_i \in V_l$ **faire**
        Trier $\mathcal{M}$ par $LV_{i,j}$ décroissant
        **tant que** $LV_{i,j} \geq LV_i^c$ **faire**
            **si** $st_j(v_i) \leq st^c(v_i)$ **alors**
                Mapper $v_i$ sur $M_j$
                Mise à jour du *usage profile* de $M_j$
                **break**
            **fin si**
        **fin tant que**
        **si** $v_i$ est mappée sur $M^i$ **ou**
          $st_{M^i}(v_i) > st^c(v_i)$ **alors** ▷ Pas de meilleur mapping
            $V_l \leftarrow V_l \setminus \{v_i\}$ ▷ Le mapping est définitif
            niveau_est_rearrange $\leftarrow$ FAUX
        **fin si**
    **fin pour**
    **si** $\neg$ niveau_est_rearrangé **alors**
        **pour tout** $v_i \in V_l$ **faire**
            Annuler le mapping actuel de $v_i$
        **fin pour**
    **fin si**
**fin tant que**

cours de l'étape de réarrangement. Le niveau est alors considéré comme entièrement réarrangé et les mapping décidés deviennent définitifs.

## III.5 Evaluation des performances

La simulation permet d'imiter des phénomènes réels. Le processus de simulation nécessite un modèle. Le modèle est développé pour décrire les propriétés fondamentales du modèle simulé.La simulation est l'une des techniques d'évaluation des performances les

plus populaires dans les systèmes de workflows scientifiques. L'environnement informatique du cloud computing permet la mise en place dynamique de l'infrastructure pour le calcul haute performance.

La modélisation et la simulation du workflow scientifique jouent un rôle essentiel dans l'allocation des ressources dans un environnement distribué. La simulation est l'un des moyens sur lesquels l'on se fonde pour résoudre le problème d'ordonnancement des workflows scientifiques complexes dans un environnement distribué. Il existe de nombreux framework de simulation de workflow scientifique disponibles pour les environnements de grille calcul et de cloud computing.

Nous nous intéressons aux frameworks de simulation de cloud. Dans la section suivante, nous présentons le cadre expérimental ainsi que les différents frameworks utilisés.

### III.5.1 Frameworks de simulation utilisés

Pour mener à bien notre étude scientifique, nous nous sommes basés sur la simulation pour implémenter les différents algorithmes de notre approche. Nous avons étudié plusieurs simulateurs afin de trouver le mieux adapté aux hypothèses émises.

SimGrid [153] est un framework scientifique permettant d'étudier le comportement des systèmes distribués à grande échelle tels que les grilles de calcul, le cloud computing, les systèmes HPC et P2P [154]. Simgrid en lui-même n'est pas un simulateur mais plutôt un framework permettant de construire des simulateurs à évènements discrets. SimGrid fournit les fonctionnalités de base nécessaires à un simulateur et un certain nombre d'API utilisateur avec lesquelles il est possible d'interagir avec les fonctions de base. La Figure III.4 schématise la structure interne du framework SimGrid.

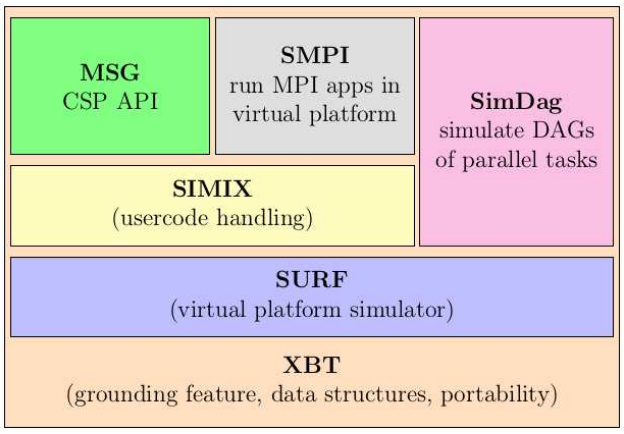



FIGURE III.4 – Architecture de SimGrid.

SimGrid a été sélectionné comme support pour WRENCH pour les raisons suivantes. SimGrid a été utilisé avec succès dans de nombreux domaines informatiques répartis (cluster, peer-to-peer, grille, cloud, etc.), et peut donc être utilisé pour simuler des systèmes de

gestion de workflow (WMS) qui s'exécutent sur une large gamme de plateformes. SimGrid est open source et disponible gratuitement. Il est stable depuis de nombreuses années et est activement développé. Il possède une communauté d'utilisateurs importante et a fourni des résultats de simulation pour plusieurs publications de recherche depuis sa création. La plupart des simulations SimGrids peuvent être exécutées en quelques minutes sur un ordinateur portable standard, ce qui permet d'effectuer rapidement un grand nombre de simulations avec un minimum de dépenses en ressources de calcul.

L'objectif de WRENCH est de permettre l'étude des systèmes de gestion de workflow (WMS) par la simulation de manière précise (modélisation fidèle des exécutions des applications parallèles du monde réel), évolutive (faible ressources de calcul et de mémoire sur un seul ordinateur) et expressive (capacité à simuler des scénarios arbitraires de WMS, de workflow et de plateforme avec un effort minimal de programmation). WRENCH n'est pas un simulateur mais un framework de simulation qui est fourni sous forme de bibliothèque C++ [155]. Il fournit des abstractions modifiables de haut niveau pour le développement et l'implémentations de WMS simulés et des simulateurs pour l'exécution de ces implémentations.

La Figure III.5 illustre l'architecture logicielle de WRENCH. Dans la couche inférieure se trouve le noyau de simulation qui simule les piles de logiciels et de matériel de bas niveau en utilisant les abstractions et les modèles de simulation fournis par SimGrid. La couche suivante met en œuvre des services de l'infrastructure informatique simulés que l'on trouve couramment sur les plateformes distribuées actuelles et qui sont utilisés par les WMS. Au moment de la rédaction de cette thèse, WRENCH fournit des services classés en quatre (4) catégories : (i) les services de calcul qui donnent accès à des ressources de calcul pour exécuter des tâches de workflows ; (ii) les services de stockage donnant accès à des ressources de stockage pour stocker des données de workflows ; (iii) les services de surveillance de réseau qui peuvent être interrogés pour déterminer les distances entre les nœuds ; (iv) et les services de registre de données qui peuvent être utilisés pour suivre l'emplacement des données du workflow. Par exemple, dans sa version actuelle, WRENCH fournit un service de calcul "*batch-scheduled cluster*", un service de calcul "*cloud*" et un service de calcul "*bare-metal*". La couche supérieure de l'architecture logicielle constitue un WMS simulé qui interagit avec les services de l'infrastructure informatique en utilisant les API de WRENCH.

Il existe plusieurs frameworks de simulation dans le domaine de calcul haut performance, mais deux frameworks complémentaires sont les plus adaptés au problèmes d'ordonnancement résolu dans cette thèse : les frameworks SimGrid [154] et Wrench [155]. Ces deux frameworks permettent de simuler un environnement réaliste et fournissent tous les éléments indispensables à l'exécution de workflows scientifiques. SimGrid et WRENCH permettent de mettre en place un simulateur qui répond à toutes les hypothèses émises

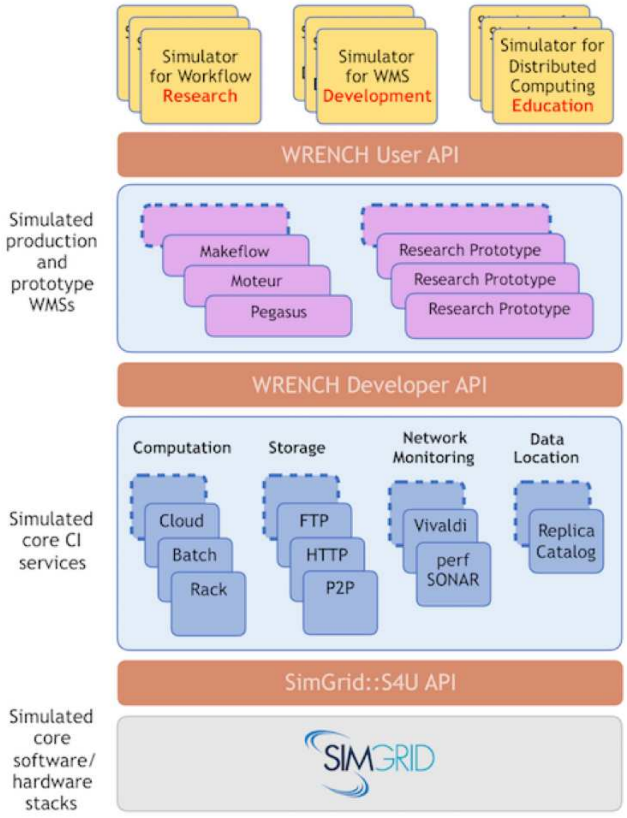



FIGURE III.5 – Architecture de WRENCH.

dans cette étude, tel que la création de plusieurs unité de stockage pour la répartition des données produites lors de l'exécution de l'application scientifique à forte intensité de données.

### III.5.2 Cadre expérimental

SimGrid est une framework de niveau inférieur pour créer des simulateurs de systèmes distribués. Sa conception ascendante signifie que SimGrid se préoccupe des *hosts*, de leurs processeurs, de leurs interfaces réseau et du partage de ses ressources entre les processus s'exécutant sur les *hosts*. De ce fait, SimGrid en lui-même offre des outils de simulation du cloud computing, tels que le partitionnement des ressources d'un *host* en machines virtuelles simulées. Du point de vue des fonctionnalités, les interfaces VM disponibles dans certains modules de SimGrid, correspondent aux fonctionnalités attendues d'un hyperviseur.

Il s'agit de la possibilité de créer une machine virtuelle en fonction d'un certain nombre de ressources physiques disponibles, de la possibilité de démarrer, de la suspension ou de l'arrêt de cette machine virtuelle simulée et de la capacité d'assigner des processus simulés à la machine virtuelle au lieu de l'*host* sous-jacent. Les machines virtuelles avec SimGrid

sont très abstraites et n'ont pas de temps de démarrage ou d'arrêt.

Une plateforme est constituée de plusieurs machines virtuelles interconnectées par le biais d'un commutateur via des bandes passantes. La bande passante qui lie une machine virtuelle au commutateur est proportionnelle au nombre de cœurs présent dans la machine virtuelle comme mentionné à la section III.2. Une machine virtuelle dispose également d'un service de stockage afin d'enrégistrer les fichiers produits en sortie par une tâche. Chaque machine virtuelle est directement liée au service de stockage EBS et la bande passante qui les lient est proportionnelle au nombre de cœurs de calcul de la machine virtuelle.

Pour générer les différentes plateformes utilisées lors de nos expériences, un script [1] en langage Python est utilisé. Ce script requiert le nombre total de cœurs de la plateforme et le nombre de cœurs par machine virtuelle. Une plateforme générée est un fichier XML [2] utilisée par SimGrid pour simuler un environnement réaliste du cloud IaaS d'Amazon EC2. Chaque plateforme dispose également d'une machine virtuelle pour le système de gestion de workflow, indispensable à WRENCH afin d'orchestrer l'ordonnancement. Généralement, les machines virtuelles du système de gestion de wokflow et de l'EBS ont pour identifiant respectif id=00000 et id=00001 et sont dotées d'un seul cœur de calcul.

Pour implémenter un algorithme d'ordonnancement, WRENCH dispose de plusieurs APIs afin de faciliter la programmation. Nous avons utilisé quelques uns [3] parmi ceux-ci pour mettre en place les différents algorithmes comparés.

### III.5.3 Critère d'évaluation

Le critère utilisé ici pour comparer notre algorithme sans réarrangement (algorithme 2) et notre algorithme avec réarrangement (algorithme 3) est le gain (exprimé en pourcentage) sur le makespan des deux algorithmes proposés. Le makespan de l'algorithme 2 étant la référence, le gain est ainsi l'impact qu'a le makespan de l'algorithme 3 par rapport à l'algorithme 2 sur différentes plateformes. Pour évaluer la performance de notre approche avec réarrangement et un algorithme de la littérature (HEFT), nous utilisons que le makespan.

1. Voir annexe A
2. Voir annexe B
3. Voir annexe C

## III.6 Résultats et discussion

### III.6.1 Détermination des données transférées

Le Tableau III.2 donne une description de la quantité de données produites par les différents workflows utilisés dans notre étude et le nombre de cœurs utilisés en parallèle. Ces éléments sont indispensables pour la suite de notre étude.

Pour générer les différentes plateformes et éviter le gaspillage de ressources, c'est-à-dire éviter de louer des ressources qui ne seront pas utilisées, il est indispensable de déterminer le nombre de tâches pouvant être exécutées en parallèle pour chacun des workflows scientifiques. Pour ce faire, la plateforme considérée a été surdimensionnée, c'est-à-dire que la plateforme considérée a autant de cœurs qu'il y a de tâches dans le workflow. Dans le cas des workflows utilisés dans cette étude, il y a mille (1000) tâches ; ainsi la plateforme considérée a mille (1000) cœurs de calcul. Cela permet de déterminer le nombre total de cœurs pouvant être utilisés en parallèle pour chacun des workflows. Ainsi, pour CyberShake, trois cent soixante-quatorze (374) cœurs sont utilisés en parallèle tandis que pour Epigenomics et Montage ce sont respectivement deux cent quarante six (246) et six cent soixante-deux (662) cœurs utilisés en parallèle (comme illustré dans le Tableau III.2). Cette étude préliminaire sera utilisée dans les sections suivantes afin de déterminer la limite de plateforme à utiliser pour chaque workflow dans les expériences.

L'ordonnancement produit par les algorithmes 2 et 3 minimise la quantité de données transférées sur le réseau pendant l'exécution du workflow. Cependant, la qualité de cet ordonnancement dépend fortement de l'ensemble des machines virtuelles multi-cœurs qui possèdent chacune un espace de stockage. L'objectif des algorithmes proposés est de minimiser le makespan du workflow scientifique par la réduction des données à transférées sur le réseau connaissant la plateforme d'exécution.

Le Tableau III.3 présente le makespan (en secondes) et le volume de donnée transféré (en Go [4]) pour trois (3) applications à forte intensité de données. Les résultats obtenus d'une première simulation récapitulée à travers le Tableau III.2, c'est-à-dire pour le workflow CyberShake par exemple, trois cent soixante-quatorze (374) cœurs de calcul sont utilisés en parallèle, et donc trois cent soixante-quatorze (374) tâches s'exécutent en parallèle. Connaissant le nombre total de cœurs utilisés en parallèle, et faisant varier le nombre total de cœur par machines virtuelles, c'est-à-dire pour trois cent soixante-quatorze (374) cœurs total, et deux (2) cœurs par VM, la plateforme est constituée de cent quatre-vingt-sept (187) VMs. Pour quatre (4) cœurs par VM, la plateforme est constituée de quatre-vingt-treize (93) VMs de quatre (4) cœurs et une (1) VM de deux (2) cœurs. Pour quatre-vingt-

4. Voir annexe D

seize (96) cœurs par VM, la plateforme est constituée de trois (3) VMs de quatre-vingt-seize (96) cœurs, une (1) VM de soixante-quatre (64) cœurs, une (1) VM de seize (16) cœurs, une (1) VM de quatre (4) cœurs et une (1) VM de deux (2) cœurs. Ce même principe a été appliqué pour les deux autres workflows scientifiques, c'est-à-dire Epigenomics et Montage.

Tableau III.2 – Mesure du volume de données total et le nombre de cœur utilisé en parallèle.

| **Workflow** | **Volume Total (Go)** | **Cœur Max** |
|---|---|---|
| CyberShake | 400,39 | 374 |
| Epigenomics | 1230,93 | 246 |
| Montage | 17,32 | 662 |

Chaque VM, en plus de posséder des unités de calcul (cœur), possède une unité de stockage SSD pour enregistrer les données produites en sortie par chaque tâche sur la VM où elle s'est exécutée. Ce principe est appelé stockage décentralisé.

Tableau III.3 – Impact du réarrangement sur le volume de données transféré et le makespan.

| | | **Sans Réarrangement** | | **Avec Réarrangement** | |
|---|---|---|---|---|---|
| **Workflow** | **cœurs/VM** | **Makespan** | **Volume transféré** | **Makespan** | **Volume transféré** |
| CyberShake | 2 | 5075,5 | 396,383 | 3192,7 | 394,696 |
| | 4 | 4467,09 | 395,769 | 2589,082 | 394,082 |
| | 8 | 2314,96 | 392,935 | 962,288 | 390,693 |
| | 16 | 1247,98 | 386,609 | 593,825 | 382,703 |
| | 32 | 684,051 | 372,869 | 382,869 | 368,407 |
| | 48 | 574,942 | 358,89 | 369,413 | 354,556 |
| | 64 | 410,759 | 345,036 | 303,11 | 339,912 |
| | 96 | 351,101 | 317,234 | 311,026 | 315,57 |
| Epigenomics | 2 | 46020 | 1222,58 | 34357,6 | 1222,23 |
| | 4 | 42809,7 | 1222,5 | 34277,3 | 1222,12 |
| | 8 | 34245 | 1222,41 | 34235,7 | 1221,93 |
| | 16 | 34231,6 | 1222,13 | 34227,2 | 1221,65 |
| | 32 | 34311,4 | 1221,51 | 34271,8 | 1221,06 |
| | 48 | 34327,1 | 1220,94 | 34326,7 | 1220,84 |
| | 64 | 34297,6 | 1220,44 | 34301,5 | 1220,29 |
| | 96 | 34314,2 | 1219,84 | 34330,4 | 1219,81 |
| Montage | 2 | 417,854 | 11,1314 | 417,877 | 11,0771 |
| | 4 | 402,649 | 10,976 | 402,571 | 10,8753 |
| | 8 | 394,922 | 10,7123 | 394,459 | 10,5418 |
| | 16 | 391,107 | 10,5027 | 390,375 | 10,2854 |
| | 32 | 387,399 | 9,92851 | 386,837 | 9,50955 |
| | 48 | 387,947 | 9,5797 | 386,85 | 9,04469 |
| | 64 | 386,048 | 9,03648 | 385,683 | 8,50923 |
| | 96 | 385,566 | 8,21447 | 385,244 | 7,54779 |

Les résultats obtenus de l'algorithme sans réarrangement est l'algorithme 2 en ignorant

les lignes 8-11, et les résultats obtenus de l'algorithme avec réarrangement est la combinaison des algorithmes 2 et 3. En appliquant le principe de stockage décentralisé, dans l'approche avec réarrangement, l'algorithme évite de transférer jusqu'à 21,33% de données pour le workflow scientifique CyberShake et 0,9% et 49,36% de données sont évités d'être transférer pour les workflows Epigenomics et Montage respectivement, par rapport à l'approche sans réarrangement.

### III.6.2 Impact des grosses VMs sur le makspan

Dans la section précédente, nous avons remarqué que notre approche avec réarrangement qui combine les algorithmes 2 et 3 permet d'éviter de transférer certaines données sur le réseau.

Pour chaque workflow, nous considérons les infrastructures où nous faisons varier (c'est-à-dire augmenter) le nombre maximum de cœurs par VM (de deux (2) cœurs à quatre-vingt-seize (96) cœurs maximum). Pour un nombre total de cœurs à utiliser, nous générons des plateformes avec deux (2) cœurs par VM, quatre (4) cœurs par VM, ..., quatre-vingt-seize (96) cœurs par VM.

Sur les Figures III.6, III.7 et III.8, les plateformes composées de deux (2) cœurs par machine virtuelle fournissent des temps d'exécution médiocres par rapport aux plateformes de trente-deux (32) cœurs par machine virtuelle et elles-mêmes fournissent des temps d'exécution médiocres par rapport aux plateformes de quatre-vingt-seize (96) cœurs par machine virtuelle. Nous pouvons conclure qu'en augmentant le nombre total de cœurs par VM, nous obtenons de bons makespan. C'est pour cette raison que dans notre étude, nous privilégions les grosses machines virtuelles (c'est-à-dire les machines virtuelles avec plusieurs cœurs) car ces machines virtuelles peuvent exécuter plusieurs tâches en parallèle et réduire considérablement le temps d'exécution des applications à forte intensité de données.

L'impact qu'a l'augmentation du nombre de cœurs sur le makespan est certainement trivial car favorisant l'exécution simultanée de plusieurs tâches, mais le problème qui se pose est de savoir comment mapper chaque tâche du workflow en fonction de ses prédécesseurs et ses successeurs en tenant compte des différentes données produites en sortie par chacune des tâches pour aboutir à un bon makespan. Sur les Figures III.6-III.8, *max_ft* représente la borne minimale s'il n'y a pas eu de transfert de données lors de l'exécution du workflow.

Les plateformes considérées pour ces simulations ont un nombre total de cœurs qui est un multiple de quatre-vingt-seize (96). Par exemple, pour les workflows CyberShake et Epigenomics, nous avons considérés respectivement les nombres totaux de cœurs suivants : quatre-vingt-seize (96), cent quatre-vingt-douze (192), deux cent quatre-vingt-huit

(288), trois cent quatre-vingt-quatre (384) et quatre-vingt-seize (96), cent quatre-vingt-douze (192), deux quatre-vingt-huit (288). Car ces deux workflows utilisent respectivement trois cent soixante-quatorze (374) et deux cent quarante six (246) cœurs en parallèle. Pour Montage, nous allons jusqu'à six cent soixante-douze (672), c'est-à-dire que nous utilisons quatre-vingt-seize (96), cent quatre-vingt-douze (192), deux cent quatre-vingt huit (288), trois cent quatre-vingt quatre (384), quatre cent quatre-vingt (480), cinq cent soixante-seize (576) et six cent soixante-douze (672) cœurs puisque six cent soixante deux (662) cœurs sont utilisés en parallèle pour ce workflow.

Nous notons ici que le temps d'exécution obtenu sur les plateformes avec de grosses VMs (c'est-à-dire ayant quatre-vingt-seize (96) cœurs) se rapproche de cette borne minimale théorique à laquelle nous nous comparons. Ici, nous appliquons l'étape de réarrangement de l'ordonnancement hors ligne effectuée avec l'algorithme 3 pour mesurer les performances de la planification hors ligne initiale uniquement.

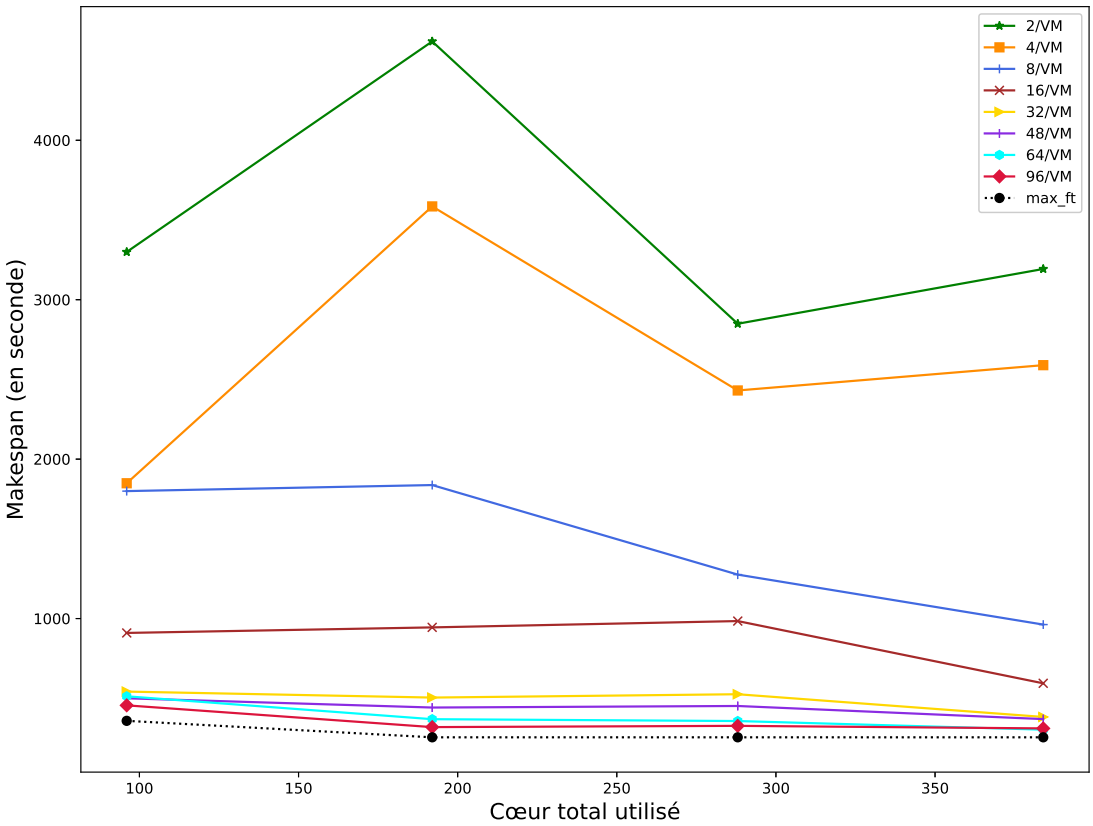



FIGURE III.6 – Évolution du makespan en augmentant le nombre total de cœur par VM avec le workflow CyberShake.

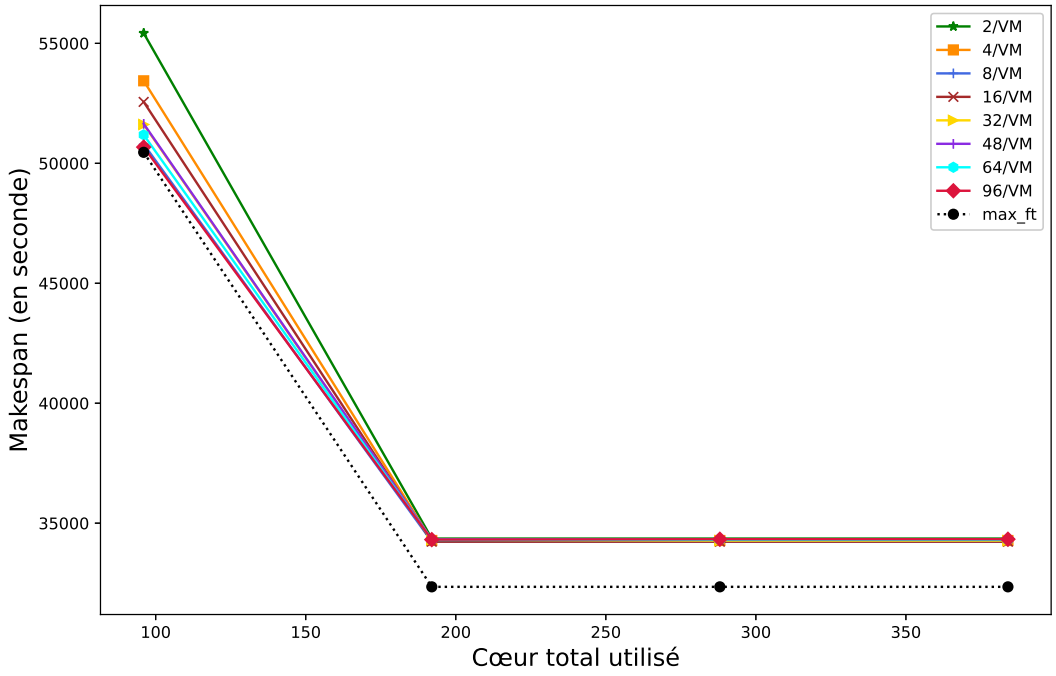



FIGURE III.7 – Évolution du makespan en augmentant le nombre total de cœur par VM avec le workflow Epigenomics.

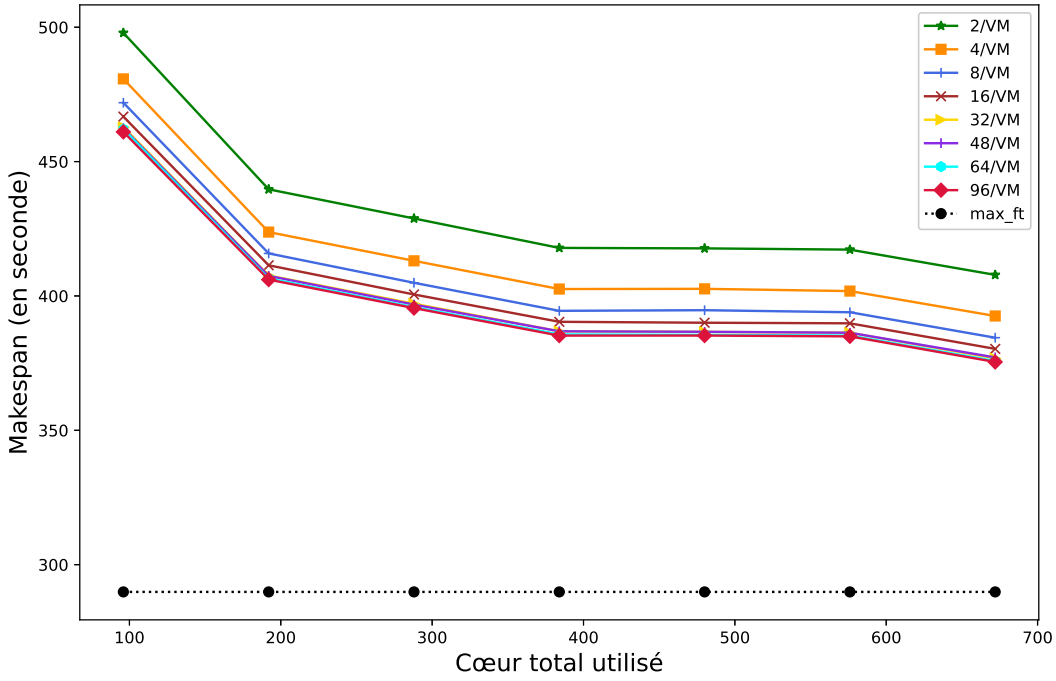



FIGURE III.8 – Évolution du makespan en augmentant le nombre total de cœur par VM avec le workflow Montage.

### III.6.3 Étude comparative des algorithmes 2 et 3

Deux approches ont été proposées : l'approche sans réarrangement (Algorithme 2 en ignorant les lignes 8 à 11) qui vise à minimiser les transferts de données provenant des

prédécesseurs d'une tâche et l'approche avec réarrangement (qui combine les Algorithmes 2 et 3), dont l'objectif est de réduire les transferts de données provenant des prédécesseurs et allant vers les successeurs d'une tâche.

Le gain de l'algorithme 3 par rapport à l'algorithme 2 se traduit par l'équation suivante :

$$gain(\%) = \frac{mksp_3 - mksp_2}{mksp_2} \times 100 \tag{III.8}$$

où $mksp_2$ et $mksp_3$ sont respectivement le makespan obtenu par les algorithmes 2 et 3 sur la même plateforme d'exécution. $mksp_2$ est pour la majorité des cas, supérieur à $mksp_3$. Le gain est calculé par rapport à $mksp_2$, qui représente la référence des makespans obtenus.

Les Figures III.9 - III.11 montrent l'impact de l'algorithme avec réarrangement sur le temps d'exécution. Pour l'application Montage (Figure III.11), la structure du workflow est telle que le réarrangement n'a aucune influence sur l'ordonnancement hors ligne et sur le temps d'exécution. Pour les deux autres workflows CyberShake et Epigenomics (respectivement Figures III.9 et III.10), la réorganisation de l'ordonnancement hors ligne permet d'avoir des gains sur le makespan allant jusqu'à 60% pour CyberShake et 40% pour Epigenomics. Ce gain sur le makespan s'explique par les données qui ont été évitées d'être transférées sur le réseau. L'amélioration observée est plus significative pour les petites tailles d'instances.

Changer le mapping de certaines tâches pour favoriser le stockage local réduit efficacement le nombre de transferts sur le réseau et donc les effets de contention vus dans les Figures III.6 - III.8. Pour des tailles d'instances plus grandes, l'impact du réarrangement est moins important mais toujours significatif (supérieur à 5%) pour CyberShake.

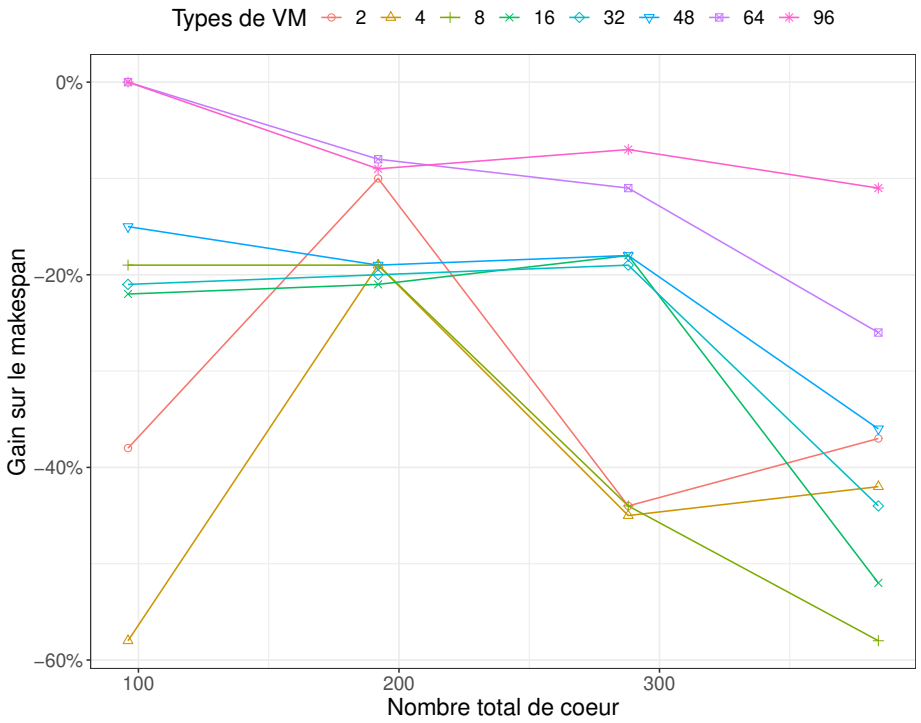



FIGURE III.9 – Gain sur le makespan de l'algorithme 3 par rapport à l'algorithme 2 avec le workflow CyberShake.

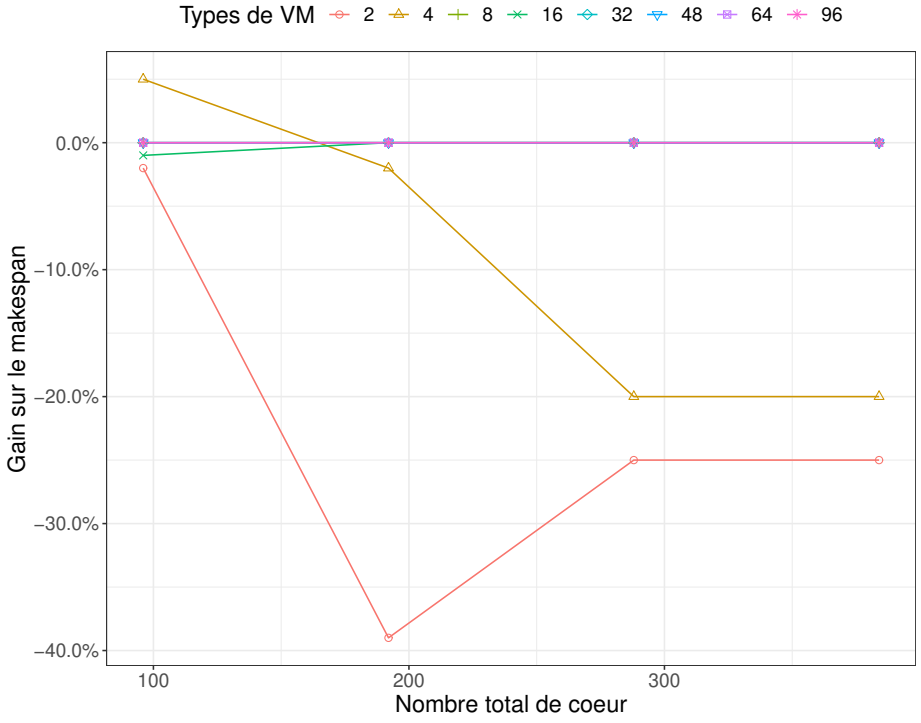



FIGURE III.10 – Gain sur le makespan de l'algorithme 3 par rapport à l'algorithme 2 avec le workflow Epigenomics.

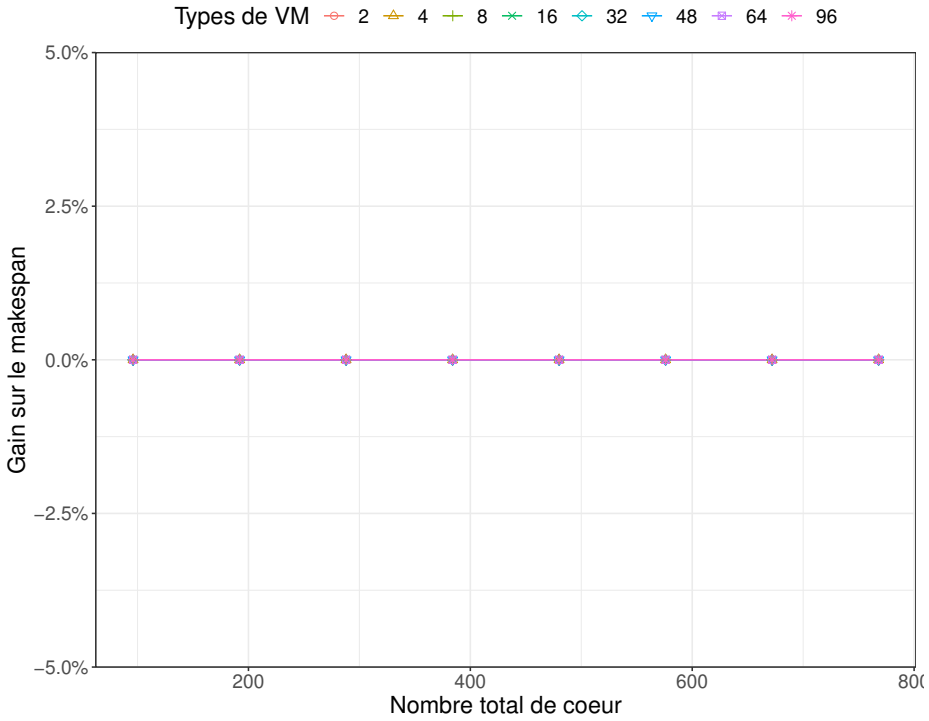



FIGURE III.11 – Gain sur le makespan de l'algorithme 3 par rapport à l'algorithme 2 avec le workflow Montage.

### III.6.4 Étude comparative des algorithmes WSRDT et HEFT

Dans la suite de cette thèse, nous dénommons cette approche proposée qui combine les algorithmes 2 et 3 ***Workflow Scheduling Reducing Data Transfers*** (WSRDT).

L'objectif principal de notre étude est de minimiser le temps d'exécution d'une application à forte intensité de données. Pour atteindre cet objectif, notre principale contribution est la réduction des données à transférer pendant l'exécution de l'application, ce qui a un impact considérable sur son temps d'exécution. Afin d'évaluer le makespan produit par notre algorithme, nous comparons notre approche à celle de HEFT [113] qui est une heuristique très populaire dans l'ordonnancement d'applications parallèles. Pour cela, nous avons adapté HEFT aux ressources multi-cœurs du cloud IaaS et à notre environnement de simulation. Plusieurs plateformes ont permis d'effectuer ces simulations. Le nombre total de cœurs par plateforme varie de deux (2) cœurs, à trois cent soixante quatorze (374) cœurs pour l'application CyberShake, à deux cent quarante six (246) cœurs pour Epigenomics et à six cent soixante deux (662) cœurs pour Montage, comme l'illustre le Tableau III.2 par pas de deux (2) cœurs. Le nombre de simulation est ainsi de cent quatre vingt sept (187), cent vingt trois (123) et trois cent trente un (331) respectivement pour Cybershake, Epigenomics et Montage. Chaque plateforme comporte moins de VMs possible et plus de cœurs par VM. Pour une plateforme dont le nombre total de cœurs est fixé à cent (100)

par exemple, cette plateforme comporte une (1) VM de quatre vingt seize (96) cœurs et une (1) VM de quatre (4) cœurs, plutôt que cinquante (50) VMs de deux (2) cœurs. Cela permet d'une part d'exécuter le maximum de tâches en parallèle et d'autre part de stocker le plus de données sur la même VM afin d'éviter le transfert de certaines données. Pour les applications à forte intensité de données, il est certes important de tenir compte du temps de traitement de chaque tâche de l'application, mais il est encore plus important de considérer le temps de transfert des données entre les tâches.

Les résultats de nos simulations à travers les Figures III.12 - III.14 montrent que notre approche donne de bons résultats par rapport à HEFT. Le makespan obtenu par notre algorithme sur chacune des plateformes est inférieur au makespan obtenu par HEFT sur la même plateforme. Cela s'explique par le fait que WSRDT est un algorithme "clairvoyant", car cette approche, en plus de s'adapter à la plateforme de simulation, permet à chaque tâche du workflow de s'adapter à la localisation des données avant le mapping de celle-ci sur la bonne VM.

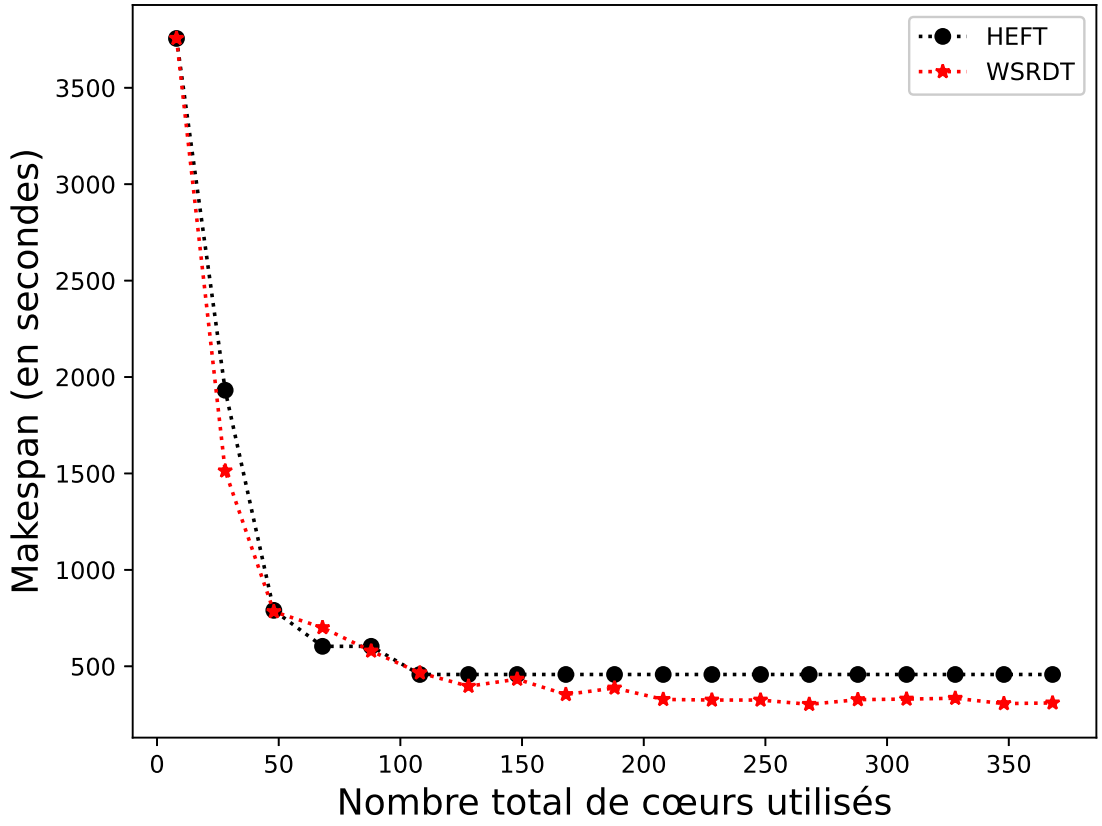



FIGURE III.12 – Évaluation des algorithme WSRDT et HEFT avec le workflow CyberShake.

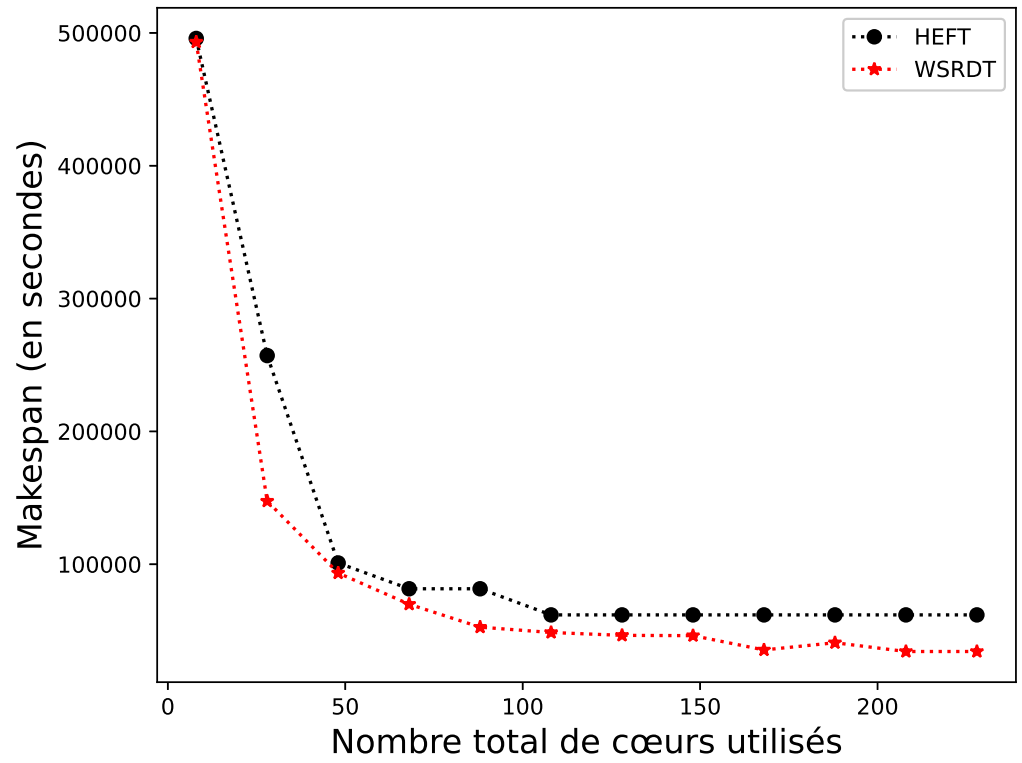



FIGURE III.13 – Évaluation des algorithme WSRDT et HEFT avec le workflow Epigenomics.

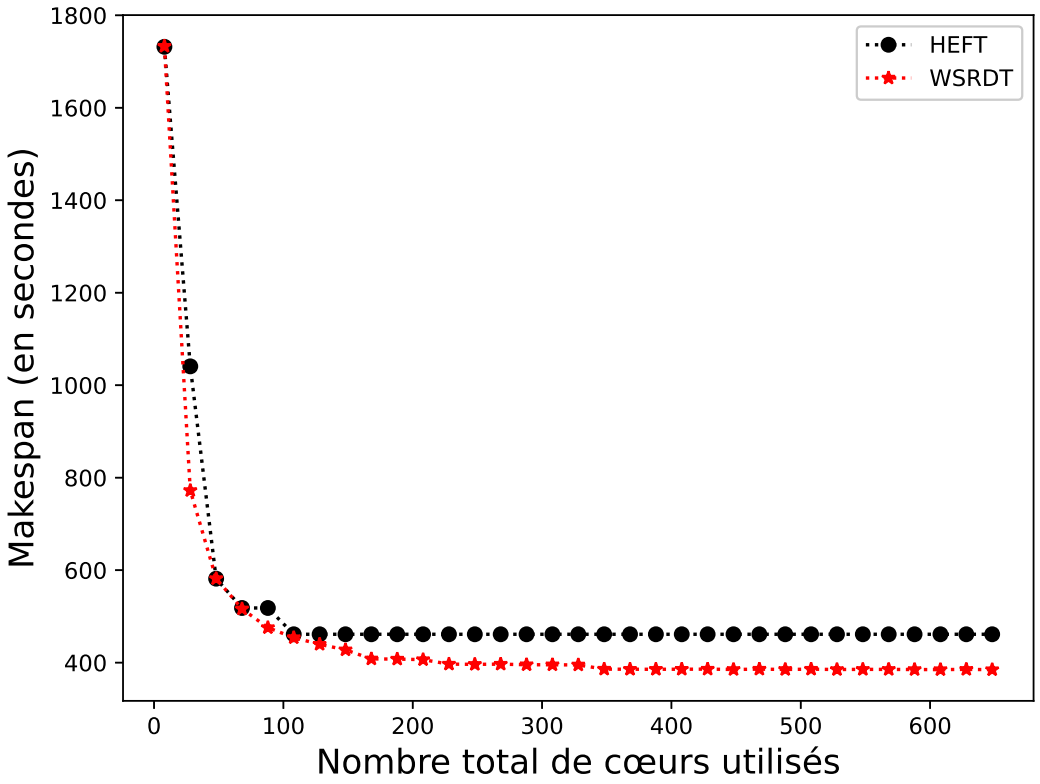



FIGURE III.14 – Évaluation des algorithme WSRDT et HEFT avec le workflow Montage.

## III.7 Conclusion partielle

Dans ce chapitre, nous avons mis au point deux algorithmes d'ordonnancement dont l'objectif est de minimiser le makespan par la réduction des données transférées provenant des prédécesseurs d'une part, et allant vers les successeurs d'autre part. Pour valider les hypothèses émises, nous avons effectué plusieurs simulations. Parmi celles-ci, nous remarquons que l'augmentation du nombre de cœurs par machine virtuelle a un impact considérable sur le makespan compte tenu des tâches qui s'exécutent en parallèle, mais surtout du stockage local des données produites en sortie par les tâches. Ainsi, dans la suite de nos travaux nous avons priorisé les grosses machines virtuelles, c'est-à-dire celles avec le plus de cœurs tout en se basant sur le nombre total de cœurs de la plateforme. Nous avons par la suite comparé nos deux algorithmes et nous remarquons que l'algorithme 3 donne de meilleures performances par rapport à l'algorithme 2. Pour terminer cette étude nous avons comparé l'algorithme 3 à HEFT qui était le meilleur dans le domaine de l'ordonnancement. Nous remarquons que notre approche fournit de bons résultats par rapport à HEFT.

Un tel algorithme (WSRDT), c'est-à-dire qui minimise le makespan, peut servir à dimensionner la plateforme afin de proposer à l'utilisateur des configurations de plateforme qui offre de bon compromis entre le makespan et le coût d'utilisation des ressources du cloud IaaS.

Chapitre IV

# Détermination des configurations offrant un bon compromis entre le makespan et le coût

## Sommaire

## IV.1 Introduction partielle

Les services du cloud computing, en particulier le cloud IaaS apparaît comme une infrastructure appropriée pour l'exécution des workflows scientifiques. Le cloud IaaS permet d'accéder à un pool de VMs, chacune ayant ses propres ressources (de calcul : cœurs, stockage, etc.). Exécuter les différentes tâches d'un workflow scientifique sur chacune des VMs est un grand défi, car l'on ne sait pas d'avance la localisation des ressources et des données en raison de leur répartition géographique. Par conséquent, un problème majeur est de savoir comment exécuter un workflow scientifique sur des ressources du cloud IaaS afin de minimiser à la fois le makespan et le coût d'exploitation des ressources du cloud computing.

L'ordonnancement orienté données produit par les algorithmes 2 et 3 minimise la quantité de données transférées sur le réseau pendant l'exécution du workflow scientifique. Cependant, la qualité de cette planification dépend fortement de l'ensemble des machines virtuelles multi-cœurs fournies en entrée.

La question à résoudre dans ce chapitre est de savoir comment déterminer un ensemble de machines virtuelles qui réalise un bon compromis entre le temps d'exécution du workflow scientifique et le nombre de ressources à louer auprès d'un fournisseur de cloud IaaS. À cette fin, un simulateur basé sur le projet WRENCH [155, 156] a été proposé et offrant un framework de simulation Cyber-Infrastructure. Ce framework fournit des abstractions de simulation de haut niveau pour construire des simulateurs à part entière précis et évolutifs avec un minimum d'efforts de développement logiciel.

Dans ce chapitre, nous présentons quelques concepts et définitions de l'optimisation multi-objectif à la section IV.2, suivie de la description du problème à la section IV.3. A la section IV.4 nous présentons le cadre expérimental de ce travail, avec les différents résultats obtenus qui sont par la suite discutés à la section IV.5.

## IV.2 Optimisation multi-objectif

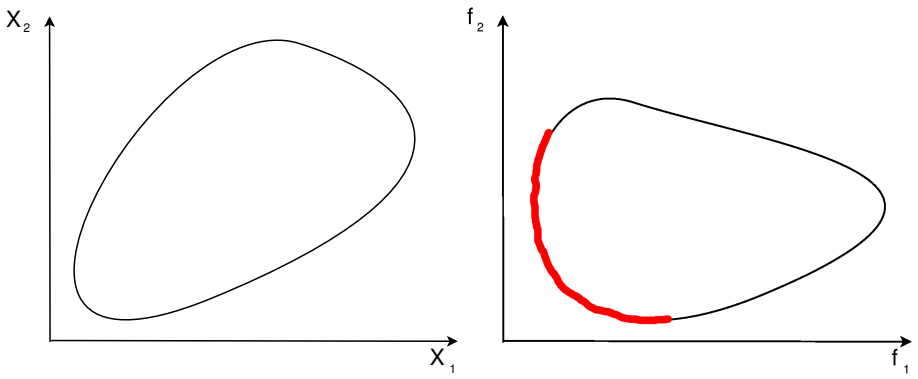
Les expériences scientifiques à grande échelle tirent généralement parti des workflows pour modéliser les opérations de données telles que le chargement des données d'entrée, le traitement et l'analyse des données intermédiaires et l'agrégation des données de sortie. Les workflows permettent de représenter le traitement des données de ces expériences sous la forme d'un graphe orienté acyclique (DAG) dans lequel les sommets représentent les tâches de traitement des données et les arcs des dépendances de données entre les différentes tâches. Un workflow est l'assemblage de tâches de traitement de données scientifiques avec des dépendances. Une tâche est la description d'une activité qui constitue une étape logique au sein d'une représentation de l'application parallèle. Puisque les tâches de

cette application traitent de grandes quantités de données, il convient de mieux exploiter la répartition de ces données sur les ressources du cloud computing, pour éviter que certaines tâches ne les téléchargent sur une ressource de stockage distante, plutôt que de les récupérer localement sur la VM où la tâche est prévue s'exécuter.

Afin d'exécuter les workflows scientifiques efficacement, ceux-ci exploitent généralement les ressources de calcul haute performance (HPC) dans un environnement de cluster, de grille ou de cloud computing. En raison de ressources pratiquement infinies, de la diversité des services élastiques, d'une qualité de services stables et d'une politique de paiement flexibles, le cloud computing est devenu une solution intéressante pour l'exécution de workflows scientifiques.

Plusieurs études de la littérature ont proposé une approche multi-objectif pour résoudre le problème d'ordonnancement de workflows scientifiques. Celle de Durillo et Prodan [121] (MOHEFT) est l'une des meilleures études pour évaluer la performance de notre algorithme. MOHEFT est un algorithme de *list scheduling*, c'est-à-dire l'algorithme travaille sur une liste de tâches triées par priorité tout comme HEFT [113]. Les auteurs affirment que MOHEFT est une extension de HEFT, mais leur algorithme propose $K$ solutions d'ordonnancement, contrairement à HEFT qui ne propose qu'une seule. A chaque itération $i$, une nouvelle tâche est sélectionnée dans la liste des tâches triées et un workflow partiel est par la suite constitué. Ainsi MOHEFT construit $P_i$ solutions qui sont des extensions des $P_{i-1}$ solutions non-dominées obtenues à l'itération précédente, en tenant compte de la $i^{ème}$ tâche. Parmi ces $P_i$ combinaisons, l'algorithme conserve au maximum $K$ solutions ($K \leq P_i$ ). Cela permet d'éviter une explosion combinatoire du nombre de solutions intermédiaires trouvées à la fin de chaque itération. Si plus de $K$ solutions ont été produites, MOHEFT utilise la méthode de distance d'encombrement [114] pour sélectionner les $K$ meilleures solutions. Les solutions sont triées par distance d'encombrement décroissante et l'algorithme sélectionne les $K$ premières solutions. La distance d'encombrement donne une mesure de la zone entourant une solution où aucune autre solution de compromis n'est placée. Ce critère permet de sélectionner les solutions ayant une distance d'encombrement élevée. L'ensemble représente une zone plus large de solutions de compromis différentes.

Une solution peut ne pas être bonne à une itération donnée, mais peut permettre d'obtenir de bons ordonnancements aux itérations suivantes. A cet effet, il devient important de modifier les critères de sélection de MOHEFT. Pour mieux justifier la performance de notre approche, nous avons adapté MOHEFT afin de tenir compte de l'infrastructure de la plateforme, qui est constituée de machines virtuelles multi-cœurs et différents services de stockage.




(a) Espace décisionnel (b) Espace objectif

FIGURE IV.1 – Espaces décisionnel et objectif d'un POM.

## IV.2.1 Concepts et définitions

Les principaux concepts, définitions et notations liés au domaine de l'optimisation multi-objectif, tels que la relation de dominance, l'optimalité et le front de Pareto sont présentés dans cette section.

### IV.2.1.1 Formulation du problème d'optimisation multi-objectif

**Définition IV.1** Un problème d'optimisation multi-objectif (POM) peut être défini par :

$$POM = \begin{cases} Min\ F(x) = (f_1(x), \ldots, f_n(x)) \\ s.c.\ x \in X \end{cases} \quad \text{(IV.1)}$$

où $n$, le nombre d'objectifs est généralement supérieur ou égal à deux ($n \geq 2$), $x = (x_1, \ldots, x_n) \in X$ représente le vecteur de $k$ variables de décision et $X$ est l'ensemble de solutions réalisables dans l'espace décisionnel. A chaque solution $x \in X$, est associé un vecteur objectif $y \in Y$ basé sur un vecteur de fonctions : $f : X \to Y$ tel que $y = (y_1, \ldots, y_n) = f(x) = (f_1(x), \ldots, f_n(x))$. $Z = f(X)$ représente l'ensemble des points réalisables dans l'espace objectif. Le lien qui existe entre l'espace décisionnel et l'espace objectif est représenté par la Figure IV.1. Contrairement au problème d'optimisation mono-objectif, la solution d'un problème d'optimisation multi-objectif n'est pas unique, mais c'est un ensemble de solutions non dominées, connu comme l'ensemble des solutions Pareto optimales.

### IV.2.1.2 Notions de dominance

Dans le cadre des problèmes d'optimisation multi-objectif, l'évaluation d'une solution est faite sur la base de chaque critère et de sa place dans l'espace objectif. Cependant,

en raison de la nature conflictuelle des objectifs, il n'existe généralement pas de solution unique qui est à la fois optimale pour chaque objectif, contrairement au problème d'optimisation mono-objectif où il existe une relation d'ordre entre les solutions réalisables. Ainsi, une relation d'ordre partiel est généralement définie, appelée relation de dominance.

**Définition IV.2 : Dominance de Pareto.** Un vecteur objectif $y \in Y$ domine un vecteur objectif $y' \in Y$ si les deux conditions suivantes sont vérifiées :

$$\forall i \in \{1, \ldots, n\},\ y_i \leq y'_i \tag{IV.2}$$

$$\exists j \in \{1, \ldots, n\},\ y_j < y'_j \tag{IV.3}$$

Si $y$ domine $y'$, cette relation est notée $y \prec y'$. Par extension, une solution $x \in X$ domine une solution $x' \in X$, notée $x \prec x'$, si et seulement si $f(x) < f(x')$.

La notion de dominance est illustrée par la Figure IV.2. Outre la relation de dominance de Pareto, il existe d'autres relations de dominance énoncées ci-dessous.

**Définition IV.3 : Dominance faible.** Un vecteur objectif $y \in Y$ domine faiblement un vecteur objectif $y' \in Y$, si et seulement si $\forall i \in \{1, \ldots, n\},\ y_i \leq y'_i$. Cette relation est notée $y \preceq y'$.

**Définition IV.4 : Dominance stricte.** Un vecteur objectif $y \in Y$ domine strictement un vecteur objectif $y' \in Y$, si et seulement si $\forall i \in \{1, \ldots, n\},\ y_i < y'_i$. Cette relation est notée $y \prec y'$.

**Définition IV.5 : $\varepsilon$-dominance.** Un vecteur objectif $y \in Y$ $\varepsilon$-domine un vecteur objectif $y' \in Y$ avec $\varepsilon$>1, si et seulement si $\forall i \in \{1, \ldots, n\},\ y_i < y'_i$. Cette relation est notée $y \leq \varepsilon.y'$. Cette relation est notée $y \preceq \varepsilon y'$.

**Définition IV.6 : non-dominé** Un vecteur objectif $y \in Y$ est non dominé, si et seulement si $\forall y' \in Y$, $y' \not\prec y$, comme l'illustre la Figure IV.2.

#### IV.2.1.3 Optimalité de Pareto

Le front de Pareto peut être considéré comme un outil d'aide à la décision et de découverte de solutions de préférence. Sa forme fournit un aperçu permettant dans de nombreux cas d'explorer l'espace possible de solutions non dominées avec certaines propriétés et éventuellement des régions d'un intérêt particulier qui ne peuvent être vues avant le front de Pareto. Un bon front de Pareto est celui qui fournit une précision (solutions proches des

optimales) et une diversité (couvre toutes les gammes possibles de solutions optimales). L'une des approches utilisées pour répondre à la diversité des solutions est le *crowding distance*.

**Définition IV.7 : Pareto optimale.** Une solution $x \in X$ est Pareto optimale (ou non-dominée) si et seulement si $\forall x' \in X$, $x' \not\prec x$.

Une solution Pareto optimale est une solution qui améliore la valeur d'un objectif sans dégrader celle d'au moins un autre objectif. Toutes les solutions Pareto optimales constituent un ensemble exact de solutions noté $X_E$.

**Définition IV.8 : Ensemble Pareto optimal.** Etant donné un POM $(f, X)$, l'ensemble Pareto optimal est : $X_E = \{x \in X | \nexists x' \in X,\ x' \prec x\}$.

**Définition IV.9 : Front de Pareto.** Etant donné un POM $(f, X)$ et son ensemble Pareto optimal $X_E$ le front de Pareto est : $Z_N = \{f(x) | x \in X_E\}$.

La Figure IV.2 illustre le front de Pareto et la notion de dominance. Ainsi, le point noir est dominé par les triangles et domine les cercles. Par contre il est incomparable aux carrés.

**Définition IV.10 : La distance d'encombrement.** La distance d'encombrement (*crowding distance*) est l'aire de la zone entourant une solution où aucune autre solution de compromis n'est placée, c'est-à-dire pour une solution $i$, la zone constituée de son prédécesseur $i-1$ et de son successeur $i+1$ immédiat. La Figure IV.3 illustre la distance d'encombrement de la solution $i$.

La distance d'encombrement permet de diversifier les solutions et de mesurer la qualité d'un ensemble de solutions de compromis.

### IV.2.2 Optimisation multi-objectif et aide à la décision

Résoudre un problème d'optimisation multi-objectif permet de trouver un ensemble de solutions Pareto optimales. Il est cependant nécessaire de faire intervenir l'être humain à travers un outil d'aide à la décision pour le choix final de la solution optimale. Les décisions qui doivent être prises lors de la résolution du problème multi-objectif portent sur le processus décisionnel et la façon de combiner les techniques de recherche de solutions. L'une des trois méthodes suivantes peuvent être utilisée pour atteindre cet objectif.

#### IV.2.2.1 Méthode *a priori*

Dans ce cas, le décideur intervient en aval du processus d'optimisation pour fournir des préférences sur le problème à résoudre, afin d'aider la méthode de résolution dans sa re-

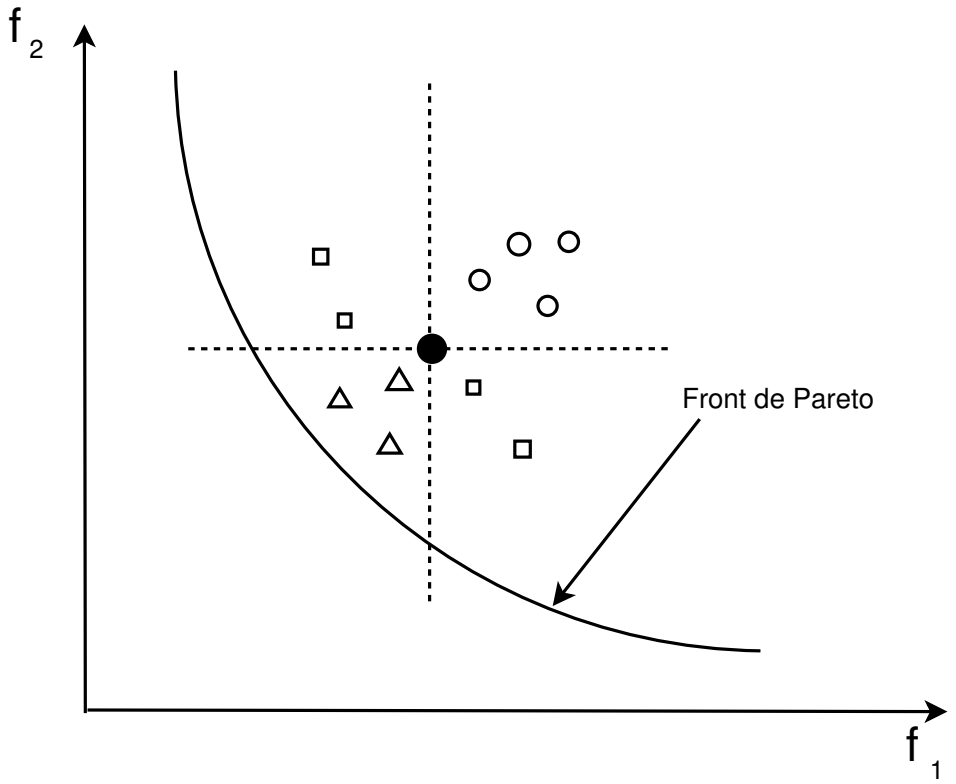



FIGURE IV.2 – Front de Pareto et relation de dominance.

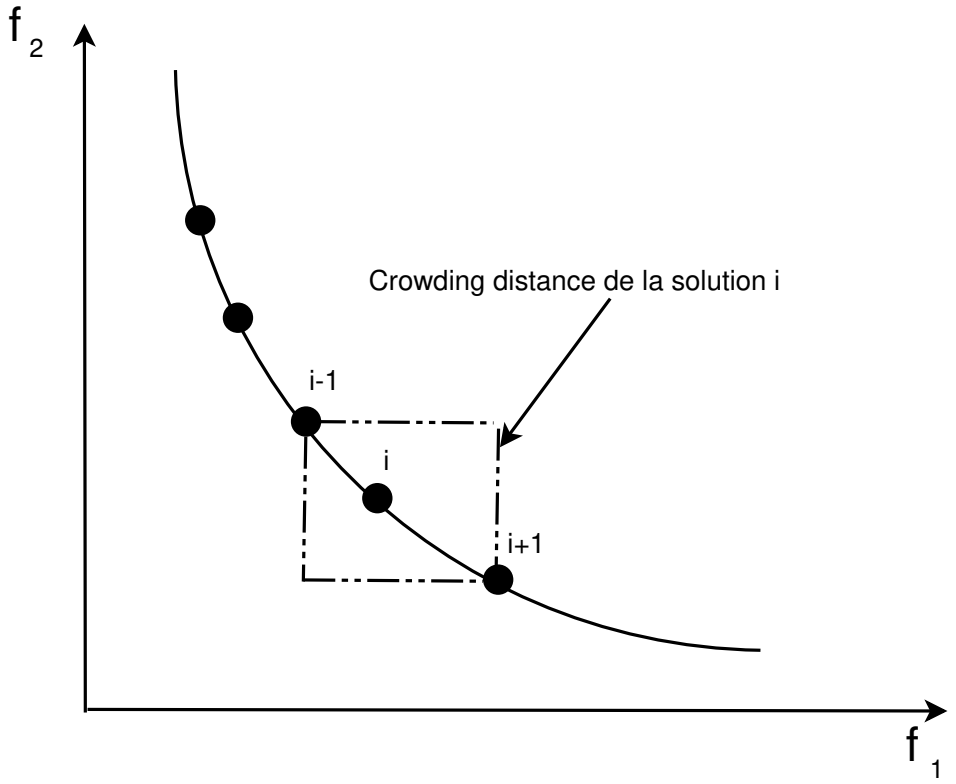



FIGURE IV.3 – Comparaison des solutions du front de Pareto.

cherche. En pratique, ceci revient à transformer le problème d'optimisation multi-objectif du départ en un problème mono-objectif. Ce dernier peut être résolu par une méthode dont une seule exécution permettra d'obtenir la solution recherchée. Cette approche nécessite une bonne connaissance a priori du problème. Elle est rapide, mais il faut cependant prendre en compte le temps de modélisation du problème et la possibilité pour le décideur de ne pas être satisfait de la solution trouvée et de relancer la recherche avec une autre formulation de ce problème.

#### IV.2.2.2 Méthode *a posteriori*

La méthode de résolution fourni au décideur un ensemble de solutions Pareto optimales bien réparties. Il peut alors, au regard de cet ensemble de solutions, choisir celle qui lui semble la plus appropriée au vu de ses préférences. Ainsi il n'est plus nécessaire de modéliser les préférences du décideur, mais il faut désormais fournir un ensemble de solutions bien réparties, ce qui peut être difficile et coûteux en termes de temps de calcul.

#### IV.2.2.3 Méthode itérative

Dans cette approche, il existe une coopération directe et itérative entre le décideur et la méthode de résolution. Le décideur intervient pendant la recherche en ajustant ses préférences afin de guider la recherche. Cette approche permet de bien prendre en compte les préférences du décideur, mais nécessite sa présence tout au long du processus de recherche.

## IV.3 Description du problème

La recherche de bonnes configurations passe par plusieurs simulations. Par conséquent, nous effectuons un ensemble exhaustif de simulations couvrant tous les nombres possibles de cœurs allant de deux (c'est-à-dire la plus petite taille de machine virtuelle) à mille (c'est-à-dire le nombre de tâches). L'extraction de l'ensemble de configurations sur le front de Pareto peut être fait dans un temps abordable (moins d'une heure dans nos expériences). Néanmoins, le temps d'extraction peut être supérieur à la durée réelle d'exécution du workflow scientifique. Nous proposons par la suite une stratégie pour réduire le nombre de simulations nécessaires pour retourner un ensemble restreint de solutions Pareto optimales pour aider l'utilisateur à dimensionner l'infrastructure.

Nous commençons par simuler l'exécution du workflow scientifique sur une configuration avec autant de cœurs que de tâches le composant. Cette exécution sur une configuration surdimensionnée permet de déterminer $C_{max}$, le nombre maximal de cœurs pouvant être exploités simultanément. Ensuite, nous déterminons deux limites inférieures impor-

tantes. Dans un premier temps, nous simulons l'exécution du workflow sur $C_{max}$ cœurs afin d'obtenir une approximation du temps d'exécution minimum réalisable, $E_{C_{max}}$. Deuxièmement, nous simulons l'exécution du workflow scientifique sur seulement deux cœurs pour obtenir une approximation du coût minimum, $C_2$ (c'est-à-dire le prix à payer pour une instance à deux cœurs exécutant le workflow scientifique). Nous utilisons ces deux valeurs pour déterminer quelles pourraient être les bonnes configurations candidates pour l'exécution du workflow scientifique. En effet, nous supposons arbitrairement qu'une bonne configuration ne doit pas dégrader $E_{C_{max}}$ et $C_2$, formaliser comme suit :

$$E_i \leq 2 \times E_{c_{max}} \textbf{ et } C_i \leq 2 \times C_2 \tag{IV.4}$$

où $E_i$ et $C_i$ sont respectivement le temps d'exécution et le coût d'exploitation de ressources sur une configuration avec $i$ cœurs.

Pour identifier les configurations candidates, nous effectuons une recherche dichotomique qui s'arrête soit lorsqu'il n'y a plus de configurations à tester soit nous avons trouvé deux solutions consécutives qui respectent la condition ci-dessus (Equation IV.4). Au cours de cette recherche dichotomique, nous enregistrons également le nombre de cœurs correspondant à la première solution candidate trouvée. Cela forme un deuxième espace de recherche avec le nombre de cœurs pour lesquels la recherche dichotomique s'est arrêtée. Ensuite, nous simulons de manière exhaustive l'exécution du workflow pour tout le nombre de cœurs dans ce deuxième espace de recherche.

Enfin, nous sélectionnons l'ensemble des configurations à retourner à l'utilisateur en déterminant les solutions non dominées qui composent le front de Pareto parmi les résultats de la recherche exhaustive. Les deux configurations extrêmes avec deux et $C_{max}$ cœurs sont également renvoyées à l'utilisateur.

### IV.3.1 Makespan

La métrique utilisée dans l'évaluation des algorithmes 2 et 3 d'ordonnancement de workflow scientifique est le temps de complétion ou makespan. Le makespan est la différence entre la date de réception des résultats et la date de soumission de l'application parallèle, c'est-à-dire la date de fin de la dernière tâche ($v_{exit}$). Le makespan est représenté par l'équation suivante :

$$makespan = \max_{v_i \in V}(ft(v_i)) \tag{IV.5}$$

où $ft(v_i)$ est la date de fin (*finish time*) d'exécution de la tâche $v_i$.

Généralement, le makespan se compose du temps de transferts des données sur le réseau et du temps d'exécution des tâches. Le temps de transferts des données sur le réseau dépend, à la fois, de la taille des données transmises et de la latence du réseau. Le temps

d'exécution dépend à la fois de la charge de travail et des performances de machines virtuelles (dans notre cas les machines virtuelles sont homogènes en terme de puissance de calcul).

### IV.3.2 Coût

Les fournisseurs mettent à la disposition des utilisateurs plusieurs types de ressources à la demande, généralement selon le principe du paiement à l'utilisation. Le caractère payant du cloud computing signifie que l'utilisateur doit être en mesure de prévoir la quantité de ressources (calcul, stockage, bande passante) nécessaires par unité de temps (par exemple en heure). Le coût total d'exécution d'un workflow scientifique sur les ressources IaaS du cloud computing est défini par l'Equation IV.6. Ce coût est composé du coût des ressources de calcul pour exécuter chaque tâche du workflow scientifique $C_{ex}$ et de celui de stockage et de transfert des données $C_{tr}$.

$$Coût = C_{ex} + C_{tr} \tag{IV.6}$$

Le coût d'utilisation des ressources IaaS du cloud computing pour exécuter une tâche $v_i$, est fonction du temps de traitement de la tâche sur la machine virtuelle $M_j$ qui lui est allouée et du prix unitaire de cette machine. Quant au coût de transfert, il dépend du coût de stockage et de la bande passante utilisée.

Au moment de la rédaction de cette thèse, nous avons considéré le coût par cœur et par unité de temps (en heure) égal à 0,0565 dollar. Ce coût comporte également le coût de stockage de la machine virtuelle et de la bande passant qui relie la machine virtuelle au reste du réseau. Ainsi le coût d'exploitation d'une VM de 96 cœurs est de 0,0565 dollar $\times$ 96, soit 5,424 dollars.

Généralement, le makespan et le coût sont inversement proportionnels, c'est-à-dire la minimisation du makespan (respectivement le coût) dégrade le coût (respectivement le makespan). Pour résoudre ce problème, il existe plusieurs approches, entre autres la méthode de Pareto. Cette méthode permet de trouver un compromis entre deux objectifs contradictoires, dans notre cas, entre le makespan et le coût.

## IV.4 Cadre expérimental

Nous décrivons dans cette section les expériences réalisées pour valider le front de Pareto obtenu des algorithmes 2 et 3 présentés au chapitre III, dont l'objectif est de minimiser le temps d'exécution du workflow scientifique par la réduction du transfert de données

sur le réseau. Nous dénommons cette approche qui prend en compte les données et basée sur la simulation "*Data-Aware and Simulation-Driven*" (DASD) pour l'ordonnancement de workflows scientifiques sur les ressources IaaS du cloud computing. Tout d'abord, nous résumons les critères de comparaison pour évaluer la qualité de notre approche. Ensuite, nous décrivons les différents types de workflows et l'infrastructure Amazon EC2 pris en compte dans les expériences.

Nous considérons deux critères pour comparer la qualité des solutions candidates par notre approche et MOHEFT. Tout d'abord, nous considérons les deux temps d'exécution $E_{C_{max}}$ et $E_{C_2}$ représentant respectivement la borne minimale et maximale des solutions produites. Ensuite, nous nous focalisons sur l'aspect économique des ressources utilisées lors de l'exécution du workflow en analysant la solution la moins chère rapportée par chaque algorithme. L'idée de ces deux indicateurs est d'évaluer le comportement des différentes approches en optimisant chaque critère individuel. Enfin, nous considérons la valeur de la distance d'encombrement pour évaluer la qualité des solutions de compromis candidates.

Nous analysons également les solutions de compromis calculées par MOHEFT pour différents types de workflows. L'idée est d'étudier l'équilibre entre les deux objectifs en conflit, le makespan et le coût d'exploitation des ressources cloud, et combien peut être gagné dans un objectif en détériorant l'autre. Pour cette analyse, nous nous appuyons sur une représentation graphique des solutions calculées par les deux algorithmes. Les graphiques présentent l'ordonnancement le plus efficace en termes de makespan et continuent le long du front de Pareto vers la solution la moins chère. Par conséquent, bien que MOHEFT calcul le compromis entre le coût et le makespan afin de mettre en évidence le potentiel des résultats obtenus, cette approche présente les économies de coût relativement à la détérioration du makespan en pourcentage par rapport à la solution la plus efficace en makespan, calculée par HEFT.

Enfin, les auteurs de MOHEFT prêtent attention au nombre et au type d'instances sélectionnées par les différentes solutions d'ordonnancement calculées. Il convient de mentionner que MOHEFT et HEFT sont des algorithmes non stochastiques, et donc, la même solution est calculée dans des exécutions différentes.

## IV.5 Résultats et discussion

Puisque l'exécution de workflow scientifique peut prendre beaucoup de temps et coûter beaucoup d'argent, le problème d'ordonnancement peut avoir plusieurs objectifs. Ainsi, ce dernier doit prendre en compte l'impact des ressources réparties sur différents datacenters ; Par exemple la taille des instances louées et les différentes bandes passantes correspondantes ainsi que la distribution de données sur différentes VMs.

Les algorithmes d'ordonnancement présentés au chapitre III permettent de minimiser le makespan en tenant compte des caractéristiques sus-mentionnées. Un tel algorithme peut permettre de proposer à un utilisateur la bonne configuration de plateforme qui offre un bon compromis entre le makespan et coût d'exploitation des ressources du cloud computing pour exécuter son application data-intensive.

Par conséquent, le lancement d'un ensemble exhaustif de simulations couvrant tous les nombres possibles de cœurs, d'un minimum de deux (c'est-à-dire la plus petite taille de machine virtuelle) à mille (c'est-à-dire le nombre de tâches) et l'extraction de l'ensemble des configurations sur le front de Pareto pourrait être fait dans un temps abordable (moins d'une heure dans nos expériences), mais cela peut également être supérieur à la durée réelle d'exécution du workflow scientifique. Ensuite, une stratégie pour réduire le nombre de simulations nécessaires pour retourner un ensemble restreint de solutions Pareto pour aider l'utilisateur à dimensionner l'infrastructure est proposé.

### IV.5.1 Évaluation des solutions du Front de Pareto

Cette section présente l'évaluation de la qualité des solutions du front de Pareto produites par les algorithmes 2 et 3 présentés au chapitre III. Ces algorithmes ont montré que pour minimiser le temps d'exécution pour un nombre fixe de cœurs, la priorité devrait être donnée aux grandes instances de VM. Pour obtenir le front de Pareto donné par cette approche, il faudrait lancer une simulation pour chaque nombre total de cœurs et éliminer toutes les solutions dominées.

Le front de Pareto obtenu à partir de cette approche est comparé à celui obtenu d'une version modifiée de l'algorithme MOHEFT proposé par Durillo et Prodan [121]. Les auteurs de MOHEFT ont montré que leur algorithme donne de meilleurs résultats que les algorithmes concurrents. Cependant, MOHEFT est initialement conçu de telle sorte que chaque tâche du workflow utilise tous les cœurs de la machine virtuelle sur laquelle elle est mappée. La modification apportée à MOHEFT permet de s'adapter aux hypothèses émises dans cette recherche, c'est-à-dire une instance de machine virtuelle est partagée par plusieurs tâches séquentielles.

La modification de MOHEFT pour répondre au problème d'ordonnancement résolu dans cette thèse où les VMs sont partagées par plusieurs tâches peut produire des solutions équivalentes. Deux solutions $S_i$ et $S_j$ sont équivalentes si les mêmes tâches sont mappées ensemble sur des machines virtuelles de même type, c'est-à-dire ayant le même nombre de cœurs et le même *start time* ($st(v_i)$) pour chacune des tâches. Pour pallier ce problème, dès la première itération (il y a une seule tâche), la première tâche est mappée sur différents types de VMs, c'est-à-dire des machines de 2 cœurs/VM, 4 cœurs/VM, etc. selon les types de VmM disponibles dans l'infrastructure virtuelle. Pour les autres itérations, un seul

représentant de ces solutions équivalentes est conservé. De plus, dans toutes les itérations sauf la dernière, seules les solutions strictement dominées sont éliminées. Une solution $S_i$ est strictement dominée par une solution $S_j$ si le temps d'exécution et le coût de $S_i$ sont strictement supérieurs à ceux de $S_j$. Cela empêche d'éliminer les solutions intermédiaires qui peuvent conduire à de bons ordonnancements. Dans la dernière itération, toutes les solutions, simplement dominées, sont éliminées ; C'est-à-dire qu'une seule métrique (par exemple le makespan ou le coût) doit être plus élevée pour que la solution soit considérée comme dominée.

Les Figures IV.4 - IV.6 montrent respectivement les fronts de Pareto des workflows CyberShake, Epigenomics et Montage produits par les algorithmes proposés dans cette thèse par rapport à ceux produits par MOHEFT. Le front de Pareto de MOHEFT est représenté par les points rouges et celui de notre approche est représenté par les points noirs. Malgré une valeur $K$ élevée et un temps de calcul d'une dizaine d'heures pour chaque workflow, les fronts Pareto de MOHEFT sont souvent dominés par ceux de l'approche proposée. De plus, MOHEFT conduit à moins diversifié malgré l'utilisation de la distance d'encombrement qui permet de sélectionner les meilleurs solutions.

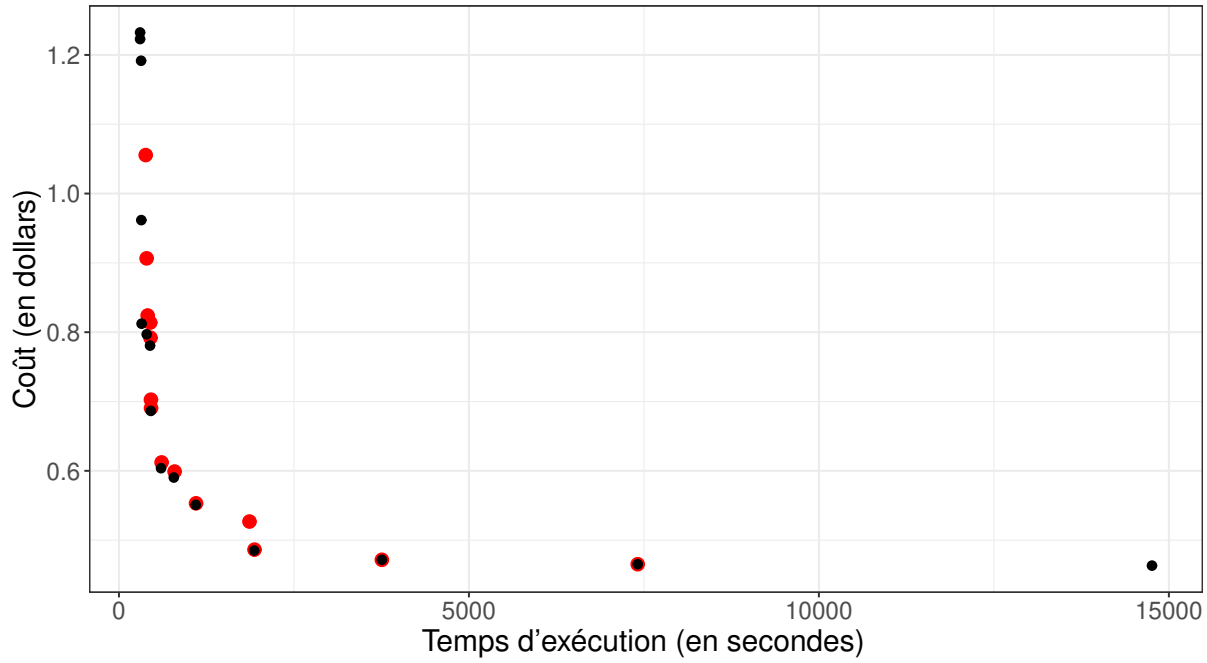



FIGURE IV.4 – Fronts de Pareto MOHEFT (en rouge) vs DASD (en noir) avec le workflow CyberShake.

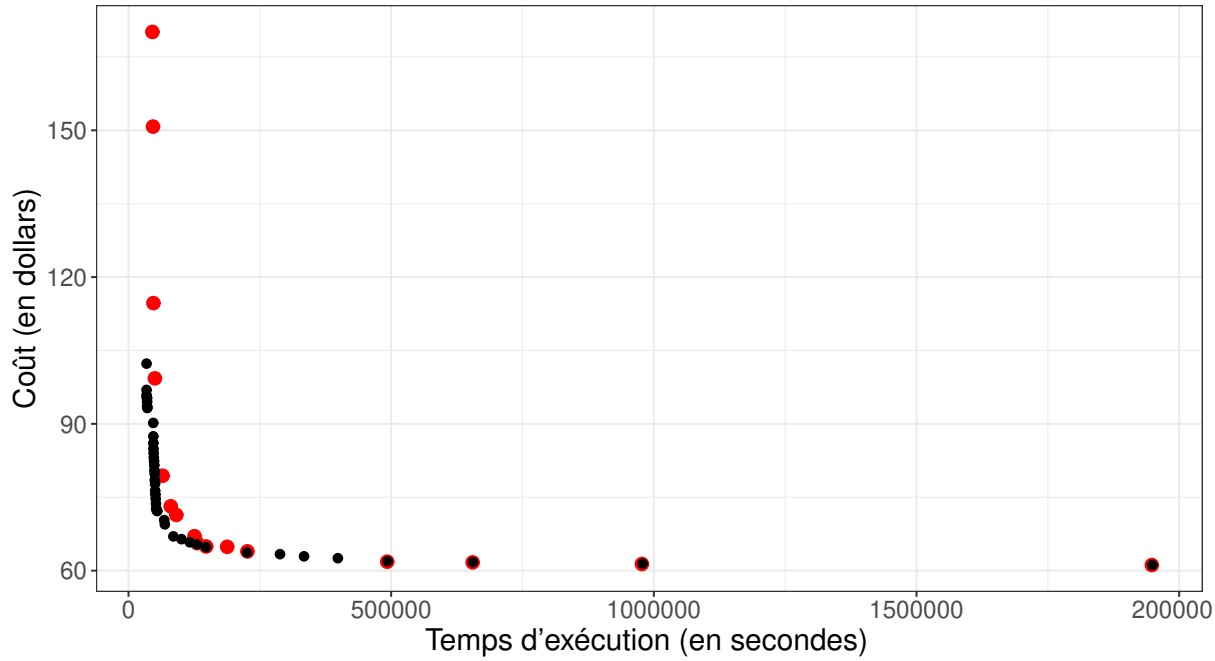



FIGURE IV.5 – Fronts de Pareto MOHEFT (en rouge) vs DASD (en noir) avec le workflow Epigenomics.

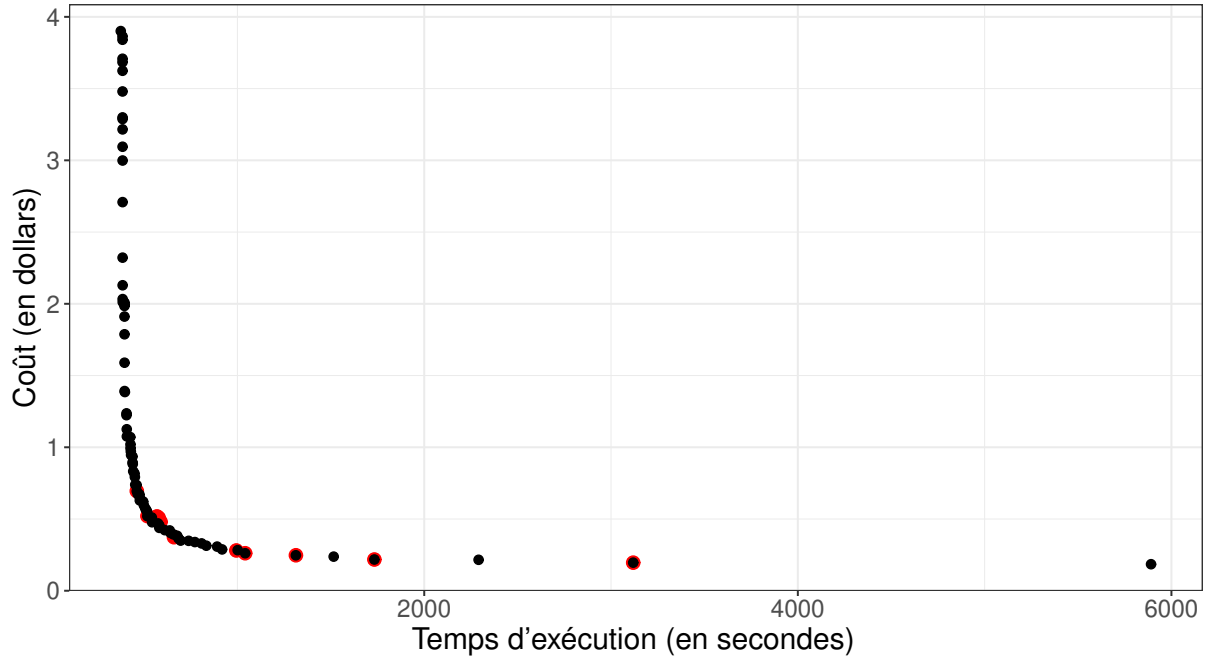



FIGURE IV.6 – Fronts de Pareto MOHEFT (en rouge) vs DASD (en noir)avec le workflow Montage.

### IV.5.2 Recherche par la simulation d'un ensemble efficace de VMs

Grâce à WRENCH, le temps nécessaire pour créer un ordonnancement orienté données et simuler son exécution sur un ensemble d'instances de machine virtuelle donné est très faible. Les expériences détaillées dans la suite de cette thèse montrent que le temps de

simulation varie de 2,28 à 50,88 secondes avec une moyenne d'environ 13,4 secondes. La taille du workflow étant fixe, le temps de simulation dépend directement du nombre de cœurs dans l'infrastructure virtuelle. Pour un nombre total donné de cœurs dans l'infrastructure simulée, la sélection de l'ensemble d'instances est traitée comme un "problème du rendu de monnaie" (*Change-making problem*). Ce problème s'énonce de la façon suivante : au vu d'un système de monnaie (pièces et billets), comment rendre une somme donnée de façon optimale, c'est-à-dire avec le nombre minimal de pièces et billets ? Dans cette étude, il s'agit donc de trouver le nombre minimal de VMs repondant au nombre total de cœurs de l'infrastructure virtuelle. Un algorithme glouton simple est par la suite utilisé pour résoudre ce problème et trouver le nombre minimal de VMs.

Cette section illustre le fonctionnement de la détermination par simulation afin de proposer un ensemble rentable d'instances de machine virtuelle. Le Tableau IV.1 montre les résultats, pour chacun des workflows considérés, des trois premières simulations visant à déterminer le degré maximal de parallélisme ($C_{max}$), c'est-à-dire le nombre maximal de cœurs utilisés en parallèle. Le temps d'exécution atteint lors de l'utilisation de ce nombre particulier de cœurs est $E_{C_{max}}$ et le coût minimum qu'un utilisateur doit payer pour exécuter le workflow sur une seule instance à deux cœurs est représenté par $C_2$. $E_{C_2}$ est le temps d'exécution de l'application sur une instance de deux cœurs, c'est-à-dire la borne maximale.

Tableau IV.1 – Détermination des métriques de base.

| **Workflow** | $\mathbf{C_{max}}$ | $\mathbf{E_{C_{max}}}$ | $\mathbf{C_2}$ | $\mathbf{E_{C_2}}$ |
|---|---|---|---|---|
| CyberShake | 374 | 310,266s | $0,463 | 4h 06m |
| Epigenomics | 246 | 9h 31m 54s | $61,17 | 541h 20m |
| Montage | 662 | 375,45s | $0,185 | 1h 38m 11s |

Les workflows scientifiques CyberShake et Montage peuvent se terminer en environ cinq (5) ou six (6) minutes pour un budget inférieur à un dollar (1$). Cependant, ces deux applications nécessitent de récupérer plusieurs données différentes pour produire des résultats scientifiques. Malgré ces temps d'exécution courts et leur faible coût, l'optimisation des deux métriques est pertinente. Le workflow scientifique Epigenomics a un temps d'exécution beaucoup plus long (plusieurs heures) ce qui implique un budget plus important pour une seule exécution, mais aussi plusieurs possibilités d'optimisation.

Les Figures IV.7, IV.8 et IV.9 montrent toutes les solutions (configurations) qui remplissent la condition de ne pas dégrader le meilleur temps d'exécution et le meilleur coût pour les workflows scientifiques respectifs : CyberShake, Epigenomics et Montage. Ces figures distinguent les configurations qui sont réellement simulées lors des recherches dichotomiques et exhaustives (représentées par des triangles) de celles qui ont été ignorées par l'algorithme proposé (représentées par des cercles). Les triangles et les cercles plus grands

(rouges) identifient les configurations sur le front de Pareto, tandis que les triangles et les cercles plus petits (noirs) sont les configurations dominantes. Enfin, les étiquettes indiquent le nombre respectif de cœurs de chaque configuration sur le front de Pareto. Pour des raisons de lisibilité, les étiquettes pour les configurations dominantes simulées pour le workflow scientifique Epigenomics sont respectivement cent quatre-vingt (180), cent soixante-dix-huit (178), cent soixante-quatorze (174), cent soixante-douze (172), cent soixante-dix (170), cent soixante-huit (168), cent soixante-six (166) (pour les triangles rouges).

Pour le workflow scientifique CyberShake (Figure IV.7), la recherche dichotomique commence à $C_{max}$ = 374 et s'arrête à cent vingt-huit (128) après huit itérations. La première configuration rencontrée qui remplit les conditions est constituée de quatre-vingt-quatorze (94) cœurs. Cependant, cette plateforme est constituée d'une seule machine virtuelles de quatre-vingt-seize (96) cœurs selon l'algorithme glouton utilisé pour résoudre le problème du rendu de monnaie. Ensuite, la simulation exhaustive de toutes les configurations entre quatre-vingt-seize (96) et cent vingt-six (126) cœurs est lancée. Parmi ces configurations, seules trois sont sur le front de Pareto avec respectivement quatre-vingt-seize (96), cent douze (112) et cent vingt-huit (128) cœurs. L'algorithme proposé ignore également trois configurations dominantes avec soixante-quatre (64), cent quarante-quatre (144) et cent soixante (160) cœurs. Celui avec soixante-quatre (64) cœurs n'est clairement pas une bonne solution, car cela conduit à un temps d'exécution beaucoup plus long. Les deux autres configurations ignorées sont de meilleures solutions prétendantes et auraient également pu être retournées à l'utilisateur. Avec cent soixante (160) cœurs, le temps d'exécution est réduit de soixante-douze (72) secondes pour une augmentation de coût de 0,019 dollar par rapport à la meilleure configuration retournée par l'algorithme avec cent vingt-huit (128) cœurs.

Pour le workflow scientifique Epigenimics (Figure IV.8), $C_{max}$ = 246. La recherche dichotomique n'a besoin que de deux itérations pour trouver deux configurations candidates consécutives avec cent vingt-quatre (124) et cent quatre-vingt-six (186) cœurs respectivement. Ensuite, trente configurations doivent être simulées dans la recherche exhaustive de candidats. Parmi ces configurations, seules trois sont sur le front de Pareto avec respectivement cent vingt-quatre (124), cent soixante-dix (170) et cent soixante-douze (172) cœurs. L'algorithme proposé ignore environ vingt configurations dominantes, sauf une ayant moins de cent vingt-quatre (124) cœurs. Cependant, elles ne peuvent pas être considérés comme intéressantes pour l'utilisateur. En effet, il y a un gain important sur le temps d'exécution lorsque l'on passe de cent vingt-quatre (124) à cent soixante-dix (170) cœurs ($\approx$ 3,5 heures) pour une augmentation de coût de seulement 0,42 dollar. La dernière configuration ignorée avec cent quatre-vingt-dix (190) cœurs augmente le coût de près de dix dollars (10$) mais réduit le temps d'exécution de moins de cinq secondes (5 sec.) par rapport à la configuration à cent soixante-quatorze (174) cœurs que l'approche

dichotomique proposée sélectionne.

Pour le workflow scientifique Montage (Figure IV.9), la sélection se fait entièrement par la recherche dichotomique. En partant de $C_{max} = 662$, il se termine en trouvant deux configurations candidates consécutives avec trente-deux (32) et trente-quatre (34) cœurs. Il n'est donc pas nécessaire d'effectuer une recherche exhaustive. Nous pouvons voir sur la Figure IV.9 que seules trois configurations répondent aux exigences initiales émises. Parmi eux, l'algorithme favorise clairement la réduction du temps d'exécution. La configuration sélectionnée avec trente-quatre (34) cœurs réduit le temps d'exécution de soixante-quatorze secondes (74 sec.) pour un coût supplémentaire de 0,03 dollar par rapport à l'utilisation de vingt-huit (28) cœurs.

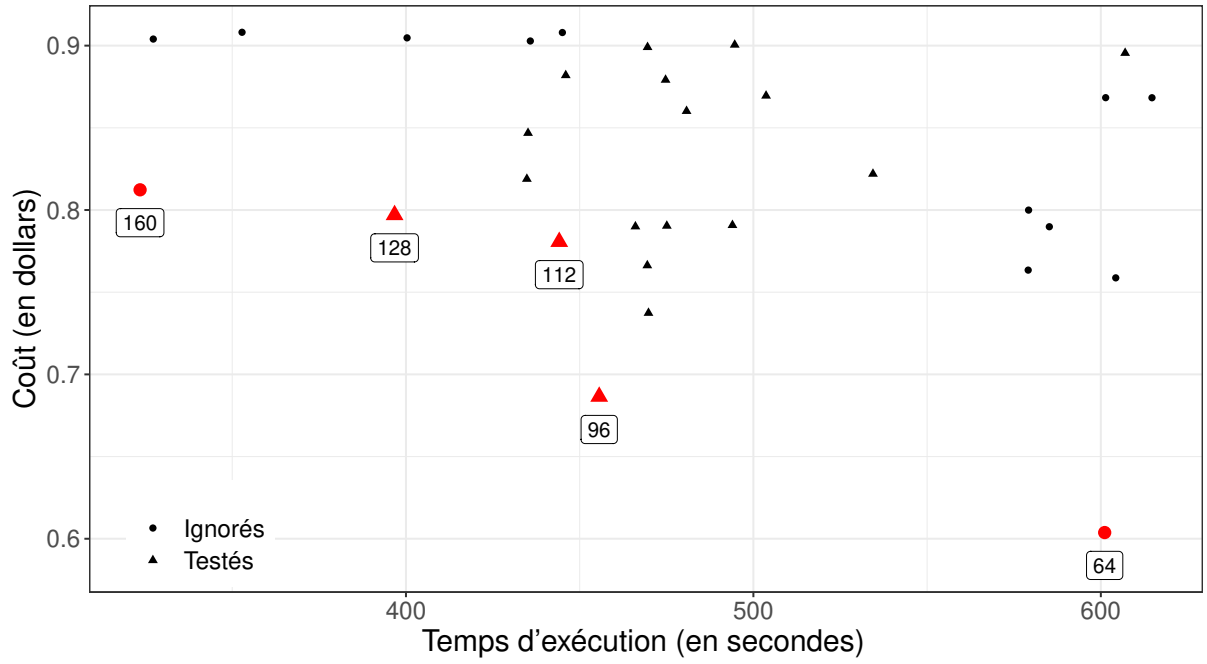



FIGURE IV.7 – Configuration de la plateforme avec le workflow CyberShake.

Le tableau IV.2 résume le nombre de simulations et le temps nécessaire pour renvoyer un ensemble de configurations à l'utilisateur. Ce temps est décomposé en trois parties : (i) Init correspond à la détermination de $C_{max}$, $E_{C_{max}}$ et $C_2$ ; (ii) Dichotomie est le temps passé dans la recherche dichotomique ; et (iii) Exhaustive résume la durée des essais dans la recherche exhaustive.

La décomposition de ce temps diffère d'un workflow à l'autre, mais l'approche proposée est en mesure de déterminer un ensemble de configurations adaptées à tous les workflows scientifiques en moins de cinq minutes. La recherche dichotomique du workflow scientifique Montage se termine par la recherche de deux configurations candidates ayant respectivement trente-deux (32) et trente-quatre (34) cœurs. Il n'y a donc pas besoin d'une recherche exhaustive. La recherche dichotomique pour Epigenomics s'arrête après deux

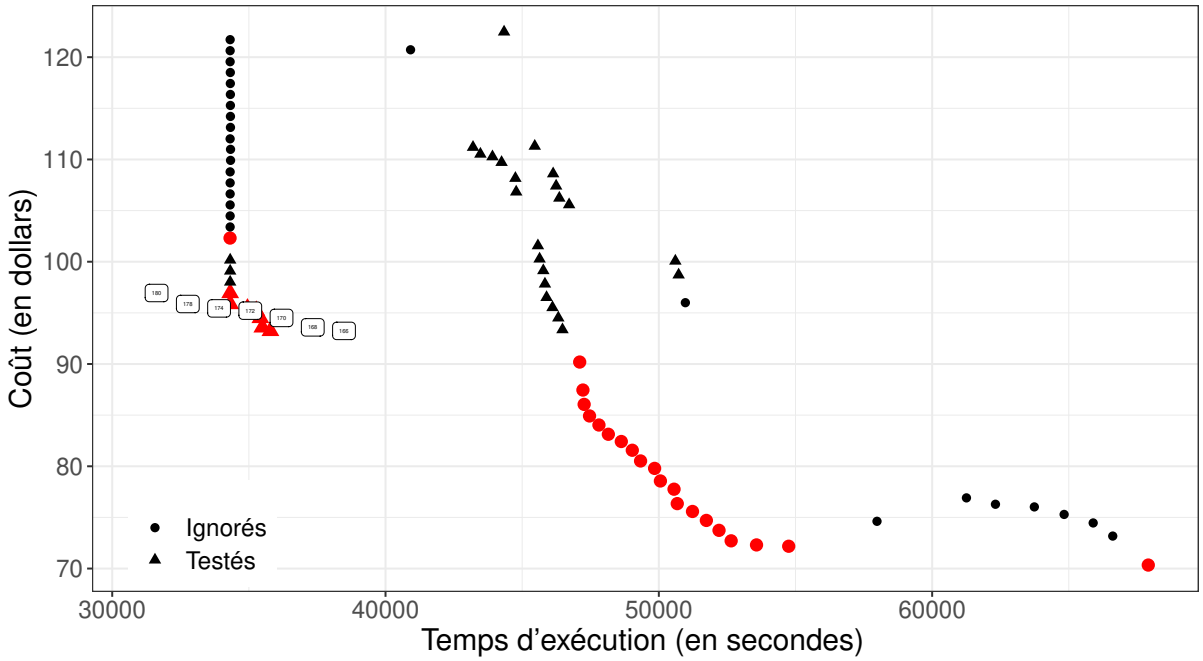



FIGURE IV.8 – Configuration de la plateforme avec le workflow Epigenomics.

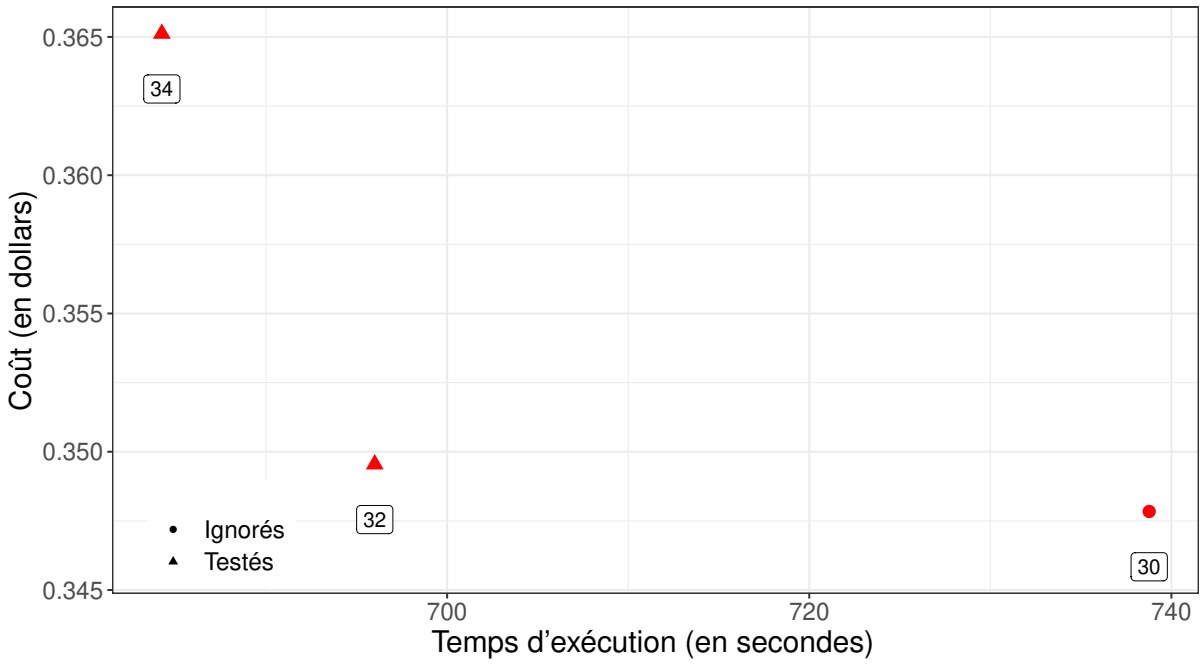



FIGURE IV.9 – Configuration de la plateforme avec le workflow Montage.

itérations mais laisse un grand espace de recherche à couvrir (de cent vingt-quatre (124) à cent quatre-vingt-six (186) cœurs) faisant de la recherche exhaustive la partie la plus longue de l'approche proposée. Cependant, cette approche est composée de plusieurs simulations indépendantes qui peuvent être lancées en parallèle pour réduire le temps de résolution.

Tableau IV.2 – Nombre de simulations et temps nécessaire pour renvoyer un ensemble de configurations d'instance de machine virtuelle à l'utilisateur.

| Workflow | # runs | Init | Dichotomie | Exhaustive | Total |
|---|---|---|---|---|---|
| CyberShake | 24 | 18,4s | 74,3s | 121s | 213,7s |
| Epigenomics | 35 | 35,1s | 8,5s | 142s | 185,6s |
| Montage | 12 | 106s | 143,2s | 0s | 249,2s |

Cette évaluation se conclut en mesurant, pour chacune des configurations sélectionnées par l'approche présentée dans la section IV.3, le gain en termes de temps d'exécution et de volume des données transférées sur le réseau par rapport aux configurations faites du même ensemble d'instances de machines virtuelles mais où tous les fichiers sont stockés sur le service EBS. Les résultats de cette comparaison figure dans le Tableau IV.3.

Tableau IV.3 – Réduction du volume de données transférées et du temps d'exécution en utilisant le service EBS partagé.

| | | Réduction | |
|---|---|---|---|
| Workflow | # cœurs | Données transférées | Temps d'exécution |
| CyberShake | 96 | 62,35% | 10,79% |
| | 112 | 53,31% | 7,45% |
| | 128 | 45,99% | 8,68% |
| Epigenomics | 166 | 0.91% | 0% |
| | 168 | 0,91% | 0% |
| | 170 | 0,91% | 0% |
| | 172 | 0,91% | 0% |
| | 174 | 0,91% | 0% |
| | 178 | 0,91% | 0% |
| | 180 | 0,91% | 0% |
| Montage | 32 | 96,22% | 0,60% |
| | 34 | 89,02% | 0,04% |

L'ordonnancement basé sur la localisation des données, permet une réduction significative du volume de données transférées sur le réseau en exploitant le stockage rapide offert par les instances M5d pour les applications CyberShake et Montage. Le mouvement des données est divisé pratiquement par un facteur deux pour CyberShake et quatre pour Montage. Le gain est cependant très limité pour Epigenomics car la plupart des données impliquées correspondent aux fichiers d'entrée du workflow. Une partie de cette réduction des transferts de données a un impact sur le temps d'exécution de CyberShake qui produit plus de fichiers intermédiaires tandis que le temps d'exécution de Montage est dominé par le temps de traitement des tâches.

Les ressources du cloud IaaS permettent aux scientifiques d'exécuter leurs workflows

scientifiques à forte intensité de données sur des infrastructures personnalisées qui correspondent aux besoins informatiques et de stockage de ces applications. La détermination de l'ensemble des instances de machine virtuelle qui doivent composer ces infrastructures est une tâche complexe, généralement déléguée aux systèmes de gestion de workflows. Une clé de la performance est de pouvoir exploiter les caractéristiques des instances de machines virtuelles.

## IV.6 Conclusion partielle

Dans ce chapitre, nous avons proposé une approche basée sur la simulation en utilisant les résultats du front de Pareto pour sélectionner un ensemble d'instances de machines virtuelles en résolvant un problème d'optimisation coût-performance. Nous avons évalué les performances des algorithmes proposés sur trois workflows scientifiques populaires avec des caractéristiques différentes. Les résultats expérimentaux présentés ont montré que notre approche est capable de renvoyer un ensemble étroit de configurations à un utilisateur dans un délai raisonnable. Nous avons également montré que pour ces configurations nous avons pu réduire significativement le volume de données transférées sur le réseau, ce qui, pour l'un des workflows considérés, se traduit par une réduction du temps d'exécution de 7 à 10% par rapport au temps obtenu si on considère que le service partagé *Elastic Block Storage* pour stocker les données intermédiaires.

# Conclusion générale

Ce travail est une contribution à l'ordonnancement de workflows scientifiques sur les ressources IaaS du cloud. Nous nous sommes concentrés sur les applications parallèles à forte intensité de données. L'analyse des différents ressources de calcul et leur utilisation dans le chapitre I, nous a révélé que :

1. le service du cloud computing le plus adapté pour exécuter une application parallèle est le service IaaS (*Infrastructure as a Service*). Avec ce service l'utilisateur à un accès quasi illimité à un pool de ressources, telles que les ressources de calcul, de stockage, de réseau, etc. L'utilisateur ne se soucie guère du matériel physique sous-jacent, et ne sera facturé que pour les ressources utilisées ;
2. les applications parallèles qui génèrent et utilisent beaucoup de données pendant leur exécution sont modélisées sous forme de workflows scientifiques et représentées par un graphe orienté acyclique (DAG : *Directed Acyclic graph*). Les tâches du workflow scientifique sont liées par des dépendances de données à ne pas négliger dans le processus d'ordonnancement.

La littérature révèle que les méthodes heuristiques, plus précisément l'approche du *list scheduling*, est la mieux adaptée à l'ordonnancement de workflows scientifiques. Ainsi, le chapitre II a été consacré à une revue bibliographique des algorithmes d'ordonnancement sur les ressources IaaS du cloud. Dans cette littérature, nous distinguons deux méthodes, à savoir les méta-heuristiques et les heuristiques. Dans la première méthode, il existe plusieurs approches, entre autres les algorithmes génétiques, l'optimisation de colonie de fourmi, l'essaim de particule, etc. Quant à la deuxième méthode, nous remarquons qu'il existe deux approches qui sont l'approche de *clustering* et l'approche du *list scheduling*.

Dans nos propositions au chapitre III, nous avons fondé notre étude sur l'approche du *list scheduling*. Premièrement, nous avons montré l'intérêt d'utiliser des machines virtuelles à plusieurs vCPUs. Nos expériences montrent qu'en augmentant le nombre de vCPU par machine virtuelles nous avons des gains sur le makespan compte tenu des données qu'on évite

de transférer sur le réseau. Deuxièmement, nous avons proposé des algorithmes d'ordonnancement dont l'objectif est de minimiser le makespan à travers la réduction des données transférées sur le réseau, tout en exploitant des ressources multi-vCPUs. Le réduction des données transférées est faite en deux étapes, d'une part nous avons proposé un algorithmes d'ordonnancement qui réduit les données provenant des tâches précédentes à une tâche $v_i$ et d'autre part, nous avons proposé un autre algorithme qui réduit également les données allant vers les successeurs d'une tâche $v_i$. Les algorithmes proposés dans cette étude exploite le profil d'utilisation (*usage profile*) des machines virtuelles afin de savoir à quel moment et sur quelle machine mapper une tâche. Le profil d'utilisation des machines virtuelles permet de savoir à tout moment le nombre de vCPU disponible sur chaque machine afin d'y affecter plusieurs pouvant s'exécuter en parallèles.

Le chapitre IV a permis de montrer qu'un algorithme qui minimise le makespan sur les ressources IaaS du cloud peut permettre de trouver un ensemble de configuration, c'est-à-dire des plateformes d'exécution qui respecte le compromis entre le makespan et le coût d'exploitation des ressources cloud. Cette approche est fondée d'une part sur la méthode de Pareto en utilisant la notion de dominance pour trouver l'ensemble des solutions non dominées et de l'autre, sur une recherche dichotomique qui retourne un ensemble restreint de solution par rapport aux solutions Pareto.

# Perspectives

Les diverses expérimentations réalisées aux chapitres III et IV ont permis de confirmer l'originalité, les performances et les avantages de nos méthodes. Afin d'améliorer les performances de nos méthodes quelques perspectives peuvent être envisagées :

1. Une première idée consiste a évaluer la tolérance aux fautes de nos algorithmes. Lors de la phase d'exécution des tâches du workflow scientifique, une (ou des) machine(s) virtuelle(s) peut (peuvent) ne pas fonctionner correctement et entraîner des pertes de données qui ont été stockées sur cette (ces) machine(s).
2. Une deuxième idée consiste à voir l'impact de notre algorithme sur la gestion énergétique des machines physiques qui hébergent les machines virtuelles.
3. Une troisième idée est de trouver le bon paramètre pour la recherche dichotomique (fixé à deux dans cette thèse), afin de proposer à l'utilisateur plus de plateformes de compromis.

**Travaux connexes**

Yu et Buyya [157] donnent un bref aperçu des différentes techniques d'ordonnancement de workflow avec des tolérantes aux pannes. La tolérance aux pannes des workflows scientifiques peut être considérée à deux niveaux ; au niveau d'une tâche particulière ou de l'ensemble des tâches du workflow. La réplication de tâches ou de données, la resoumission, le point de contrôle (*Checkpointing*) et la ressource alternative sont des techniques largement utilisées. Poola et al. [158] donnent un aperçu complet des techniques de tolérance aux pannes employées dans divers systèmes de gestion de workflow. Ils présentent également une taxonomie détaillée des différentes techniques utilisées pour la tolérance aux pannes dans les environnements distribués. Les techniques de resoumission sont les plus largement utilisées pour fournir une tolérance aux pannes dans les workflows, suivie de la réplication et du Checkpointing des données [159].

Zhang et al. [160] utilisent la technique pour trouver le plus petit sous-ensemble de ressources pour répliquer les tâches de manière à satisfaire leurs contraintes de dépendances de données. Si le plus petit sous-ensemble de ressources est introuvable, la probabilité de

succès pour toutes les combinaisons de ressources est calculée et les tâches sont répliquées sur l'ensemble de ressources avec la probabilité de succès la plus élevée. La méthode proposée par [161] n'utilise pas le Checkpointing, mais soumet à nouveau une tâche lorsque toutes ses répliques ont échoué. La nouvelle soumission de l'ensemble de la tâche augmente considérablement le temps d'exécution de la tâche, ce qui, à son tour, augmente le makespan du workflow.

# Références


[1] Sabri Pllana, Thomas Fahringer, Johannes Testori, Siegfried Benkner, and Ivona Brandic. Towards an uml based graphical representation of grid workflow applications. In *European Across Grids Conference*, pages 149–158. Springer, 2004.

[2] Rubing Duan, Thomas Fahringer, Radu Prodan, Jun Qin, Alex Villazon, and Marek Wieczorek. Real world workflow applications in the askalon grid environment. In *European Grid Conference*, pages 454–463. Springer, 2005.

[3] Saurabh Kumar Garg and Rajkumar Buyya. Networkcloudsim : Modelling parallel applications in cloud simulations. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 105–113. IEEE, 2011.

[4] Christopher JO Baker and Kei-Hoi Cheung. Semantic web : Revolutionizing knowledge discovery in the life sciences. *Springer Science & Business Media*, 2007.

[5] Lavanya Ramakrishnan and Beth Plale. A multi-dimensional classification model for scientific workflow characteristics. In *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science*, pages 1–12, 2010.

[6] Jia Yu and Rajkumar Buyya. A taxonomy of scientific workflow systems for grid computing. *ACM Sigmod Record*, 34(3) :44–49, 2005.

[7] Yong Zhao, Ioan Raicu, and Ian Foster. Scientific workflow systems for 21st century, new bottle or new wine ? In *2008 IEEE Congress on Services-Part I*, pages 467–471. IEEE, 2008.

[8] Louis Zani, Benoît Lacroix, Alexandre Torre, Jean-Claude Vallet, Clément Berthinier, Christophe Bourcier, Nicolas Misiara, François Nunio, Roser Vallcorba, Christian Van Wambeke, bonne françois, Christine Hoa, Slim constants, and Quentin Le Coz. Olympe, a multi-physic platform for fusion magnet design : Development status and first applications. *Cryogenics*, 108 :103086, 2020.

[9] Mueller ML, Ganslandt T, Frankewitsch T, Krieglstein CF, Senninger N, and Prokosch HU. Workflow analysis and evidence-based medicine : towards integration of knowledge-based functions in hospital information systems. *AMIA Annual Symposium Proceedings Archive*, page 330–334, 1999.

[10] Wendy A Warr. Scientific workflow systems : Pipeline pilot and knime. *Journal of computer-aided molecular design*, 26(7) :801–804, 2012.

[11] Feng Wang, Hui Deng, Ling Guo, and Kaifan Ji. A survey on scientific-workflow techniques for e-science in astronomy. In *2010 International Forum on Information Technology and Applications*, volume 1, pages 417–420. IEEE, 2010.

[12] Gideon Juve, Mats Rynge, Ewa Deelman, Jens-S Vöckler, and G Bruce Berriman. Comparing futuregrid, amazon ec2, and open science grid for scientific workflows. *Computing in Science & Engineering*, 15(4) :20–29, 2013.

[13] Xiu Li, Jingdong Song, and Biqing Huang. A scientific workflow management system architecture and its scheduling based on cloud service platform for manufacturing big data analytics. *The International Journal of Advanced Manufacturing Technology*, 84(1-4) :119–131, 2016.

[14] Anthony M Middleton, David Alan Bayliss, Gavin Halliday, Arjuna Chala, and Borko Furht. The hpcc/ecl platform for big data. In *Big Data Technologies and Applications*, pages 159–183. Springer, 2016.

[15] Ewa Deelman and Ann Chervenak. Data management challenges of data-intensive scientific workflows. In *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pages 687–692. IEEE, 2008.

[16] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4) :457–493, 2015.

[17] Jinghui Zhang, Mingjun Wang, Junzhou Luo, Fang Dong, and Junxue Zhang. Towards optimized scheduling for data-intensive scientific workflow in multiple datacenter environment. *Concurrency and Computation : Practice and Experience*, 27(18) :5606–5622, 2015.

[18] Reagan Moore, Chaitanya Baru, Richard Marciano, Arcot Rajasekar, and Michael Wan. Data-intensive computing. *The Grid : Blueprint for a New Computing Infrastructure, Morgan Kaufmann*, pages 105–129, 1999.

[19] Zacharia Fadika, Madhusudhan Govindaraju, Richard Canon, and Lavanya Ramakrishnan. Evaluating hadoop for data-intensive scientific operations. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 67–74. IEEE, 2012.

[20] Mohammad Alkhalaileh, Rodrigo N Calheiros, Quang Vinh Nguyen, and Bahman Javadi. Data-intensive application scheduling on mobile edge cloud computing. *Journal of Network and Computer Applications*, 167 :102735, 2020.

[21] Xiaozhong Geng, Yingshuang Mao, Mingyuan Xiong, and Yang Liu. An improved task scheduling algorithm for scientific workflow in cloud computing environment. *Cluster Computing*, 22(3) :7539–7548, 2019.

[22] Simon Ostermann, Alexandria Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. A performance analysis of ec2 cloud computing services for scientific computing. *International Conference on Cloud Computing, Springer*, 2009.

[23] Christina N Hoefer and Georgios Karagiannis. Taxonomy of cloud computing services. In *2010 IEEE Globecom Workshops*, pages 1345–1350. IEEE, 2010.

[24] Maria Alejandra Rodriguez and Rajkumar Buyya. A taxonomy and survey on scheduling algorithms for scientific workflows in iaas cloud computing environments. *Concurrency and Computation : Practice and Experience*, 29(8) :e4041, 2017.

[25] Jiagang Liu, Ju Ren, Wei Dai, Deyu Zhang, Pude Zhou, Yaoxue Zhang, Geyong Min, and Noushin Najjari. Online multi-workflow scheduling under uncertain task execution time in iaas clouds. *IEEE Transactions on Cloud Computing*, 2019.

[26] Mengxia Zhu, Qishi Wu, and Yang Zhao. A cost-effective scheduling algorithm for scientific workflows in clouds. In *2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC)*, pages 256–265. IEEE, 2012.

[27] S. Jaya Nirmala and S. Mary Saira Bhanu. Catfish-pso based scheduling of scientific workflows in iaas cloud. *Computing 98*, page 1091–1109, 2016.

[28] Wang Jing Wang Qiang, Li Xiongfei. A data placement and task scheduling algorithm in cloud computing. *Journal of Computer Research and Development*, 51(11) :2416, 2014.

[29] Robabeh Ghafouri, Ali Movaghar, and Mehran Mohsenzadeh. A budget constrained scheduling algorithm for executing workflow application in infrastructure as a service clouds. *Peer-to-Peer Networking and Applications*, 12 :241–268, 2018.

[30] Jin Sun, Lu Yin, Minhui Zou, Yi Zhang, Tianqi Zhang, and Junlong Zhou. Makespan-minimization workflow scheduling for complex networks with social groups in edge computing. *Journal of Systems Architecture*, 2020.

[31] Sen Su, Jian Li, Qingjia Huang, Xiao Huang, Kai Shuang, and Jie Wang. Cost-efficient task scheduling for executing large programs in the cloud. *Parallel Computing*, 39(4-5) :177–188, 2013.

[32] Vishakha Singh, Indrajeet Gupta, and Prasanta K. Jana. An energy efficient algorithm for workflow scheduling in iaas cloud. *Journal of Grid Computing*, 18 :357–376, 2020.

[33] Xiping Liu, Wanchun Dou, Jinjun Chen, Shaokun Fan, Shing-Chi Cheung, and Shijie Cai. On design, verification, and dynamic modification of the problem-based scientific workflow model. *Simulation Modelling Practice and Theory*, 15(9) :1068–1088, 2007.

[34] Ali Asghari, Mohammad Karim Sohrabi, and Farzin Yaghmaee. Task scheduling, resource provisioning, and load balancing on scientific workflows using parallel sarsa reinforcement learning agents and genetic algorithm. *The Journal of Supercomputing*, pages 1–29, 2020.

[35] Xin Ye, Jiwei Liang, Sihao Liu, and Jia Li. A survey on scheduling workflows in cloud environment. In *2015 International Conference on Network and Information Systems for Computers*, pages 344–348. IEEE, 2015.

[36] Sudhir Shenai et al. Survey on scheduling issues in cloud computing. *Procedia Engineering*, 38 :2881–2888, 2012.

[37] Haiyang Hu, Zhongjin Li, Hua Hu, Jie Chen, Jidong Ge, Chuanyi Li, and Victor Chang. Multi-objective scheduling for scientific workflow in multicloud environment. *Journal of Network and Computer Applications*, 114 :108–122, 2018.

[38] Ehab Nabiel Alkhanak, Sai Peck Lee, and Saif Ur Rehman Khan. Cost-aware challenges for workflow scheduling approaches in cloud computing environments : Taxonomy and opportunities. *Future Generation Computer Systems*, 50 :3–21, 2015.

[39] Vahid Arabnejad, Kris Bubendorfer, and Bryan Ng. Budget and deadline aware e-science workflow scheduling in clouds. *IEEE Transactions on Parallel and Distributed systems*, 30(1) :29–44, 2018.

[40] Jieun Choi, Theodora Adufu, and Yoonhee Kim. Data-locality aware scientific workflow scheduling methods in hpc cloud environments. *International Journal of Parallel Programming*, 45(5) :1128–1141, 2017.

[41] Edgar Gabriel, Graham E Fagg, George Bosilca, Thara Angskun, Jack J Dongarra, Jeffrey M Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Castain Ralph H., Daniel David J., Graham Richard L., and Woodall Timothy S. Open mpi : Goals, concept, and design of a next generation mpi implementation. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*, pages 97–104. Springer, 2004.

[42] Torsten Hoefler, James Dinan, Darius Buntinas, Pavan Balaji, Brian Barrett, Ron Brightwell, William Gropp, Vivek Kale, and Rajeev Thakur. Mpi+ mpi : a new hybrid approach to parallel programming with mpi plus shared memory. *Computing*, 95(12) :1121–1136, 2013.

[43] Rolf Rabenseifner, Georg Hager, and Gabriele Jost. Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. In *2009 17th Euromicro international conference on parallel, distributed and network-based processing*, pages 427–436. IEEE, 2009.

[44] Jidong Zhai, Tianwei Sheng, Jiangzhou He, Wenguang Chen, and Weiming Zheng. Efficiently acquiring communication traces for large-scale parallel applications. *IEEE Transactions on Parallel and Distributed Systems*, 22(11) :1862–1870, 2011.

[45] Amnon Barak and Oren La'adan. The mosix multicomputer operating system for high performance cluster computing. *Future Generation Computer Systems*, 13(4-5) :361–372, 1998.

[46] Yair Amir, Baruch Awerbuch, Amnon Barak, R Sean Borgstrom, and Arie Keren. An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE Transactions on parallel and distributed Systems*, 11(7) :760–768, 2000.

[47] Xingfu Wu and Valerie Taylor. Performance modeling of hybrid mpi/openmp scientific applications on large-scale multicore cluster systems. In *2011 14th IEEE International Conference on Computational Science and Engineering*, pages 181–190. IEEE, 2011.

[48] Top500. www.top500.org/lists/top500/2020/06, visité le 26 juin 2020.

[49] Tomasz Haupt and Marlon E Pierce. Distributed object-based grid computing environments. *Grid Computing : Making the Global Infrastructure a Reality. New York : Wiley˜ & Sons*, pages 713–728, 2003.

[50] V Rajaraman. Cloud computing. *Resonance*, 19(3) :242–258, 2014.

[51] Amazon EC2. www.aws.amazon.com/fr/ec2/instance-types, visité le 11 avril 2019.

[52] Compute Engine Google. Google instance compute engine, 2019.

[53] Microsoft Azure. www.azure.microsoft.com/fr-fr/pricing/details/virtual-machines/series, visité le 27 avril 2019.

[54] IBM-CLOUD. Ibm cloud services, visité le 02 juillet 2020. www.ibm.com/cloud, July 2020.

[55] Setrag Khoshafian and Marek Buckiewicz. *Groupware et workflow*. 1998.

[56] WfMC. Workflow Manager Coalition, www.wfmc.org/resources, visité le 07 mai 2019.

[57] El-Sayed T El-kenawy, Ali Ibraheem El-Desoky, and Mohamed F Al-rahamawy. Extended max-min scheduling using petri net and load balancing. *Int. J. Soft Comput. Eng.(IJSCE)*, 2(4) :198–203, 2012.

[58] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3) :682–692, 2013.

[59] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation : Practice and Experience*, 18(10) :1039–1065, 2006.

[60] Tom Oinn, Mark Greenwood, Matthew Addis, M Nedim Alpdemir, Justin Ferris, Kevin Glover, Carole Goble, Antoon Goderis, Duncan Hull, Darren Marvin, LI Peter, LORD Phillip, POCOCK Mattew R., Martin SENGER, STEVENS Robert, WIPAT Anil, and Chris WROE. Taverna : lessons in creating a workflow environment for the life sciences. *Concurrency and Computation : Practice and Experience*, 18(10) :1067–1100, 2006.

[61] Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. The triana workflow environment : Architecture and applications. In *Workflows for e-Science*, pages 320–339. Springer, 2007.

[62] Thomas Fahringer, Alexandru Jugravu, Sabri Pllana, Radu Prodan, Clovis Seragiotto Jr, and Hong-Linh Truong. Askalon : a tool set for cluster and grid computing. *Concurrency and Computation : Practice and Experience*, 17(2-4) :143–169, 2005.

[63] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, John Good, et al. Pegasus : A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3) :219–237, 2005.

[64] Spyridon V Gogouvitis, Kleopatra G Konstanteli, Dimosthenis Kyriazis, Gregory Katsaros, Tommaso Cucinotta, and Michael Boniface. Workflow management systems in distributed environments. In *Achieving Real-Time in Distributed Computing : From Grids to Clouds*, pages 115–132. IGI Global, 2012.

[65] Gregor Von Laszewski, Mihael Hategan, and Deepti Kodeboyina. Java cog kit workflow. In *Workflows for e-Science*, pages 340–356. Springer, 2007.

[66] Rick Wagner, Philip Papadopoulos, Dmitry Mishin, Trevor Cooper, Mahidhar Tatineti, Gregor von Laszewski, Fugang Wang, and Geoffrey C Fox. User managed virtual clusters in comet. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, page 24. ACM, 2016.

[67] GJ Toomer. Ptolemys almagest. edited by gj toomer, translated by gj toomer, 1984.

[68] Ptolemy project, www.ptolemy.berkeley.edu/ptolemyclassic/index.htm, visité le 04 septembre 2019.

[69] P. Blacha, K. Schwarz, and G.K.H. Madsen. Wien2k, an augmented plane wave plus local orbitals program for calculations crystal properties, 2001.

[70] P Blaha. Wien2k, an augmented plane wave plus local orbitals program for calculating crystal properties (karlheinz schwarz, techische universitat wien, austria, 2001). 36 rfw bader. *Atoms in Molecules—A Quantum Theory*, 1990.

[71] JW Sattlegger, J Rohde, H Egbers, GP Dohr, PK Stiller, and JA Echterhoff. Inmod—two dimensional inverse modeling algorithm based on ray theory. *Geophysical Prospecting*, 29(2) :229–240, 1981.

[72] G Kousalya, P Balakrishnan, and C Pethuru Raj. Workflow management systems. In *Automated Workflow Scheduling in Self-Adaptive Clouds*, pages 55–64. Springer, 2017.

[73] Gian Luigi Granato, Cinthia Ragone-Figueroa, Rosa Domínguez-Tenreiro, Aura Obreja, Stefano Borgani, Gabriella De Lucia, and Giuseppe Murante. The early phases of galaxy clusters formation in ir : coupling hydrodynamical simulations with grasil-3d. *Monthly Notices of the Royal Astronomical Society*, 450(2) :1320–1332, 2015.

[74] NIST. National institute of standards and technology, (https ://www.france-science.org/le-nist-livre-enfin-sa-definition.html), October 2019.

[75] Rajkumar Buyya, James Broberg, and Andrzej M Goscinski. *Cloud computing : Principles and paradigms*, volume 87. John Wiley & Sons, 2010.

[76] Jeffrey Chang and Mark Johnston. Approaches to cloud computing in the public sector : Case studies in uk local government. In *Cloud Security : Concepts, Methodologies, Tools, and Applications*, pages 188–209. IGI Global, 2019.

[77] Neal Leavitt. Hybrid clouds move to the forefront. *Computer*, (5) :15–18, 2013.

[78] Greg Goth. Virtualization : Old technology offers huge new potential. *IEEE Distributed Systems Online*, 8(2) :3–3, 2007.

[79] Ying-Chuan Chen, Shuen-Tai Wang, Hsi-Ya Chang, Te-Ming Chen, and Chin-Hung Li. The performance analysis for virtualisation cluster and cloud platforms. *International Journal of Computational Science and Engineering*, 6(4) :255–263, 2011.

[80] Joseph Esensten. Security strengths weaknesses of virtualization as a green computing solution. 2011.

[81] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *USENIX Annual Technical Conference, General Track*, pages 1–14, 2001.

[82] Daniel Bartholomew. Qemu : a multihost, multitarget emulator. *Linux Journal*, 2006(145) :3, 2006.

[83] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *ACM SIGOPS operating systems review*, volume 37, pages 164–177. ACM, 2003.

[84] Yin Zhang and Min Chen. Cloud platform for networking. In *Cloud Based 5G Wireless Networks*, pages 21–31. Springer, 2016.

[85] Hasan Fayyad-Kazan, Luc Perneel, and Martin Timmerman. Benchmarking the performance of microsoft hyper-v server, vmware esxi and xen hypervisors. *Journal of Emerging Trends in Computing and Information Sciences*, 4(12) :922–933, 2013.

[86] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. A taxonomy and survey of cloud computing systems. In *2009 Fifth International Joint Conference on INC, IMS and IDC*, pages 44–51. Ieee, 2009.

[87] Rackspace. www.rackspace.com/cloud, visité le 27 décembre 2019.

[88] GoGrid. www.gogrid.com/products/cloud-hosting, visité le 02 novembre 2019.

[89] Eucalyptus. www.eucalyptus.com/eucalyptus-cloud, visité le 03 octobre 2019.

[90] Nimbus-Project. www.nimbusproject.org, visité le 25 novembre 2019.

[91] Open-Nebula. www.opennebula.org, visité le 02 novembre 2019.

[92] Open-Stack. www.openstack.org, visté le 25 décembre 2019.

[93] Gurmeet Singh, Carl Kesselman, and Ewa Deelman. Performance impact of resource provisioning on workflows applying. *Technical, University of Southern California*, 2005.

[94] Gideon Juve and Ewa Deelman. Resource provisioning options for large-scale scientific workflows. In *2008 IEEE Fourth International Conference on eScience*, pages 608–613. IEEE, 2008.

[95] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of Scheduling under Precedence Constraints. *Operations Research*, 26(1) :22–35, February 1978.

[96] Karan Vahi, Mats Rynge, Gideon Juve, Rajiv Mayani, and Ewa Deelman. Rethinking data management for big data scientific workflows. In *2013 IEEE International Conference on Big Data*, pages 27–35. IEEE, 2013.

[97] Tansel Dokeroglu, Ender Sevinc, Tayfun Kucukyilmaz, and Ahmet Cosar. A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering*, 137 :106040, 2019.

[98] Claudia Szabo and Trent Kroeger. Evolving multi-objective strategies for task allocation of scientific workflows on public clouds. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012.

[99] A Verma and S Kaushal. Budget constrained priority based genetic algorithm for workflow scheduling in cloud. *Computing & Communication*, 2013.

[100] Lovejit Singh and Sarbjeet Singh. A genetic algorithm for scheduling workflow applications in unreliable cloud environment. In *International Conference on Security in Computer Networks and Distributed Systems*, pages 139–150. Springer, 2014.

[101] A. Verma and S. Kaushal. Deadline constraint heuristic-based genetic algorithm for workflow scheduling in cloud. *Int. J. Grid Util. Comput. 5*, 2014.

[102] J Meena, M. Kumar, and M. Vardhan. Cost effective genetic algorithm for workflow scheduling in cloud under deadline constraint. *IEEE Access 4*, 2016.

[103] Z. Zhu, G. Zhang, M. Li, and X. Liu. Evolutionary multi-objective workflow scheduling in cloud. *IEEE Trans. Parallel Distrib. Syst.*, 27 :1344–1357, 2016.

[104] Kun Li, Gaochao Xu, Guangyu Zhao, Yushuang Dong, and Dan Wang. Cloud task scheduling based on load balancing ant colony optimization. In *2011 sixth annual ChinaGrid conference*, pages 3–9. IEEE, 2011.

[105] Xin Lu and Zilong Gu. A load-adapative cloud resource scheduling model based on ant colony algorithm. In *2011 IEEE international conference on cloud computing and intelligence systems*, pages 296–300. IEEE, 2011.

[106] Kumar Nishant, Pratik Sharma, Vishal Krishna, Chhavi Gupta, Kuwar Pratap Singh, Ravi Rastogi, et al. Load balancing of nodes in cloud using ant colony optimization. In *2012 UKSim 14th international conference on computer modelling and simulation*, pages 3–8. IEEE, 2012.

[107] Priyanka Mod and Mayank Bhatt. Aco based dynamic resource scheduling for improving cloud performance. *International Journal of Science, Engineering and Technology Research (IJSETR)*, 3(11) :3012–3017, 2014.

[108] Virendra Singh Kushwah and Sandip Kumar Goyal. A basic simulation of aco algorithm under cloud computing for fault tolerant. In *Proceedings of the International Conference on Data Engineering and Communication Technology*, pages 465–472. Springer, 2017.

[109] M. A. Rodriguez and R. Buyya. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. *IEEE Trans. Cloud Comput. 2*, 2014.

[110] Amandeep Verma and Sakshi Kaushal. A hybrid multi-objective particle swarm optimization for scientific workflow scheduling. *Parallel Computing*, 62 :1–19, 2017.

[111] Navneet Kaur and Sarbjeet Singh. A budget-constrained time and reliability optimization bat algorithm for scheduling workflow applications in clouds. *Procedia Computer Science*, 98 :199–204, 2016.

[112] WikiPédia. https ://fr.wikipedia.org/wiki/Métaheuristique, visité le 28 mars 2020.

[113] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3) :260–274, March 2002.

[114] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm : Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2) :182–197, 2002.

[115] Gobalakrishnan Natesan and Arun Chokkalingam. Multi-objective task scheduling using hybrid whale genetic optimization algorithm in heterogeneous computing environment. *Wireless Personal Communications*, 110(4) :1887–1913, 2020.

[116] Zehua Zhang and Xuejie Zhang. A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation. In *2010 The 2nd International Conference on Industrial Mechatronics and Automation*, volume 2, pages 240–243. IEEE, 2010.

[117] Rodrigo N Calheiros and Rajkumar Buyya. Meeting deadlines of scientific workflows in public clouds with tasks replication. *IEEE Transactions on Parallel and Distributed Systems*, 25(7) :1787–1796, 2013.

[118] Suraj Pandey, Linlin Wu, Siddeswara Mayura Guru, and Rajkumar Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 400–407. IEEE, 2010.

[119] Nuttapong Netjinda, Booncharoen Sirinaovakul, and Tiranee Achalakul. Cost optimization in cloud provisioning using particle swarm optimization. In *2012 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, pages 1–4. IEEE, 2012.

[120] Lizheng Guo, Guojin Shao, and Shuguang Zhao. Multi-objective task assignment in cloud computing by particle swarm optimization. In *2012 8th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–4. IEEE, 2012.

[121] Juan J Durillo and Radu Prodan. Multi-objective workflow scheduling in amazon ec2. *Cluster computing*, 17(2) :169–189, 2014.

[122] Larousse. www.larousse.fr/dictionnaires/francais/heuristique/39848, visité le 05 avril 2020.

[123] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick H.J. Epema. Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds. *Future Generation Computer Systems*, 29(1) :158–169, January 2013.

[124] Bing Lin, Wenzhong Guo, Guolong Chen, Naixue Xiong, and Rongrong Li. Cost-driven scheduling for deadline-constrained workflow on multi-clouds. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 1191–1198. IEEE, 2015.

[125] Vahid Arabnejad, Kris Bubendorfer, and Bryan Ng. Scheduling deadline constrained scientific workflows on dynamically provisioned cloud resources. *Future Generation Computer Systems*, 75 :348–364, 2017.

[126] V. Arabnejad, K. Bubendorfer, B. Ng, and K. Chard. A Deadline Constrained Critical Path Heuristic for Cost-Effectively Scheduling Workflows. In *Proc. IEEE/ACM 8th Int. Conf. Utility and Cloud Computing (UCC)*, pages 242–250, December 2015.

[127] Katarzyna Keahey, Mauricio Tsugawa, Andrea Matsunaga, and Jose Fortes. Sky computing. *IEEE Internet Computing*, 13(5) :43–51, 2009.

[128] B. Lin, W. Guo, N. Xiong, G. Chen, A. V. Vasilakos, and H. Zhang. A Pretreatment Workflow Scheduling Approach for Big Data Applications in Multicloud Environments. *IEEE Transactions on Network and Service Management*, 13(3) :581–594, September 2016.

[129] Khaled Almi'Ani and Young Choon Lee. Partitioning-based workflow scheduling in clouds. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 645–652. IEEE, 2016.

[130] Cui Lin and Shiyong Lu. Scheduling scientific workflows elastically for cloud computing. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 746–747. IEEE, 2011.

[131] Jyoti Thaman and Manpreet Singh. Green cloud environment by using robust planning algorithm. *Egyptian Informatics Journal*, 18(3) :205–214, November 2017.

[132] Bhaskar Prasad Rimal and Martin Maier. Workflow Scheduling in Multi-Tenant Cloud Computing Environments. *IEEE Transactions on Parallel and Distributed Systems*, 28(1) :290–304, January 2017.

[133] Khaled Almi'ani, Young Choon Lee, and Bernard Mans. On efficient resource use for scientific workflows in clouds. *Computer Networks*, 146 :232–242, December 2018.

[134] Ehab Nabiel Alkhanak and Sai Peck Lee. A hyper-heuristic cost optimisation approach for Scientific Workflow Scheduling in cloud computing. *Future Generation Computer Systems*, 86 :480–506, September 2018.

[135] Daniel de Oliveira, Eduardo Ogasawara, Kary Ocaña, Fernanda Baião, and Marta Mattoso. An adaptive parallel execution strategy for cloud-based scientific workflows. *Concurrency and Computation : Practice and Experience*, 24(13) :1531–1550, 2012.

[136] Hamid Mohammadi Fard, Radu Prodan, and Thomas Fahringer. Multi-objective list scheduling of workflow applications in distributed computing infrastructures. *Journal of Parallel and Distributed Computing*, 74(3) :2152–2165, 2014.

[137] M. A. Rodriguez and R. Buyya. A Responsive Knapsack-Based Algorithm for Resource Provisioning and Scheduling of Scientific Workflows in Clouds. In *Proc. 44th Int Parallel Processing (ICPP) Conf*, pages 839–848, September 2015.

[138] Xiumin Zhou, Gongxuan Zhang, Jin Sun, Junlong Zhou, Tongquan Wei, and Shiyan Hu. Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT. *Future Generation Computer Systems*, 93 :278–289, April 2019.

[139] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick H.J Epema. Cost-Driven Scheduling of Grid Workflows Using Partial Critical Paths. *IEEE Transactions on Parallel and Distributed Systems*, 23(8) :1400–1414, August 2012.

[140] A. Rezaeian, H. Abrishami, S. Abrishami, and M. Naghibzadeh. A Budget Constrained Scheduling Algorithm for Hybrid Cloud Computing Systems Under Data Privacy. In *Proc. IEEE Int. Conf. Cloud Engineering (IC2E)*, pages 230–231, April 2016.

[141] A. C. Zhou, B. He, and C. Liu. Monetary cost optimizations for hosting workflow-as-a-service in iaas clouds,. *IEEE Trans. Cloud Comput. 4*, page 34–48, 2016.

[142] James Blythe, Sonal Jain, Ewa Deelman, Yolanda Gil, Karan Vahi, Anirban Mandal, and Ken Kennedy. Task scheduling strategies for workflow-based applications in grids. In *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.*, volume 2, pages 759–767. IEEE, 2005.

[143] Thomas Stützle and Holger H Hoos. Max–min ant system. *Future generation computer systems*, 16(8) :889–914, 2000.

[144] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-science : An overview of workflow system features and capabilities. *Future generation computer systems*, 25(5) :528–540, 2009.

[145] Adam Barker and Jano Van Hemert. Scientific workflow : a survey and research directions. In *International Conference on Parallel Processing and Applied Mathematics*, pages 746–753. Springer, 2007.

[146] Scott Callaghan, Philip Maechling, Ewa Deelman, Karan Vahi, Gaurang Mehta, Gideon Juve, Kevin Milner, Robert Graves, Edward Field, David Okaya, et al. Reducing time-to-solution using distributed high-throughput mega-workflows-experiences from scec cybershake. In *2008 IEEE Fourth International Conference on eScience*, pages 151–158. IEEE, 2008.

[147] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei-Hui Su, and Karan Vahi. Characterization of scientific workflows. In *2008 third workshop on workflows in support of large-scale science*, pages 1–10. IEEE, 2008.

[148] Joseph C Jacob, Daniel S Katz, Thomas Prince, Bruce G Berriman, John C Good, Anastasia C Laity, Ewa Deelman, Gurmeet Singh, and Mei-Hui Su. The montage architecture for grid-enabled science processing of large, distributed datasets. 2004.

[149] Joseph C Jacob, Daniel S Katz, G Bruce Berriman, John C Good, Anastasia Laity, Ewa Deelman, Carl Kesselman, Gurmeet Singh, Mei-Hui Su, Thomas Prince, et al. Montage : a grid portal and software toolkit for science-grade astronomical image mosaicking. *International Journal of Computational Science and Engineering*, 4(2) :73–87, 2009.

[150] Alexander Thomson and Daniel J Abadi. Calvinfs : Consistent {WAN} replication and scalable metadata management for distributed file systems. In *13th {USENIX} Conference on File and Storage Technologies ({FAST} 15)*, pages 1–14, 2015.

[151] Tchimou N'takpé and Frédéric Suter. Don't hurry be happy : A deadline-based backfilling approach. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82. Springer, 2017.

[152] Ahuva W. Mu'alem and Dror G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE transactions on parallel and distributed systems*, 12(6) :529–543, 2001.

[153] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10) :2899–2917, 2014.

[154] SimGrid. Versatile simulation of distributed systems, www.simgrid.org, visité le 08 janvier 2019.

[155] WRENCH. https ://wrench-project.org, visité le 09 janvier 2019.

[156] Henri Casanova, Rafael Ferreira da Silva, Ryan Tanaka, Suraj Pandey, Gautam Jethwani, William Koch, Spencer Albrecht, James Oeth, and Frédéric Suter. Developing accurate and scalable simulators of production workflow management systems with wrench. *Future Generation Computer Systems*, 112 :162–175, 2020.

[157] Jia Yu and Rajkumar Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3-4) :171–200, 2005.

[158] Deepak Poola, Mohsen Amini Salehi, Kotagiri Ramamohanarao, and Rajkumar Buyya. A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments. In *Software architecture for big data and the cloud*, pages 285–320. Elsevier, 2017.

[159] Kassian Plankensteiner, Radu Prodan, Thomas Fahringer, Attila Kertesz, and Peter Kacsuk. Fault-tolerant behavior in state-of-the-art grid workflow management systems. Technical report, CoreGRID, 2007.

[160] Yang Zhang, Anirban Mandal, Charles Koelbel, and Keith Cooper. Combined fault tolerance and scheduling techniques for workflow applications on computational grids. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 244–251. IEEE, 2009.

[161] Kassian Plankensteiner, Radu Prodan, and Thomas Fahringer. A new fault tolerance heuristic for scientific workflows in highly distributed environments based on resubmission impact. In *2009 Fifth IEEE International Conference on e-Science*, pages 313–320. IEEE, 2009.

# Liste des publications de l'auteur

## Communication orale

Tchimou N'Takpé, Jean Edgar Gnimassoun, Souleymane Oumtanaga and Frederic Suter (2019). Data-Aware and Simulation-Driven Planning of Scientific Workflows on IaaS Clouds. WORKS 2019 Workshop on Workflows in Support of Large-Scale Science Sunday, 17 November 2019, Denver, CO.
https ://sc19.supercomputing.org/proceedings/workshops/workshop_pages/pec243.html

## Articles scientifiques

1. Jean Edgard GNIMASSOUN, Tchimou N'TAKPE, Gokou Hervé Fabrice DIEDIE and Souleymane OUMTANAGA, "A Workflow Scheduling Algorithm for Reducing Data Transfers in Cloud IaaS" International Journal of Advanced Computer Science and Applications(IJACSA), 11(5), 2020.
   http ://dx.doi.org/10.14569/IJACSA.2020.0110534
2. Tchimou N'Takpé, Jean Edgar Gnimassoun, Souleymane Oumtanaga and Frederic Suter (2019). Data-Aware and Simulation-Driven Planning of Scientific Workflows on IaaS Clouds. Concurrency and Computation : Practice and Experience (CCPE). **En cours de publication**

# ANNEXES

## A Génération de la plateforme

```
# -*- coding: utf-8 -*-
import os
import sys
from xml.etree import ElementTree
from lxml import etree
from xml.etree.ElementTree import Element, SubElement, tostring, XML

total_cores = int(sys.argv[1])
max_core_by_vm = int(sys.argv[2])

if total_cores % 2 != 0 & max_core_by_vm > 1 :
    total_cores = total_cores + 1

instance_sizes = [96, 64, 48, 32, 16, 8, 4, 2, 1]
instance_bws = [3125, 2500, 1250, 1250, 416.67, 208.33, 104.17, 52.08,
   26.042]
ebs_bws      = [1750, 1250, 875, 625, 437.5, 218.75, 109.375, 54.6875,
   27.34375]

# determine which and how many instances are needed
instance_numbers = [0]*len(instance_sizes)

def compute_instance_numbers(total_cores, index):
  if total_cores > 0:
      while total_cores >= instance_sizes[index]:
          total_cores -= instance_sizes[index]
          instance_numbers[index] += 1
      return compute_instance_numbers(total_cores, index + 1)

compute_instance_numbers(total_cores, instance_sizes.index(max_core_by_vm)
```

```
    )

# check resultand
computed_total_cores = 0
for i in range(len(instance_sizes)):
    computed_total_cores += instance_sizes[i] * instance_numbers[i]
assert(total_cores == computed_total_cores)

# Create the XML tree
platform = Element('platform', version="4.1")
zone = SubElement(platform, 'zone', id="AS0", routing = "Full")

# WMS Host
SubElement(zone, 'host', id = "00000", speed = "1000Gf", core = "1")

# EBS Host
SubElement(zone, 'host', id = "00001", speed = "1000Gf", core = "1")

# Generate Compute Hosts. Start at "00002"
host_id = 2
for i in range(0, len(instance_sizes)):
    for j in range (0, instance_numbers[i]) :
        SubElement(zone, 'host', id = str(host_id).zfill(5), speed = "1000Gf", core = str(instance_sizes[i]))
        host_id += 1

# Create special links: "switch"
SubElement(zone, 'link', id = "switch", bandwidth="25Gbps", latency="100us", sharing_policy="FATPIPE")

# Create WMS and EBS links, bandwidth depends on the largest instance size
largest_instance_index = instance_numbers.index(filter(lambda x : x!=0, instance_numbers)[0])
bw = ebs_bws[largest_instance_index]
SubElement(zone, 'link', id = "00000", bandwidth= str(bw) + "MBps", latency="100us", sharing_policy="FATPIPE")
SubElement(zone, 'link', id = "00001", bandwidth= str(bw) + "MBps", latency="100us", sharing_policy="FATPIPE")

# Create links from VM to other VM or to WMS
link_id = 2
for i in range(0, len(instance_sizes)):
    for j in range (0, instance_numbers[i]) :
        SubElement(zone, 'link', id = str(link_id).zfill(5), bandwidth = str(instance_bws[i]) + 'MBps',
```

```
                    latency = "100us")
        link_id += 1

# Create links from EBS to the VM
for i in range(0, len(instance_sizes)):
    for j in range (0, instance_numbers[i]) :
        SubElement(zone, 'link', id = str(link_id).zfill(5), bandwidth =
    str(ebs_bws[i]) + 'MBps',
                    latency = "100us")
        link_id += 1

# Create the routes
for i in range((host_id - 1)):
    for j in range((i + 1), host_id):
        route = SubElement(zone, 'route', src = str(i).zfill(5), dst = str
    (j).zfill(5))
        if (i != 1):
            SubElement(route, 'link_ctn', id = str(i).zfill(5))
            SubElement(route, 'link_ctn', id = "switch")
            SubElement(route, 'link_ctn', id = str(j).zfill(5))
        else:
            SubElement(route, 'link_ctn', id = str(j + host_id - 2).zfill
    (5))

# dump XML tree on file
f = open(os.getcwd() + '/pf_' + str(total_cores) + '_' + str(
    max_core_by_vm) + '.xml',"wb")
tree = etree.fromstring(ElementTree.tostring(platform))
f.write(etree.tostring(tree, encoding="UTF-8", xml_declaration=True,
    pretty_print=True,
                       doctype='<!DOCTYPE platform SYSTEM "http://simgrid.
    gforge.inria.fr/simgrid/simgrid.dtd">'))
f.close()
```

# B Exemple de plateforme

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid/simgrid.dtd">
<platform version="4.1">
  <zone id="AS0" routing="Full">
    <host core="1" id="00000" speed="1000Gf"/>
    <host core="1" id="00001" speed="1000Gf"/>
    <host core="96" id="00002" speed="1000Gf"/>
    <host core="96" id="00003" speed="1000Gf"/>
    <host core="96" id="00004" speed="1000Gf"/>
    <link bandwidth="25Gbps" id="switch" latency="100us" sharing_policy="FATPIPE"/>
    <link bandwidth="1750MBps" id="00000" latency="100us" sharing_policy="FATPIPE"/>
    <link bandwidth="1750MBps" id="00001" latency="100us" sharing_policy="FATPIPE"/>
    <link bandwidth="3125MBps" id="00002" latency="100us"/>
    <link bandwidth="3125MBps" id="00003" latency="100us"/>
    <link bandwidth="3125MBps" id="00004" latency="100us"/>
    <link bandwidth="1750MBps" id="00005" latency="100us"/>
    <link bandwidth="1750MBps" id="00006" latency="100us"/>
    <link bandwidth="1750MBps" id="00007" latency="100us"/>
    <route dst="00001" src="00000">
      <link_ctn id="00000"/>
      <link_ctn id="switch"/>
      <link_ctn id="00001"/>
    </route>
    <route dst="00002" src="00000">
      <link_ctn id="00000"/>
      <link_ctn id="switch"/>
      <link_ctn id="00002"/>
    </route>
    <route dst="00003" src="00000">
      <link_ctn id="00000"/>
      <link_ctn id="switch"/>
      <link_ctn id="00003"/>
    </route>
    <route dst="00004" src="00000">
      <link_ctn id="00000"/>
      <link_ctn id="switch"/>
      <link_ctn id="00004"/>
    </route>
    <route dst="00002" src="00001">
```

```
      <link_ctn id="00005"/>
    </route>
    <route dst="00003" src="00001">
      <link_ctn id="00006"/>
    </route>
    <route dst="00004" src="00001">
      <link_ctn id="00007"/>
    </route>
    <route dst="00003" src="00002">
      <link_ctn id="00002"/>
      <link_ctn id="switch"/>
      <link_ctn id="00003"/>
    </route>
    <route dst="00004" src="00002">
      <link_ctn id="00002"/>
      <link_ctn id="switch"/>
      <link_ctn id="00004"/>
    </route>
    <route dst="00004" src="00003">
      <link_ctn id="00003"/>
      <link_ctn id="switch"/>
      <link_ctn id="00004"/>
    </route>
  </zone>
</platform>
```

# C APIs WRENCH utilisés

Tableau 4 – Liste non exhaustive des APIs utilisés.

| **APIs name** | **Description** |
|---|---|
| CloudComputeService | A cloud-based compute service that manages a set of physical hosts and controls access to their resources by (transparently) executing jobs in VM instances |
| ComputeService | The compute service base class |
| SimpleStorageService | A storage service that provides direct access to some storage resources |
| Simulation | A class that provides basic simulation methods |
| SimulationTimestamp | A time-stamped simulation event stored in SimulationOutput |
| StorageService | The storage service base class |
| WMS | A workflow management system (WMS) |
| Workflow | A workflow (to be executed by a WMS) |
| WorkflowFile | A data file used/produced by a WorkflowTask in a Workflow |
| WorkflowTask | A computational task in a Workflow |

# D Note concernant les unités de stockage

Dans cette thèse, les unités du système international d'unités (SI) sont utilisées. Nous appliquons ici, les unités de tailles de données selon les préfixes du système SI. Nous n'utilisons pas dans cette thèse les recommandations internationales de la norme *International Electrotechnical Commission* (IEC 60027-2).

Tableau 5 – Multiples de l'octet, selon le système international d'unités (SI).

| **Nom** | **Symbole** | **Valeur (octets)** |
|---|---|---|
| Kilooctet | Ko | $2^{10}$ = 1 024 |
| Mégaoctet | Mo | $2^{20}$ = 1 048 576 |
| Gigaoctet | Go | $2^{30}$ = 1 073 741 824 |
| Téraoctet | To | $2^{40}$ = 1 099 511 627 776 |