Energy-aware management of scientific workflows in the Cloud : a Cloud provider-centric vision

Emile Cadorel

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THÈSE DE DOCTORAT DE

l'École Nationale Supérieure
Mines-Télécom Atlantique
Bretagne Pays de la Loire - IMT Atlantique

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par

Emile CADOREL

Energy-aware management of scientific workflows in the Cloud : A Cloud provider-centric vision

Thèse présentée et soutenue à Nantes, le 21 octobre 2020
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes (LS2N)
Thèse N° : 2020IMTA0195

Rapporteurs avant soutenance :

Romain ROUVOY Professeur (HDR), Université de Lille
Patricia STOLF Maîtresse de conférence (HDR), Université de Toulouse Jean-Jaurès

Composition du Jury :

Examinateurs :	Frédéric DESPREZ	Directeur de recherche (HDR), Inria Grenoble Rhones Alpes
	Stéphane GENAUD	Professeur (HDR), Université de Strasbourg
	Romain ROUVOY	Professeur (HDR), Université de Lille
	Patricia STOLF	Maîtresse de conférence (HDR), Université de Toulouse Jean-Jaurès
Dir. de thèse :	Jean-Marc MENAUD	Professeur (HDR), IMT Atlantique
Co-encadrante de thèse :	Hélène COULLON	Maîtresse assistante, IMT Atlantique

Invité(s) :

Georges DA COSTA Maître de conférence (HDR), Université de Toulouse Paul Sabatier

Remerciements

Je souhaiterais tout d'abord remercier mes deux encadrants Jean-Marc Menaud, et Hélène Coullon, pour leur soutien et leur contribution durant ces trois années de thèse. Tous les travaux présentés dans les pages de ce manuscrit n'auraient pu être possibles sans leurs précieux conseils et remarques. Je les remercie pour leur supervision avisée, sur des questions à la fois scientifiques et pédagogiques, qui m'aura transmis une méthode de travail efficace et éclairée, que je m'efforcerai de préserver par la suite.

J'adresse également mes remerciements aux rapporteurs de ce manuscrit, Romain Rouvoy et Patricia Stolf, pour leur patient travail de relecture. La lisibilité et la qualité de ce document ont pu être grandement améliorées grâce à leurs commentaires pertinents et précis. Je remercie également Frédéric Desprez, Stéphane Genaud et Georges Da Costa, d'avoir accepté de faire partie de mon jury de thèse.

Tout au long de ces trois années de thèse j'ai eu la chance de pouvoir interagir avec des personnes remarquables. Je souhaite adresser un grand remerciement à l'ensemble de l'équipe STACK pour leur accueil chaleureux et leur convivialité. Un remerciement particulier à toutes les personnes avec qui j'ai eu l'occasion de m'asseoir pour boire un verre et parler d'informatique ou bien d'autres sujets.

Merci aussi à toute ma famille, notamment mes grands-mères, mes oncles et tantes, et bien évidemment mes deux frères Benoît et Paul et mes parents, dont le soutien et les encouragements constants m'ont été d'une grande aide et ont contribué à l'aboutissement de ce travail.

Abstract

Computer-based scientific simulations and experimentation are generally very complex and are composed of many parallel processes. In order to easily highlight the parallelizable parts of such applications, and to allow efficient execution, many scientists have opted to define their applications as scientific workflows. A scientific workflow represents an application in the form of a set of unitary processing tasks (e.g. data formatting, performing an analysis, etc.), linked by dependencies. Nowadays, thanks to their low cost, elasticity and on-demand aspect, Cloud computing services are widely used for the execution of scientific workflows. Users executing workflows on such environment manage the execution of their workflow as well as the needed resources, using a standard service such as IaaS (Infrastructure-as-a-Service). Nevertheless, since Cloud computing services are not specific to the nature of the application to be run, resource usage is not as optimized as it could be.

In this thesis we propose to shift the management and the execution of scientific workflows on the Cloud computing provider's side in order to offer a new type of service dedicated to scientific workflows. This new approach aims at improving resource management for a Cloud provider, by providing additional information on the nature of the applications to be executed. This improvement leads to a reduction of the energy consumption, thus to a reduction of the environmental impact of the distributed infrastructure. In this context, two problems are raised, namely, the resolution of the scheduling problem, and the execution of the computed planning. On the one hand, to solve the scheduling problem, we introduce two algorithms. These algorithms solve the problem of scheduling several scientific workflows submitted by several users, on a distributed infrastructure. We show through experiments on real infrastructures that they offer an important gain in terms of energy consumption. On the other hand, to be able to execute a planning computed by a scheduling algorithm, we propose a new Cloud computing service called WaaS (Workflow-as-a-Service). This service is designed as a turnkey solution for a Cloud computing provider, allowing extensibility in the management of virtualization mechanisms, extensibility in scheduling policies, and consideration of scalability issues.

Résumé

Beaucoup de scientifiques réalisant des simulations ou des expériences sur ordinateur ont pris l'habitude de développer leurs applications sous la forme de ce que l'on appelle des workflows scientifiques. Les workflows peuvent être considérés comme un outil permettant de séparer les différentes étapes d'une opération complexe en plusieurs tâches plus simples et interdépendantes, par exemple récupérer des données à partir d'un instrument de mesure, formater des données ou effectuer des analyses sur les données acquises. Un workflow scientifique décrit donc la topologie d'une application comme un ensemble de tâches liées entre elles par des dépendances, ces dépendances définissant l'ordre d'exécution de chaque tâche.

En séparant le traitement de l'application en un ensemble de tâches, les traitements qui peuvent être exécutés en parallèle sont mis en évidence. Les workflows peuvent être composés d'un grand nombre de tâches et de dépendances, et peuvent donc devenir très complexes. Afin d'obtenir des temps de calcul raisonnables, il est nécéssaire d'utiliser des infrastructures offrant de nombreuses ressources de calcul pour tirer parti de la parallélisation et ainsi réduire le temps d'attente avant l'obtention des résultats. Les infrastructures distribuées semblent être le choix idéal pour résoudre ce problème. Elles tirent parti du regroupement d'un ensemble de machines physiques communiquant via un réseau pour obtenir plus de puissance de calcul qu'une seule machine physique ne pourrait en fournir.

Les infrastructures distribuées peuvent être compliquées à gérer, et les scientifiques qui développent des workflows scientifiques ne sont généralement pas des experts en informatique, en gestion de matériel ou en gestions des dépendances logiciels. En effet, pour pouvoir exécuter les workflows, les scientifiques s'appuient généralement sur des outils appelés moteurs de workflows. Ces moteurs prennent en charge l'exécution des workflows scientifiques dans un environnement offrant des ressources de calculs. Ces moteurs de workflow, en plus de permettre l'exécution automatique des workflows, sont un bon moyen d'améliorer la réutilisation des expérimentations, car ils masquent la complexité de l'infrastructure et sont capables de reproduire l'exécution. Cette amélioration de la réutilisation et de la ré-éxecution permet tout autant d'améliorer le partage des résultats.

Aujourd'hui, la grande majorité des moteurs de workflow proposent d'utiliser le Cloud

computing comme environnement d'exécution. Le Cloud computing est devenu un environnement très populaire ces dernières années, fournissant des ressources de calculs et de stockage à faible coût, et à la demande. Le Cloud computing peut définir différents types de services, offrant différents types d'abstraction des ressources mises à disposition. Ces services s'appuient sur des technologies de virtualisation pour permettre aux utilisateurs d'accéder aux ressources de calcul ou de stockage de manière élastique. L'élasticité désignant la capacité pour un utilisateur à accéder rapidement à une ressource de calcul ou de stockage et de pouvoir la libérer lorsqu'elle n'est plus nécessaire. Cette gestion des ressources dans un environnement Cloud donne aux utilisateurs l'illusion de disposer d'un nombre infini de ressources, et ceux-ci sont donc uniquement limitées par leur budget. Dans ce contexte, les moteurs de workflows qui visent l'exécution dans un environnement de Cloud computing ne se préoccupent pas de la gestion des ressources physiques - le support des ressources virtuelles utilisées pour l'exécution des différentes tâches des workflows - et laissent cette préoccupation au fournisseur de Cloud.

Malheureusement, cela conduit à une déconnexion entre la réalité logicielle et la réalité matérielle. D'un côté les fournisseurs de Cloud n'étant pas informés du type d'application s'exécutant sur leurs ressources, ne peuvent efficacement optimiser la gestion de leur infrastructure physique. D'un autre côté, les moteurs de workflow, même s'ils sont informés de la forme des applications à exécuter, ne peuvent fournir d'optimisation pour la gestion de l'infrastructure physique, comme cette préoccupation est entièrement laissée au fournisseur de Cloud. Dans cette thèse, nous introduisons une nouvelle approche pour la gestion et l'exécution des workflows scientifiques, en plaçant les décisions d'allocation de ressources du côté du fournisseur de Cloud, afin d'offrir un nouveau type de service de Cloud dedié au workflows scientifiques. Grâce à cette approche, les fournisseurs de Cloud peuvent être plus informés sur les applications tournant sur leur infrastructure, et par conséquent peuvent prendre des décisions plus intelligentes quant à la gestion de leurs ressources.

Contributions

Dans cette thèse est introduite une nouvelle approche pour la gestion des ressources physiques, dédiée aux workflows scientifiques. La nouvelle approche vise à gérer l'infrastructure physique plus efficacement qu'elle ne le serait en utilisant un service Cloud existant. Afin de pouvoir exécuter les workflows scientifiques sur une infrastructure distribuée, en dé-

plaçant la gestion des workflows du côté du fournisseur de Cloud, nous proposons de nous concentrer sur les deux étapes de l'exécution des workflows : la résolution du problème de l'ordonnancement et l'exécution de la planification calculée. L'ordonnancement d'un workflow scientifique consiste à créer un planning où chaque tâche du workflow est associée à une ressource de calcul à un instant donné.

L'exécution de la planification, outre la mise à disposition et la libération des ressources, doit pouvoir garantir le respect des dépendances entre les tâches des workflows (dépendances de fichiers). En effet, chaque tâche d'un workflow scientifique, prend des fichiers en entrée et produit des fichiers, qui doivent être transférés aux tâches qui lui succèdent.

Les contributions principales de cette thèse sont les suivantes :

— **Algorithme statique d'ordonnancement de workflows scientifiques pour l'optimisation énergétique dans un environnement de Cloud computing -** Cet algorithme d'ordonnancement est présenté comme une solution pour un fournisseur de Cloud qui vise à réduire la consommation d'énergie de son infrastructure physique. Cet algorithme vise à minimiser le nombre de machines physiques nécessaires pour exécuter un ensemble de workflows scientifiques soumis par les utilisateurs dans l'environnement de Cloud à un instant donné. En effet, l'un des principaux coûts d'exploitation d'un fournisseur de Cloud computing est la consommation d'énergie. Cette consommation peut être réduite en limitant le nombre de machines physiques sous-utilisées [43, 79]. À cette fin, dans le problème abordé dans cette contribution, une date limite est associée à chaque workflow. L'algorithme proposé est basé sur une fonction heuristique, qui ne prend en compte que les machines physiques déjà utilisées lors de l'ordonnancement des tâches d'un workflow, tant que le délai est respecté (ordonnancement des workflows les uns après les autres, triés par priorité). Lorsque la date limite ne peut plus être respectée, l'algorithme fait marche arrière et prend en compte de nouvelles machines physiques (celles qui ne sont pas utilisées). Ce travail a été publié dans la conférence internationale GreenCom 2019 [46].

— **Algorithme dynamique d'ordonnancement de workflows scientifiques pour l'optimisation de l'équité entre les utilisateurs et l'optimisation énergétique -** Cet algorithme peut être défini comme un algorithme d'ordonnancement de workflows scientifiques pour un fournisseur de Cloud computing, qui vise à réduire la consommation d'énergie de son infrastructure, tout en assurant l'équité entre les utilisateurs. Comme pour notre première contribution, à chaque workflow est as-

sociée une date limite, l'équité entre les utilisateurs étant définie comme le respect de la date limite associée à leurs soumissions. Cet algorithme se situe dans un contexte où les soumissions des utilisateurs sont faites à des moments incertains, ce qui doit être pris en compte afin de garantir l'équité entre les utilisateurs. Comme les nouvelles soumissions peuvent arriver à tout moment, l'algorithme est capable de reconsidérer le planning déjà calculé contenant les anciennes soumissions. En effet, les nouveaux workflows soumis peuvent avoir une priorité plus élevée (délais plus courts) que les workflows déjà ordonnancés. L'autre objectif de l'algorithme (outre l'optimisation de l'équité) est la minimisation de la consommation d'énergie, qui est réduite en localisant les différentes tâches sur les mêmes machines physiques. Ce travail a été publié dans la conférence internationale CCGrid 2020 [47].

— **WaaS : Workflow-as-a-Service, un service de Cloud computing pour l'exécution de workflows scientifiques -** Le WaaS est un nouveau service de Cloud computing spécifiquement dédié à l'exécution de workflows scientifiques. Nous montrons que ce choix conceptuel renforce la séparation des préoccupations entre les utilisateurs finaux - les acteurs qui soumettent un workflow pour une exécution - et les fournisseurs de Cloud - les acteurs qui gèrent une infrastructure physique et fournissent des ressources aux utilisateurs finaux. Le concept améliore également la gestion des ressources du point de vue des fournisseurs de Cloud. Pour faciliter l'intégration du nouveau service WaaS par les fournisseurs de Cloud, il a été conçu comme une solution clé en main qui permet une modularité des mécanismes de virtualisation et des politiques d'ordonnancement, ainsi que des problèmes de passage à l'échelle, permettant aux fournisseurs de Cloud de personnaliser le service pour des besoins spécifiques.

Toutes les contributions présentées dans ce manuscrit ont été évaluées à travers des expérimentations menées sur une infrastructure réelle. Ces expériences ont été menées sur la plateforme d'expérimentation Grid'5000 [13], et les codes sources utilisés dans les expérimentations sont disponibles sur des dépôts publics afin de pouvoir reproduire les expériences [1] [2].

1. `https://gitlab.inria.fr/ecadorel/workflowplatform`
2. `https://gitlab.inria.fr/ecadorel/waas/`

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Contents

1.1 Research problem

Scientists performing computer-based simulations or experiments have become accustomed to developing their applications in the form of what is known as a scientific workflow. Workflows can be seen as a tool to separate the different steps of a complex operation into several simple interdependent tasks, for example retrieve data from measure instrument, format data or run analysis on acquired data. A scientific workflow thus describes the topology of an application as a set of tasks linked together by dependencies, these dependencies defining the order of execution of each task, and the files to be transferred from a task to its successors.

By separating the application processing into a set of simple tasks, the processes that can be performed in parallel are highlighted. Workflows can be composed of a large number of tasks and dependencies, and therefore become very complex. In order to achieve reasonable calculation times, infrastructures offering many computing resources must be used to take advantage of the parallelism and thus reduce the waiting time before results are obtained. Distributed infrastructure seemed to be the perfect choice to resolve this problem. Indeed, they take advantage of the pooling of a large number of physical machines, that communicate through a network, to obtain more computing power than a single physical machine could possibly provide.

Distributed infrastructures can be complicated to manage, and the scientists develop-

ing scientific workflows are generally not experts in computer science, hardware management, or software dependencies management. Indeed, to be able to execute the workflows, they generally rely on tools called workflow engines. These engines take care of the execution of workflows in an environment offering computing resources. These workflow engines in addition to providing automatic execution of workflows are a good way to enhance reusability and reproducibility of experimentation and the sharing of results in scientific communities, as they hide the complexity of the infrastructure and are able to reproduce the execution.

Today, the vast majority of workflow engines offer to use Cloud computing as an execution environment. Cloud computing has become a very popular environment in recent years, providing low-cost computing and storage resources. Cloud computing defines different types of services with different abstraction levels, from low-level infrastructure services toward high-level ready to use softwares. These services rely on virtualization technologies to give users access to computing resources in an elastic manner, when elasticity refers to the ability of the user to quiclky access to computing or storage resources and to be able to release them when they are no longer needed. This management of resources in a Cloud environment gives to the users the illusion of having an infinite number of resources, limited only by their budget. In this context, workflow engines that handle Cloud environments do not worry about the management of physical resources - the support of virtual resources used for the execution of different workflow tasks - and leave this concern to the Cloud provider.

Unfortunately, this leads to a disconnection between the reality of software and hardware world. On the one hand, Cloud providers, because they are not aware of the type of applications running on their resources, cannot efficiently optimize the management of their physical infrastructures. On the other hand, workflow engines, even if they are aware of the type of applications to execute, cannot provide optimizations for the management of the physical infrastructure, as this concerns is left entirely to the Cloud provider. In this thesis, we introduce a new approach for the management and execution of scientific workflows by locating the decision of resource allocation on the Cloud provider side, in order to offer a new kind of Cloud service dedicated to scientific workflows. Thanks to this approach, the Cloud providers can be more aware of the application running on their infrastructures, and as a result can make smarter decisions regarding the management of their resources.

1.2 Contributions

In this thesis is introduced a new approach for the management of physical resources, dedicated to scientific workflows. The new approach aims at managing the physical infrastructure more efficiently than it would be using an existing Cloud service. In order to be able to run scientific workflows on a distributed infrastructure, by shifting workflow management to the Cloud provider side, we propose to focus on the two steps of workflow execution: solving the scheduling problem, and executing the computed planning. The scheduling of a scientific workflow consists in the creation of a planning where each task of the workflow is associated with a computational resource at a given instant.

The execution of the planning, in addition to carrying out resource provisioning and releases, must be able to guarantee that dependencies between the workflow tasks (file dependencies) are respected. Indeed, each task of a scientific workflow, takes files as input and produces files, that are to be transferred to its successors tasks.

The main contributions of this thesis can be summarized as follows:

— **Static scientific workflow scheduling algorithm for energy optimization in a Cloud computing environment -** This scheduling algorithm is presented as a solution for a Cloud provider that aims to reduce the energy consumption of its physical infrastructure. This algorithm aims to minimize the number of physical machines required to execute a set of scientific workflows submitted by users in the Cloud environment at a given instant. Indeed, one of the main operating costs of a Cloud computing provider is power consumption. This consumption can be reduced by limiting the number of underused physical machines [43, 79]. To this end, in the problem addressed in this contribution, a deadline is associated to each workflow. The proposed algorithm is based on a heuristic function, that considers only the already used physical machines while scheduling the tasks of a workflow as long as the deadline is respected (scheduling the workflow one by one, sorted by priority). When the deadline can no longer be respected, the algorithm performs a backtrack, and considers new physical machines (unused one). This work has been published in the GreenCom 2019 international conference [46].

— **Dynamic scientific workflow scheduling algorithm for user fairness and energy Optimization -** This algorithm can be defined as a scientific workflow scheduling algorithm for a Cloud computing provider, that aims at reducing the energy consumption of its infrastructure, while ensuring fairness or equity between

users. As for our first contribution, to each workflow is associated a deadline, the fairness between the users being defined as the respect of the deadline associated to their submissions. This algorithm takes place in a context where user submissions are made at uncertain times, which must be taken into account in order to guarantee fairness between users. As the new submissions can arrive at any moment, the algorithm is able to reconsider the already computed planning containing the old submissions. Indeed, new submitted workflows may have higher priority (shorter deadlines) than already scheduled workflows. The other objective (in addition to fairness optimization) of the algorithm is the minimization of the energy consumption, which is reduced by locating the different tasks on the same physical machines. This work has been published in the CCGrid 2020 international conference [47].

— **WaaS: Workflow as a Service, a Cloud computing service for scientific workflow execution -** The WaaS is a new Cloud computing service specifically dedicated to the execution of scientific workflows, with a *service specific vision*. We show that this conceptual choice reinforces the separation of concerns between the end-users - actors that submit a workflow for execution - and the Cloud providers - actors that manage a physical infrastructure, and provide resources to the end-users. The concept also improves resource management from the perspective of the Cloud providers. To facilitate the integration of the new WaaS service by Cloud providers, it has been designed as a turnkey solution that addresses modularity of virtualization mechanisms and scheduling policies, as well as scalability issues, enabling for the Cloud providers some customizations of the service for specific needs.

All the contributions presented in this manuscript have been evaluated through experimentation carried out on a real infrastructure. These experiments were conducted on the Grid'5000 [13] experimental platform, and the source codes used for the execution are available on public repositories in order to reproduce the experimentations [1] [2].

1. `https://gitlab.inria.fr/ecadorel/workflowplatform`
2. `https://gitlab.inria.fr/ecadorel/waas/`

1.3 Publications

Papers in International conferences

— **Emile Cadorel**, Hélène Coullon, and Jean-Marc Menaud. A workflow scheduling deadline-based heuristic for energy optimization in Cloud. In ***GreenCom 2019*** *- 15th IEEE International Conference on Green Computing and Communications, Atlanta, United States,* 2019. IEEE.

— **Emile Cadorel**, Hélène Coullon, and Jean-Marc Menaud. Online Multi-User Workflow Scheduling Algorithm for Fairness and Energy Optimization. In ***CCGrid 2020*** *- 20th International Symposium on Cluster, Cloud and Internet Computing, Melbourne, Australia,* 2020.

Papers in National conferences

— **Emile Cadorel**, Hélène Coullon, and Jean-Marc Menaud. Ordonnancement multi-objectifs de workflows dans un Cloud privé. In ***ComPAS 2018*** *- Conférence d'informatique en Parallélisme, Architecture et Système, Toulouse, France,* 2018.

1.4 Thesis organization

The rest of this manuscript is organized as follows :

The first chapter presents the context of our research. Section 2.1 presents an overview on scientific workflows, and the challenges associated to their management. Section 2.2 presents the different types of physical infrastructures that are targeted for the execution of scientific workflows. Section 2.3 introduces the operating systems and virtualization mechanisms used in the physical infrastructure management paradigms presented in Section 2.4. Section 2.5 presents the scheduling problem, and finally Section 2.6 presents the problematics of energy consumption within infrastructures, and techniques that can be used to reduce it.

The second chapter presents the state of the art of this thesis. Section 3.1 presents an analysis on the literature regarding the scheduling of scientific workflows within both Grid computing and Cloud computing environments. Section 3.2 presents the related work that has been carried out regarding the automatic execution of scientific workflows.

The third chapter presents our first contribution on the scheduling of scientific workflows. This presentation is divided in two parts. Section 4.2 details the problem modeling, Section 4.3 presents our new algorithm OnlyUsedNodes.

The fourth chapter presents our second contribution on the scheduling of scientific workflows. This presentation is divided in two parts. Section 5.2 gives a detailed model of the problem. Then Section 5.3 presents our new algorithm NearDeadline.
The fifth chapter presents detailed evaluations of our two algorithms presented in Chapter 4 and Chapter 5. This evaluation is separated in four parts. Section 6.1 present a evalution conducted on a simulator of the algorithm presented in Chapter 4. Then 6.2 presents a environment of execution on a real infrastructure, that is used in the two evaluations presented in Section 6.3 and Section 6.4.
The sixth chapter presents our last contribution on the execution of scientific workflows. This presentation is divided in 2 parts. Section 7.2 details the contribution, *i.e.*, the submission language, the architecture and the modular aspect of the new WaaS service, and Section 7.3 evaluates this solution on a real multi-cluster infrastructure.
The seventh chapter concludes this manuscript. In this chapter, the contributions are summarized and conclusions about the scheduling and the execution of scientific workflows in a virtualized distributed infrastructure are presented. This conclusion discusses the limitations of our solutions and describes some perspectives.

Chapter 2

Context

The purpose of this chapter is to give a context to our contributions. First, we present in detail the concept of scientific workflow, and the key elements necessary to be able to execute one. Next, we present physical infrastructures that can be used to execute scientific workflows, followed by an introduction to operating system and virtualization mechanisms used to manage the physical machines. Thirdly, we present paradigms for resource management in distributed infrastructures. Fourthly, we present the scheduling problem, which must be solved in order to run scientific workflows on a distributed infrastructure. Finally, we present the energy concerns regarding the physical infrastructure and the energy-saving techniques that can be used in this context.

Contents

2.1 Scientific workflows: overview and challenges

Scientific workflows model scientific applications as a set of coarse-grained tasks linked together by data dependencies. Scientific workflows have emerged as an alternative to ad-hoc approaches for scientist where the efficiency and reproducibility of experimentation are enhanced. Indeed, the concept of scientific workflow allows users to easily define multi-step applications in distinct computational tasks, for example: retrieving data from measure instrument, formating data or running analysis on acquired data. A workflow defines the dependencies between the tasks, and is generally defined as a DAG (Directed Acyclic Graph) [37, 54, 69, 78], where the nodes model the computation tasks, and the arrows the dependencies between the tasks. Modeling a scientific workflow as a DAG is an interesting approach to define efficient scientific applications. Indeed, by defining an application as a flow of unitary operations (single treatment), the parts of the global application that can be executed in parallel are highlighted naturally. Workflows can be composed of an arbitrary number of tasks, as well as an arbitrary number of dependencies between the tasks.

The workflow tasks take input files as entry, and produce output files. These output files are then used by other tasks of the workflow as their input files and so on, thus creating data dependencies between tasks. A simple example of scientific workflow is presented in Figure 2.1, where the tasks B and C depend on the task A, and need to wait the end of its execution, but can be executed in parallel as they do not depend on each other.

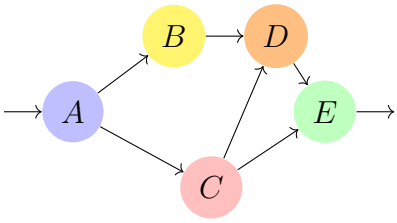


Figure 2.1 – Example of scientific workflow

There is no accepted consensus on the technology or language that should be used to describe the tasks making up the workflows, that are addressed in this thesis. For example, the *Montage* workflow [35] is developed using the C language and can be run in a Linux environment, when the workflow *diffcyt* [129], uses libraries and scripts developed in the R language [75]. In addition, some scientists might be constrained by the scientific equipment they use to create the input data of their workflow. For example, genomics

workflows designed in [109, 111] use data produced by a vendor specific machine [1]. To convert the output files formatted by the machine into standard format, vendor specific software running on the Windows operating system has to be used, while the other applications used in the workflow (*e.g.*, Openswath [112]) must be executed on top of Linux. This heterogeneity allows developers to be quite flexible in the definition of workflows, but introduces challenges in their management. Indeed, one can therefore define a task of a workflow as a program with hardware and software requirements. The hardware requirements defining, for example, that a program may need a certain amount of memory, or a certain amount of CPU cores, when the software requirements refers to the need of a program to be executed in a specific environment such as a specific operating system, along with the correctly installed libraries.

In this thesis, we also assume that the users developing workflows have only basic knowledge in computer science, and no knowledge in resource and infrastructure management. To be able to run a scientific workflow, users typically use workflow engines [27, 56, 59, 86]. These engines are middlewares that bridge the gap between the application (*i.e.*, the scientific workflow) and the resources available for execution. A workflow engine manages the execution of the different tasks making up the workflow, as well as file transfers from the generating (*i.e.*, source) tasks to the consuming (*i.e.*, destination) tasks. They are deployed on a computing infrastructure that contains from one to several resources. The type of infrastructures on which workflow engines can be deployed will be presented in the next section.

A workflow engine can be divided into two parts that address two different problems, namely *scheduling* and *execution*. On the one hand, scheduling will be presented in detail in Section 2.5. This problem is defined as the creation of a *planning*. A planning associates to each task of one or multiple workflows, resource reservations on the infrastructure for a given amount of time. On the other hand, workflow execution represents the execution of the planning that has been calculated by the scheduler. These two parts are the two main problems needed to be solved to easily and optimally execute scientific workflows.

2.2 Physical infrastructures

This section presents the physical infrastructure targeted for the execution of scientific workflows. A physical infrastructure is composed of physical machines, which may contain

1. https://sciex.com/Documents/Downloads/Literature/Tech-Note-MSMSall-SWATH-Acquisition.pdf

different hardware components (CPU, memory, hard disk, GPU, etc.). A physical machine can contain several CPUs, and many CPU cores, allowing several applications to run in parallel on the same machine. In the remainder of this document, physical machines will be referred to by the abbreviation *PM* or *node*.

A cluster is a set of PMs at the same location (same geographical point) and communicating through a network. By using several PMs instead of one, computing power is improved by allowing even more concurrent computing operations. PMs within a cluster generally communicates through the same LAN (local area network), thus the bandwidth between them is assumed homogeneous in this document.

Multiple cluster can communicate through a network (linking the different LANs of the clusters and creating a N-LAN), creating a more complex physical infrastructure, that is called a multi-cluster infrastructure. The network creating a link between the clusters of the multi-cluster infrastructure can be complex and the bandwidth between the cluster may be heterogeneous. The clusters of a multi-cluster infrastructure are located at the same geographical point, grouped in data centers or computing centers. Data center and computing center referring to the capacity of their PMs to be efficient for storing or computing respectively.

The final level of physical infrastructure that can be created is the aggregation of several multi-cluster infrastructures, located at different geographical points. This type of infrastructure is called a geo-distributed infrastructure in this document. As for multi-cluster infrastructure, the bandwidth between the multi-clusters can be heterogeneous.

2.3 Operating system and Virtualization

In order to manage the various resources (hardware components) of a PM, an OS (operating system) is used. An OS is a software deployed on a PM creating a bridge between the applications and the hardware components of the PM. To perform this operation, an OS offers system calls accessible by the applications, allowing them to communicate with the hardware. The OS is also responsible for managing the execution of the applications and their access to the hardware, deciding on the reservation of resources in order to avoid resource access conflicts. The OS is the only software managing the hardware components of a PM, and only one OS can run at a given instant on a PM.

An OS is the first software that is started on a PM, when it is powered on. The operation of launching the operating system is called the boot process, and takes a certain

amount of time, as the OS needs to perform some operations to be operational. Some OS are able to manage multiple users using a PM at the same time, by dividing the resources between the users. However, this division generally does not guarantee that a user will not use all the resources of a PM, or create an overload, making the PM unusable for the other users.

Virtualization technologies emerged as a solution to that problem. Indeed, virtualization technologies such as VM (virtual machine) allows to divide the resources of a PM, and assign a certain amount of resources to a given user. Thus, users can no longer use more resources than those assign to them. Virtualization technologies can also be used to manage the resources of a PM for applications with different requirements that cannot be totally fulfilled by the OS alone. For example, applications that need to be executed in different operating system, applications using conflicting software dependencies, etc.

2.3.1 Virtual machines

Virtual machine is a virtualization mechanism, that works as follows: on a given PM running a specific OS (named the *host OS*), a software named *hypervisor* or Virtual Machine Monitor [106] is running. An hypervisor is able to create VM (virtual machine). A VM is the emulation of a PM and as for a PM is composed of different hardware components (CPU, memory, Hard drive, etc.), but those components are however emulated, and are consequently virtual - thus for example, the term VCPU (virtual CPU) will be used when referring to the CPU cores of a VM. An hypervisor uses the host OS to handle and reserve the shared resources of the host PM for VMs.

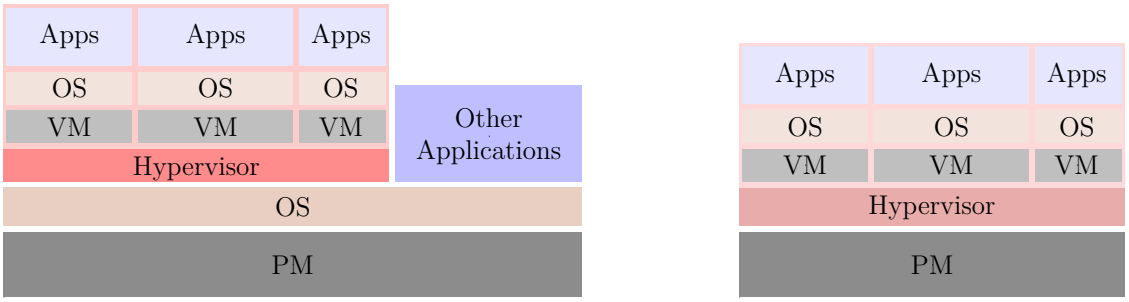
Each VM has access to virtual hardware resources and have the capacity to run a given OS. From the OS system point of view, there is no difference between a VM and a PM. Figure 2.2 presents PMs running VMs with the two existing types of hypervisor.

There are two types of hypervisor:

— *Hosted* (Figure 2.2a): the hypervisor of this type are running as a standard application and are communicating with the hardware components via the OS. Qemu [30], VirtualBox [128] are widely used hosted hypervisors.

— *Native* (Figure 2.2b): the hypervisors of this type are running at the OS level, and thus have access to the physical resources directly. With this type of hypervisor only virtual machines are launchable on the PM. Xen [28] and VMWare ESXI [95] are widely used native hypervisors.

Sometimes, hypervisors are an hybrid of a *hosted* and a *native* hypervisor, such as KVM [107] which is running along with an OS, and thus even if it has access to the physical components, an OS is running on the PM and can run applications other than just VMs.



(a) Hosted Hypervisor

(b) Native Hypervisor

Figure 2.2 – The two types of Hypervisor

As multiple VMs can be emulated on the same PM, multiple OS can be executed at the same time on the same PM, however each one of them is running in an isolated environment. That is to say, that each *guest OS* (OS running inside a VM), is aware of the hardware components of its VM, and is isolated from the other guest OSs and from the host OS (OS running inside the PM running the VMs). This isolation allows the hypervisor to reserve only a subset of the PM resources for a VM, and use the remaining resources for other VMs, or to leave these resources for other applications running on the host OS. This isolation can also be used for security reasons as applications running inside a VM on a guest OS are not aware of other VMs and their respective applications, and only have access to the virtual resources emulated by the hypervisor.

A VM being the emulation of a PM, is therefore operating at the software level, and is not different from a standard application. A single file named an image is used to represent a template of a VM. This image stores the hard drive storage (installed softwares, user directories, etc.), as well as information about the emulated hardware components (number of VCPUs, quantity of RAM, etc.). This image is used by the hypervisor, in order to create a VM. Thanks to this image file, a VM can be created easily by making a copy of this file, on any PM running the correct hypervisor. Once the VM has been created by the hypervisor in accordance with the description of the image file, the VM has to be booted. Indeed, like a PM, a VM can be turned off, and has to be turned on to be usable, and therefore the guest OS running inside the VM needs to be booted. This

may require some time, depending on the hypervisor and the capacity of the PM running the hypervisor. This booting process generally ranges from seconds to minutes [97].

2.3.2 Containers

Containers are another widely used virtualization mechanism. Unlike VMs, Containers brings virtualization of the execution environment, and not of virtual hardware components running a guest OS, that is to say containers share the same operating system as the physical machine (or the virtual machine) that is hosting them. Containers uses the system calls provided by the host OS in order to access to hardware components, as would do any application, thus removing the overhead induced by emulating virtual components in VMs. However, this makes it impossible to run different operating system on the same PM at the same time. It is also impossible to move a container from a PM to another PM running different OS.

Historically the idea behind the container technology was to provide the possibility of installing different software components on a given PM without creating software conflicts. To this end, a container creates a namespace isolation, by providing a new lib directory (directory containing the installed softwares), than the one provided by the host OS. Therefore software can be installed and accessed inside the container without impacting the configuration of the host OS, and the other application running inside the PM.

The isolation of the Container is also made at the hardware management level. Indeed, some container systems - such as Docker [8] - make it possible to reserve a subset of the physical resources for a given container, thus ensuring that a given application will not consume more resources than those reserved for it. This isolation is made by using the system calls of the host OS, as unlike VMs, the hardware components used inside a container are not emulated.

Figure 2.3 represents a PM containing three containers. Docker [8], LXC [15] and Singularity [19] are widely used container systems.

A container can be represented by a single file, called an image. This image stores directories, including the lib directory, and user directories. This image is used to create a container on one PM or to be copied to another PM to create a container on it, however a container can only be created on a PM using the correct host OS.

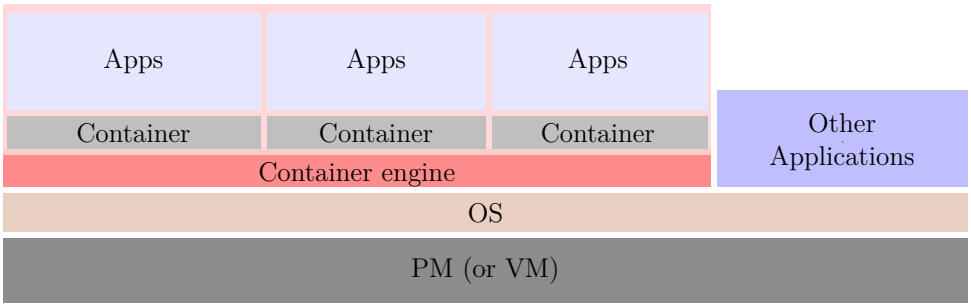


Figure 2.3 – Container system

2.4 Paradigm for infrastructure management

There are many paradigms that aim to manage multi-cluster or geo-distributed infrastructures [63, 65, 90, 108, 120]. In this section, two main paradigms for managing multi-cluster or geographically distributed infrastructures are presented. These two paradigms are the main management paradigms typically used for the execution of scientific workflows.

2.4.1 Grid computing

Grid computing [63, 65] is a paradigm for the management of a multi-cluster or a geo-distributed infrastructure. In Grid computing, multiple users have access to resources via a reservation system. In this thesis, we consider that the resources that are made accessible are PMs, even if the Grid paradigm can also give access to other resources such as network elements (routers for example), disks, etc. In this document, we will use Definition 2.4.1. This definition summarize the information presented in [62, 64].

Definition 2.4.1. "Grid computing is a model for enabling network access to heterogeneous hardware resources (*e.g.*, networks, servers, storage) to multiple users for different usages and purposes."

Grid computing uses a *middleware*, that manage the resources, and allows to hide the complexity of the infrastructure when executing distributed applications. The middleware can be seen as the bridge between the hardware and the software, and is in a way the OS of a multi-cluster infrastructure. This middleware relies on the OS running on the different PMs. The middleware is installed by the Grid computing provider and is accessible by the users, in order to manage the resources and let the users make reservations.

The middleware *gLite* [9], is an example of widely used and lightweight middleware for Grid computing. It allows to retrieve the list of available resources, and to submit jobs (basically an application) on a specific resource, as well as retrieve the list of running jobs.

2.4.2 Cloud computing

Cloud computing is a paradigm for the management of a physical infrastructure, that can also be used to execute scientific workflows, as we will see in Section 3.1. There are multiple ways to define Cloud computing [125]. In this thesis we will use the Definition 2.4.2 that is presented by the *National institute of Standards and Technology* of the U.S. Department of Commerce [92].

Definition 2.4.2. "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

The Cloud computing is provided in different kind of services by a *Cloud provider*. For example Google [12], Amazon [2], Microsoft [16] and IBM [14] are Cloud providers. A Cloud provider is defined by the NIST institute in [87] according to the Definition 2.4.3.

Definition 2.4.3. "A Cloud provider is a person, an organization; it is the entity responsible for making a service available to interested parties. A Cloud Provider acquires and manages the computing infrastructure required for providing the services, runs the cloud software that provides the services, and makes arrangement to deliver the cloud services to the Cloud Consumers through network access."

The definition 2.4.2 introduces the main concept of Cloud computing, namely the concept of *on demand* resource provisioning. A user can request access to new computing or storage resources at any time, using a simple protocol. This resource is made available by a Cloud computing provider, which typically uses virtualization mechanisms to reserve a portion of its infrastructure, which can be easily moved or released. The notion that allows resources to be reserved and released quickly is known as elasticity. This allows the user who needs access to new resources to not have to worry about managing the physical infrastructure, and moreover, to have access to more or less resources depending on the workload evolution (quantity of resources used by the application at a given point of time), thus reducing the cost of application execution. The term resource provisioning differs

from a standard reservation, as we can have in a Grid computing infrastructure. Indeed, the resource is not only reserved when provisioned, but is also configured, and unlike a resource reservation, this resource is generally virtual, and not attached to hardware resources. This is up to the Cloud provider to make the link between the provisioned resources and the hardware infrastructure. Further explanation of resource provisioning will be presented in the section 2.4.3.

Cloud infrastructures

Usually Cloud providers handle one or more *data centers*, that are distributed across multiple sites and regions containing multi-cluster infrastructures. The physical infrastructure is generally accessible only by the Cloud provider that is managing it. The resources are made available to the users via different services, that rely on virtualization mechanisms.

They are multiple types of Cloud computing environments. According to [92] there are four types of Cloud environments, to which we can add another type (*Federated Cloud*):

— *Private Cloud.* A private Cloud environment is installed and maintained for the use of a single individual or entity (*e.g.*, a company). The users, who can make resource provisioning requests, are part of the entity that maintains the environment.

— *Public Cloud.* A Public Cloud is made available for general public use. Anyone can reserve resources, which are usually rented at a certain price. Examples of Public Cloud are Google Cloud [12], Amazon AWS [2], and so on.

— *Community Cloud.* A Community Cloud can be seen as a bridge between a private and a public Cloud. It is a Cloud used by a community of users who share the same problematic but do not belong to the same entity. A Community Cloud can be managed and owned by several organizations.

— *Hybrid Cloud.* An Hybrid Cloud is the combination of two or more types of Cloud environment (private, public or community). The different Cloud environments are linked by a set of software and protocols that allow data or applications to be exported from one environment to another. For example, this could be used to extend a private Cloud with limited resources by a public Cloud environment.

— *Federated Cloud.* As with the Hybrid Cloud, a Federated Cloud is the combination of several Cloud environments, but this time of the same type. For example, it can

be used to exploit multiple Cloud offers for the deployment of applications across several countries, taking into account geographical issues (laws, taxes, etc.).

2.4.3 Cloud service models

There are four main types of Cloud computing service models that can be used to execute a user defined code and function, and by extension scientific workflows: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), Function-as-a-Service (FaaS) and Software-as-a-Service (SaaS) [49, 92].

IaaS - Infrastructure as a Service

The Infrastructure as a Service (IaaS) model, installed on an infrastructure by a Cloud provider, aims at offering computing infrastructure to the user. Generally those infrastructure consist of a set of virtual machines communicating through a virtual network, thus emulating a cluster of machines. The Cloud provider is in charge of the physical resources that are hosting the virtual ones. Furthermore, the Cloud provider is in charge of the OS that is installed on the VMs, and generally offers an access to different OSs. The user chooses the OS that has to be installed on the VM when requesting for a new virtual resource. Except for the OS, it is up to the clients to manage the software that will be installed on the virtual resources they have requested. In public Cloud, each virtual machines provisioned for the user will be charged to a specific price depending on the Cloud provider policy, and on the type of virtual machine. In the Table 2.1, is presented some examples of virtual machines that can be provisioned from the Amazon Cloud Provider [2]. For example, a user could rent four instances of *t3.micro* with 2 VCPUs and 1GB of RAM, and pay 0.0118$ per hour and per server, totaling 0.0472$ per hour.

Machine type	Virtual CPUs	Memory	Pricing (USD)
t3.nano	2	0.5 GiB	$0.0059 per Hour
t3.micro	2	1 GiB	$0.0118 per Hour
t3.small	2	2 GiB	$0.0236 per Hour
t3.medium	2	4 GiB	$0.0472 per Hour
t3.large	2	8 GiB	$0.0944 per Hour
t3.xlarge	4	16 GiB	$0.1888 per Hour
t3.2xlarge	8	32 GiB	$0.3776 per Hour

Table 2.1 – Part of the catalog of the IaaS service of Amazon EC2 - April 2020 - `https://aws.amazon.com/ec2/pricing/on-demand/`

PaaS - Platform as a Service

In Platform as a Service (PaaS) model, Cloud providers deploy a computing infrastructure as in an IaaS, but also install softwares that will be used to execute a specific task. For example, PaaS can be used to execute Big data applications such as Hadoop [6] or Cassandra [5]. The Cloud provider is also in charge of the resource elasticity, letting the users concentrate on the development of their applications (as in a FaaS). Two examples of PaaS are HDInsight [17], and Google App Engine [10]. PaaS services are widely used in Big Data applications.

FaaS - Function as a Service

The Function as a Service (FaaS) model is a Cloud service that aims at providing serverless computing, where the Cloud provider manages the allocation of compute resources in order to execute the functions of a user. Unlike the IaaS, the user does not have to manage instances of virtual resources on its own, but will submit functions to be executed by the Cloud provider. The pricing is generally based on the amount and size of the resources used by the functions of a user, at each second of the execution, rather than pre-purchased units of compute capacity. In order to be efficient, and execute the function as early as possible after the submission, the Cloud provider uses containers [34] rather than virtual machines, as they provide the ability to be provisioned in milliseconds instead of some seconds or even minutes [96, 97]. Google cloud function [11] and Amazon lambda [1] are examples of public FaaS services.

SaaS - Software as a Service

In the Software as a Service (SaaS) model, the cloud provider gives users access to an application running on its infrastructure. Typically, users connect to this application over the Internet, usually using a web browser. Examples of SaaS applications include email clients, online calendars or online document editors. SaaS is not discussed in this thesis.

2.5 Scheduling of scientific workflows

In this section, the *scheduling* problem is presented. This problem must be resolved in order to execute workflows in a distributed environment. Before explaining how the

resolution of this problem is related to scientific workflows execution, we will present it in general cases.

2.5.1 Scheduling simple tasks

A scheduling problem, has to be resolved in an OS, in order to execute the different applications running on a PM. In operating system, applications are divided into simple tasks, named threads, which define the sequence of instructions to perform. A CPU core can execute only one thread at a time. The scheduler is therefore an algorithm that is assigning to each CPU core a thread, and which is launched periodically to update the assignments, and execute the threads that have not been assigned and have been placed in a waiting queue (when there are more threads than CPU cores). If the scheduler is launched only once, and makes only one assignment decision, there is no temporal dimension to the problem and therefore the problem solved is a *placement* problem.

In the context of scheduling in an OS, we can distinguish two types of schedulers:

— *Non-preemptive*: When a thread is associated to a core, the scheduler waits the end of the thread before associating another thread to the core. In the case of an infinite thread, a core will be indefinitely lost for the execution of other tasks.

— *Preemptive*: The scheduler is allowed to stop a thread under execution on a core to place another thread. This implies that the operating system is able to stop a thread temporarily and resume it from where it stopped.

The most basic algorithm that can be implemented is the FIFO (First In First Out). As it stands, the first task that is submitted will be executed first and the last task submitted will be executed last. Figure 2.4 presents an example of scheduling with this algorithm, with one core and three threads. In this example, the length of the tasks is represented for clarity, but is not known by the scheduler. We can see in that figure that the algorithm is not good when long tasks are submitted first as potentially small tasks will have to wait the end of those tasks, this is known as the convoy effect [40]. We can also observe that this algorithm is non-preemptive.

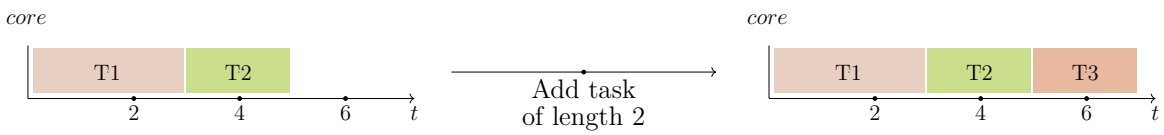


Figure 2.4 – FIFO algorithm example

A second algorithm that can be used in an operating system is the Round-Robin [82]. This scheduler is preemptive. The idea is that the scheduler assigns a thread to a core during a certain amount of time (called a time slice), and then switches to the next task in the queue. It repeats this operation until all the tasks are completed. An example of the round robin algorithm is presented in the Figure 2.5, with one core and three threads for a time slice of one second.

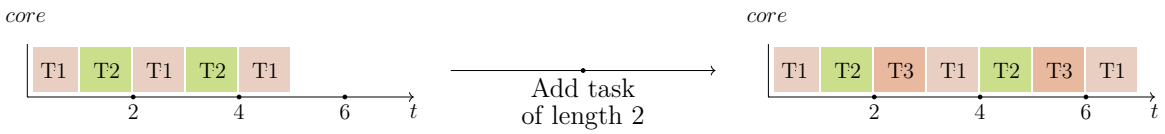


Figure 2.5 – Round-Robin algorithm example

2.5.2 Scheduling heterogeneous applications

In many cases, the applications to execute are heterogeneous, that is to say they require different amount of resources to be properly executed. The number of resources required by an application can be presented as the number of threads composing the application. The threads being dependent (operating on the same memory segments) have to be executed on the same PM. Other requirements can be considered for an application, like the quantity of required memory, the quantity of disk space, etc. Each one of those resource requirements (of one application) cannot be fulfilled by splitting the demand over multiple PMs. With the increasing complexity of the infrastructure that is composed of multiple machines, and with the increasing requirements for resources by concurrent applications, the problem has gain a new dimension, and became a space problem. This problem is known as the bin-packing problem, and is defined according to Definition 2.5.1 [67].

Definition 2.5.1. Bin packing problem:
Let $\mathcal{I}$ be a finite set of items of size $|I| \in \mathbb{N}$, and let $s(i) \in \mathbb{N}$ be a size for each item $i \in \mathcal{I}$, and $\mathcal{V} \in \mathbb{N}$ be a bin capacity. The problem is to answer the question: Is there a partition $\mathcal{B} = \mathcal{B}_1 \dots \mathcal{B}_K$ of size $K \in \mathbb{N}$ of disjoint subsets of $\mathcal{I}$, such that $\forall j \in \{1, \dots, K\}(\sum_{i \in \mathcal{B}_j} s(i)) \leq \mathcal{V}$?

Generally, the research is more interested in the optimization variant of this problem, which is to find the smallest possible value of K. For sake of simplicity in the rest of this thesis the term *bin packing problem* will refer to the optimization variant. Each element of

the partition $\mathcal{B}$ is called a bin, and in the case of placement in a distributed infrastructure refers to the PMs. It may be possible that there is no solution to the bin packing problem, and that all the applications don't fit in the bins at the same time, the number of bins being finite in the case of a multi-cluster infrastructure, as each bin is a PM. To solve that problem and still execute all the application, the bin packing problem can be resolved periodically - as a scheduler of an operating system - to adjust the application assignments, assuming that the execution time of the applications are finite even if they are unknown. Here, one can note a distinction between the *scheduling* and the *placement* problem. The bin packing problem when resolved only at a given instant is a placement problem, when the scheduling problem also has temporal consideration. By executing periodically the resolution of the bin-packing problem, and thus taking into account newly submitted and finished tasks, the problem that is resolved can be seen as a scheduling problem.

The bin-packing problem is known to be NP-hard [51,67], and therefore heuristics have been developed to resolve it. The first heuristic that can be used is First-Fit [102, 139], which is known to be a good heuristic to compute a relatively good solution in reasonable amount of time. This heuristic chooses the first bin that can contain the task. Another heuristic close to First-Fit is Best-Fit [80], this heuristic sorts the bins by usage each time a task is assigned to a bin. Therefore, the tasks will be assigned to the bin that is the more used at each turn.

2.5.3 Scheduling with temporal knowledge

We have seen in the previous section the problem of scheduling tasks, with unknown execution time, consuming only one resource at a time. Then we have seen the scheduling problem in which each task also has unknown execution time, but is consuming different amount of resources. In this section, we will present the problem of scheduling when the execution time of the tasks is known.

Definition 2.5.2. Job shop scheduling problem :
Let $\mathcal{J}$ be a set of jobs with different processing time, and $\mathcal{M}$ a set of machines with different computing capacities, where $\forall j \in \mathcal{J}$, and $\forall m \in \mathcal{M}$, $time_{j,m}$ is the time required by the job j to finish on the machine m. Assuming that a machine can run only one job at a time, define for each machine m the list $\mathcal{R}_m$ as a list of jobs, such as all jobs of $j \in \mathcal{J}$ appears once and only once in the set $\bigcup_{m \in \mathcal{M}} \mathcal{R}_m$, and such as $\max_{m \in \mathcal{M}} \left(\sum_{j \in \mathcal{R}_m} time_{j,m} \right)$ is minimal.

The scheduling problem, better known as *Job Shop Scheduling*, is an optimization problem, that can be retrieved in different context. This problem is known to be NP-hard [118], and its most common and simplest version can be formalized by the Definition 2.5.2 [44].

Assuming that each task is consuming exactly one resource, this problem can be seen as an extension of the problem seen in the section 2.5.1, where each task has to be assigned to a core. The main difference being that to each task is associated a processing time. This enables the possibility to make prediction, and therefore schedule tasks in the future, and create a scheduling plan. A scheduling plan or *planning* assigns to each task a resource and a start instant.

In this context, we can distinguish two types of scheduler :

— *Static*: the scheduler generates a planning that will not change even if new information arrives (*i.e.*, new task submissions). This type of scheduler can be launch periodically, and creates a planning for a certain period of time, for example creates the planning of the next hour.

— *Dynamic*: the scheduler generates a planning that can be updated when new information arrives. This should not be confused with preemptive scheduler. Indeed, it is possible to design a non-preemptive dynamic scheduler that is reconsidering the planning only for the tasks that are not already running. Obviously, it is also possible to design a dynamic scheduler that is preemptive.

2.5.4 Workflow scheduling

A scientific workflow can be seen as any application that is consuming resources. In that case, its topology can be totally ignored. In order to consider a workflow as a single application, its resource requirements has to be adapted to its high peak charge, depending on the number of tasks that can be executed in parallel at each step of the workflow. For example in a Cloud IaaS services a VM with sufficient resources to run the whole workflow would be provisioned, and therefore this VM can be considered as the one task that has to be scheduled.

By not knowing the topology of the workflow, we might also assume that the execution time of the workflow is unknown. And therefore, we can consider the bin packing problem as defined in Definition 2.5.1, where the items are the VMs executing the workflows, and the bins being the PMs of the infrastructure. When the workflow execution is not using the totality of the resources reserved by the VM, resources are wasted. An example of

this phenomenon is presented in Figure 2.6. In this figure the y-axis models an imprecise vision of the load on the infrastructure as the percentage of used resources, over the time.

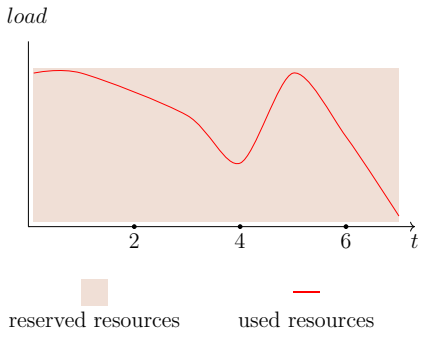


Figure 2.6 – Wasted resources from non-elastic reservation

In our context, we assume that the workflow topology is known and that it is therefore possible to place the tasks composing it, one by one. This allows resources to be used more efficiently and to take advantage of the elasticity of the executing environment. In addition, some algorithms used to schedule scientific workflows consider execution time of each task, in order to enhance optimization according to different criteria. The section 3.1 will present the related work around the scheduling of scientific workflows in details.

2.6 Energy efficiency and consolidation

Both Cloud and Grid computing environments are based on the same physical infrastructure which is a set of cluster of physical machines, regrouped in location that one can name datacenters or multi-clusters.

The energy efficiency of datacenters has become really important in the recent years as it raises economic and environmental concerns. It is known that Cloud datacenters have a significant environmental impact [52,105,115] that increases year after year. For example, in 2005, the total data center energy consumption located in the U.S. was responsible of 1% of the total US power consumption [91]. The consumption increased by 24% from 2005 to 2010, and about 4% from 2010 to 2014. In 2014, the energy consumption of datacenters in the US was estimated to 70 billion kWh, representing 1.8% of the total U.S. electricity consumption. According to [55], a datacenter may consume as much energy as 25,000 households. In this section, we will discuss the energy consumption of the datacenters during their usage.

In this thesis, the terms *power* and *energy* will be used. Although they are generally used interchangeably in everyday language, these two terms have very different meanings. Let us recall the definitions of these two terms. Energy consumption denoted E represents the amount of energy consumed during a given period of time. Its unit is the joule denoted J. Power consumption denoted P refers to the instantaneous power consumed at a given point of time. Its unit is the watt denoted W, which is equal to one joule per second. The energy consumption is the sum of the power consumption during a given period of time between T_{start} and T_{end}, as presented in Equation 2.1, where $P(t)$ represents the power consumption at the instant t. The unit kilowatt-hour (kWh) is generally the most commonly used billing unit for the delivery of electricity, and also represents energy consumption, and is equal to 3600 kilojoules (kJ).

$$E = \sum_{t=T_{start}}^{T_{end}} P(t) \tag{2.1}$$

2.6.1 Data center energy consumption

As presented in the section 2.4.2, Cloud infrastructures are composed of IT equipment such as physical machines, and network devices used to connect them. IT equipment are powered by electricity and thus are consuming energy. The energy consumption of a datacenter is divided in two elements, the consumption of the IT equipment, and the consumption of the air conditioning used to maintain the temperature of the equipment that is generating heat and has to be cooled in order to continue to operate correctly [98]. Figure 2.7 represents the percentage of energy consumption of air conditioning and IT equipment in a datacenter according to [98].

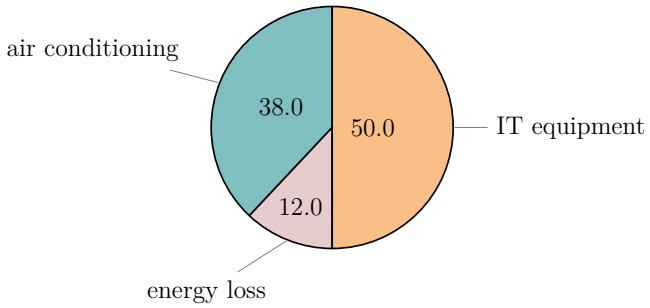


Figure 2.7 – Average energy consumption distribution of 94 datacenters from 2006 to 2015 [98]

We can see in this figure, that almost 50% of the energy consumption is used to main-

tain the datacenter at a reasonable temperature and is therefore not used as computing power. The energy consumption of the IT equipment can be divided into two different elements, the consumption of the physical machine and the consumption of the network devices, representing respectively, according to [83], 70% and 30% of the consumption of the IT equipment. The consumption of the network however is not considered in the scope of this thesis.

2.6.2 Physical machine energy consumption

Physical machines represents about 35% of the total energy consumption of a datacenters [83]. In this section, we are detailing the different elements that consume energy inside a physical machine.

The power consumption of a physical machine is the sum of the static power, that we denote P_{static}, and the dynamic power consumption $P_{dynamic}$. P_{static} is the consumption of a physical machine, only used to maintain the system wake up, we also denote this consumption P_{idle} referring to the consumption of a physical machine when it is not used. On the other hand, the consumption $P_{dynamic}$ is the power consumption of the physical machine that added to P_{idle} is equal to the power consumption of the physical machine when used to perform operations. $P_{dynamic}$ depends on the utilization of each hardware component, that can be expressed as a percentage (*e.g.*, 25% of CPU when using only one of four available CPU). When all the resources of a physical machine are used at their maximum, the power consumption reaches the value P_{max}. The figure 2.8 represents the power consumption of the different components of a physical machine according to [83].

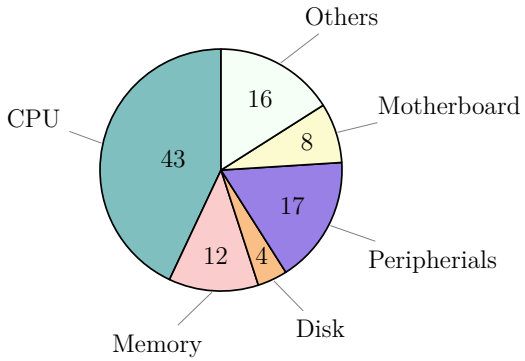


Figure 2.8 – Distribution of the power consumption of a physical machine [83]

According to [29] the collection of data of load usage of more than 5000 productions

PMs shows that PMs are generally under-used as their utilization rates reached only 10% to 50% of their full capacity most of the time. However, the power consumption of these PMs was about 70% in average of their P_{max}, even when under-used. This phenomenon is due to the poor power efficiency of a PM, as the P_{idle} consumption rarely represents less than 50% of P_{max}. Moreover, in recent years with the development of multi-core processors, the power consumption of PMs can no longer be represented by a linear function [43, 74, 135], which implies that a PM is more energy efficient at high load than at low load. Hence, parallel execution of tasks is more energy efficient than sequential execution on a PM. Figure 2.9 represents the energy consumption measured on a PM of the ecotype cluster of the Grid'5000 [13] testbed using the SeDuCe platform [103]. The other curve represents the energy consumption of an ideal PM, where the energy consumption is totally linear to the load, and $P_{idle} = 0$.

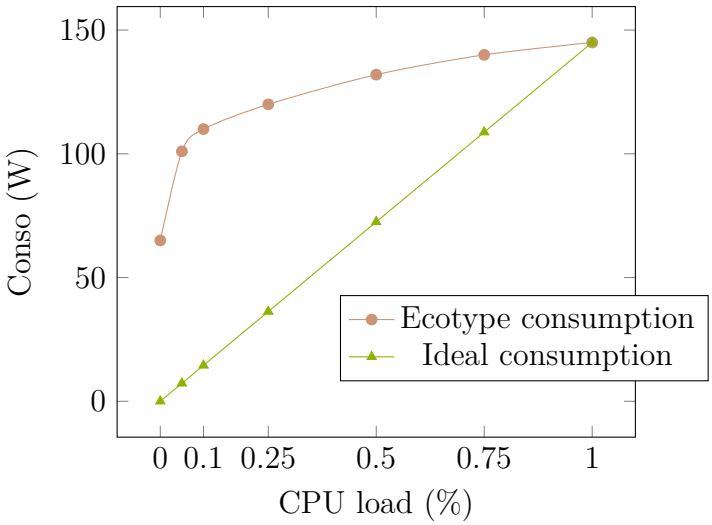


Figure 2.9 – Power consumption of one ecotype machine with a variable CPU load

It can be noted from Figure 2.9 that, since CPU consumption is not linear to the load, when using multiple PMs of a cluster, it is more likely that using one PM that is 100% used is less consuming than using two PMs that are 50% used, (for example for two Ecotype PMs used at 50% the consumption would be $132 + 132 = 264W$, when with only one PM used at 100% and the other at 0%, the consumption would be $145 + 65 = 210W$).

In order to measure the power consumption, the physical machines are generally connected to power distribution units (PDUs), that are plugged between the physical machine and the wall socket [104], and that contain watt-meters.

2.6.3 Energy saving techniques

In this section are presented some methods to save energy within a multi-cluster infrastructure when considering only PMs. Those techniques can be used in Grid computing environments as well as in Cloud computing environments, as the main difference between those two environments lies on the management of the infrastructure and not on the infrastructure itself.

Shutdown technique

Unused physical machines consume a large amount of energy, as their idle power consumption P_{idle} can represent 50% of their maximum power consumption P_{max}. This consumption can be drastically reduced by shutting down the physical machine when they are not used, and thus reducing the total energy consumption of the datacenter. This reduction of energy is due to both the idle consumption of a powered on unused PM, and the cooling system. This technique was first investigated by [50] in 2001. However, the transition between the on and off states of the server cannot be ignored [32, 100] as it may consume both energy and time.

Defining when to stop a physical machine is therefore a difficult optimization problem for two main reasons. First, the number of physical machines at a given time must be sufficient to meet the demand for computing power, as powering on a physical machine is not instantaneous. Second, when a decision to shut down a machine is made, the machine must remain off for a certain period of time to compensate the energy consumption of the off phase and the on phase, and to ensure that the sum of these two energy consumptions will be less than the energy consumption that would have resulted if the machine had remained in idle.

DVFS - Dynamic Voltage and Frequency Scaling

Many hardware components allows to control the voltage through software. The idea of the DVFS energy lever is that reducing the voltage of a component (for example the CPU), will reduce the energy consumption of a PM. This will equally lead to a reduction of the temperature of the components, and also may reduce the energy consumption required by the cooling system. This reduction of the voltage however has an impact on the efficiency of the components, and therefore on the efficiency of execution of the applications running on a PM.

Consolidation - placement algorithm

As already seen the consumption of the physical machines is not proportional to their load. The idea of the consolidation technique is to regroup the resource demand on the same physical machine, in order to use a less number of machine. This can be combined with shutdown techniques, in order to select a certain number of awake physical machines and shutdown the rest of them.

The problem of consolidation can also be seen as a bin-packing problem (See Definition 2.5.1). It has mainly be investigated in the Cloud IaaS service [53,66,85,99], in order to regroup the VMs on the same physical machines, and use as few number of physical machines as possible. Even, if energy saving is not always the main concern when trying to solve the bin-packing for VM placement [113], this technique can lead to energy saving as the used physical machines are used more efficiently and the unused ones could be turned off.

2.7 Conclusion

In this chapter we have presented the concept of scientific workflows, as well as the environments that may be used to efficiently execute them. We have presented the main challenges that are raised when targeting the execution of the workflows in a distributed environment. We have also seen that distributed infrastructure are highly energy consuming, and that there are techniques that may be used to reduce the energy consumption. In the following chapter, we will present the related work around the scheduling and the execution of scientific workflows, in the Grid computing and the Cloud computing. This related work will allow us to highlight some of the remaining challenges, we aimed at solving in the contributions of this thesis.

Chapter 3

Related work

In this chapter, we discuss the related work that has been carried out around the scheduling and the execution of scientific workflows. This chapter is divided into two main sections, the first one presents the noteworthy works that have been conducted on scientific workflow scheduling. The second section presents the works that have been carried out around the execution of scientific workflows in a distributed environment.

Contents

3.1 Scientific workflow scheduling

As discussed in section 2.4, workflows can be scheduled and executed on various types of environments. In the literature, we can observe two main targeted environments. On the one hand, the Grid computing environment, and on the other hand the Cloud computing environment. This section presents some interesting works related to the scheduling of scientific workflows. This presentation will be divided into five parts, the first part presents the different criteria that we aim to take into consideration in this thesis, the second part presents the different classes of algorithms that can be retrieved from the literature. Then Sections 3.1.3 and 3.1.4 offer an analysis of respectively, the algorithms dedicated to the Grid computing, and those dedicated to the Cloud computing, that can be found in the

literature. This analysis uses the criteria and the classification of algorithms previously presented. Finally the last section discusses the limitations of the state of the art solutions to meet the criteria of this thesis.

3.1.1 Criteria of interest

This section presents the criteria to be considered in the contributions of this thesis. In our context, multiple users submit workflows to a Cloud computing environment for execution. This Cloud environment is managed by a Cloud provider, who has access to the management of the PMs that make up the underlying infrastructure. In this thesis, we assume that the tasks composing the workflows are very heterogeneous as presented in the section 2.1. We can list our criteria as follows:

Consideration of task heterogeneity. As previously mentioned, the tasks composing workflows are very heterogeneous and may require the use of different operating systems to be properly executed. They are also heterogeneous in terms of hardware resource demand (quantity of memory, number of CPU cores, ...).

Isolation. Workflows that are submitted belong to several users. Therefore, a strong isolation must be guaranteed for security reasons.

Energy optimization. As presented in Section 2.6, datacenters and multi-cluster infrastructures have a huge environmental impact. In this thesis, we are interested in reducing the energy consumption of these infrastructures.

Submission by multiple users and Fairness. Since many users will be sharing the same physical resources for the execution of their workflows, it is important that the service provider provides a good quality of service. We argue that fair resource sharing is one way to achieve this goal.

Uncertainty of execution and submissions. In this thesis, the term uncertainty will refer to temporal information for which it is impossible to be completely certain. Two types of uncertainty can be distinguished. On the one hand the partial uncertainty, in this case, the information on the necessary time or the temporal instant is partially known (*e.g.*, the time needed to perform a task), but may vary. On the other hand, the total uncertainty, in this case, no information is provided (*e.g.*, a new submission). It seems important to take both type of uncertainty into account in the scheduling algorithm in

order to properly manage the environment and be able to continue to consider the other criteria (*e.g.*, energy and fairness).

3.1.2 Algorithm classification

Before presenting the objectives (*i.e.*, the problem they aim at solving and the metrics they aim at optimizing) of the algorithms available in the literature, and discussing their differences with our criteria, we present the different classes of scheduling algorithms. In the state of the art, we can find different types of algorithms, that aim at solving the scientific workflow scheduling problem. These algorithms can be divided into three main classes, *heuristic-based*, *meta-heuristic-based* and *exhaustive search*.

The algorithms that are presented in the following subsections are used to solve scheduling problems. In the case of complex problems, such as the scheduling problem, there can exist multiple solutions, however those are not all equal. The optimization of an objective, refers to the selection of the solution to the problem that gives the best result based on the objective (*e.g.*, with an energy minimization objective, the best solution would be the solution that leads to a minimal energy consumption). The scheduling problem can be a single-objective problem or a multi-objective problem. A single-objective problem is an optimization problem, in which only one objective (*e.g.*, execution time minimization) needs to be optimized. Conversely, a multi-objective problem has several objectives to optimize, and these objectives are contradictory (such as minimizing energy consumption and the execution time). Indeed, if two objectives can be optimized the same way, the problem can be reduced to a single-objective problem. A multi-objective optimization problem may have several optimal solutions. A solution called optimal, is a solution to a multi-objective problem where no single objective can be improved without deteriorating other objectives, such solution is also called a Pareto optimality. A Pareto frontier is the set of Pareto optimality solutions.

Heuristic based algorithm

Heuristic-based algorithms are algorithms that decide at each step which branch to follow in the search space in order to compute a unique solution. The search space is the set of domain through wich an algorithm performs a search (by affecting values to the variables). In a heuristic-based algorithm, the decision is made locally, with the only information available at a given point. Heuristic-based algorithms are used to find a good

solution in a short amount of time. As the scheduling of scientific workflows is a complex problem known to be NP-hard [118], heuristics-based algorithms have been widely studied in the literature, in order to find a solution within a reasonable amount of time. We can list in the literature three different types of heuristic algorithms in the context of scientific workflow scheduling :

— *Independent tasks scheduling*: In this type of algorithm, the tasks with no dependencies are scheduled first and removed from the topology of the workflow, this creates new independent tasks. This operation is repeated until all the tasks of the workflow are scheduled. The following contributions present an algorithm based on an *independent task scheduling* heuristic: [130], [89], [22], [24], [21], [88], [76].

— *List scheduling*: This class contains algorithms creating a list of tasks sorted in a given order. Once the list of tasks have been sorted, the scheduling per se is performed, one task after the other. The sorting is made according to an heuristic function, that tries to sort the tasks in an order that may lead to the optimization of a given criteria. The following contributions present an algorithm based on a *list scheduling* heuristic: [122], [25], [60], [116], [68], [72], [133].

— *Iterative scheduling*: The *iterative scheduling* algorithms create a first scheduling plan with a valid solution containing all workflow tasks. It is then updated iteratively to improve the solution and objectives. In [132], an iterative scheduling algorithm is presented.

Meta-heuristic based algorithm

Some research has been conducted around algorithms based on meta-heuristics. Meta-heuristics are general algorithms often inspired by nature, which are designed to solve optimization problems. In the literature, the main meta-heuristic used for the scheduling of scientific workflows is the genetic algorithm or GA [119]. Genetic algorithms are meta-heuristics inspired by the process of natural selection, and are based on operations such as mutation, crossing and selection, which are applied to a set of individuals. Each individual represents a solution, which can be mutated and combined with other solutions. At each step of the algorithm's execution, only the best individuals are selected, and can move on to the next generation. The algorithm may not converge, and must be stopped by the user or when a given criterion is met. Genetic algorithms can be used to solve a multi-objective problem, and compute a Pareto frontier. In the literature, the following contributions use

GA meta-heuristics to solve the problem of scientific workflow scheduling: [73], [131], [93], [84], [58], [26].

Exhaustive search based algorithms

When it comes to an optimization problem known to be NP-hard, such as scheduling scientific workflows, the only known way to find the optimal solution is to perform an exhaustive search and browse through all branches of the search space. In the literature, there is not much research around the exhaustive search for scientific workflow scheduling, because the problem is very complex, and the solution computed by such an algorithm would be obsolete once computed, due to the important time required for the computation. For this reason, the researches that have been conducted, generally only propose partially exhaustive search. In [60], the authors propose an algorithm that performs a semi-exhaustive search, which can be configured, by defining the number of branches that will be browsed in the search space. As we have seen, [60] is based on a heuristic algorithm *list scheduling*, but instead of computing a unique solution, the presented algorithm computes a set of K solutions, keeping traces of the best K solutions at each step. The higher the value of K, the wider the range of branches explored, and with a sufficiently high K, the search becomes fully exhaustive.

Backtracking is a general algorithm used to find all the solutions of a given problem. The idea of backtracking is to fully explore a branch, and return to the previous partial solution (branch not fully explored), when a solution is found or a dead end is encountered (a dead end being a branch of the search space that does not lead to a solution). This algorithm makes it possible to explore the search space, by creating a tree of all the partial solutions already explored. Unlike the exhaustive algorithm presented in [60], a backtracking algorithm explores only one branch at a time. The difference between these two algorithms is therefore based on the way of exploring the tree of the search space, the backtracking algorithm performing a depth-first search, and the algorithm presented in [60] performing a breadth-first search. The backtracking algorithm does not necessarily traverse the entire search space and can be stopped as soon as a solution is found, in order to reduce the time required for the computation of the solution. The algorithm presented in [22] and [21] uses a backtracking algorithm in order to find the first solution that meets a given criteria.

Branch and Bound is an evolution of the backtracking algorithm. This algorithm computes the bound of the best possible solution of a given branch of the search space

before exploring it. This way, if the bound is lower than the current best solution found in the search space, the branch is ignored, and the exploration can be made faster. In [117], the branch and bound algorithm is used to resolve the problem of scheduling of a scientific workflow.

Static and Dynamic scheduling

Most of the algorithms presented in the above subsections are static algorithms. This means that they are capable of computing a solution (a planning) for a given problem that will not change. However, there may be occasions when some elements of the problem are uncertain, such as the execution time of a task, a new submission, and so on. For this reason, some researchers have opted for dynamic or online algorithms. On the one hand, an online algorithm is an algorithm that computes the solution only for the elements about which there is no uncertainty. The algorithm is called periodically or when new events occur, to compute the rest of the solution. For example, the algorithms presented in [88] and in [25], schedule only ready tasks (independent tasks, whose parents are completed or in progress), so the scheduling can be adapted according to the event that occurs (*e.g.*, delayed task, new submission, etc.). On the other hand, a dynamic scheduler, as presented in section 2.5.3, generates a planning in advance, but is able to modify it as the problem evolves. For example, in [61], a dynamic algorithm is considered for solving the scheduling problem.

3.1.3 Scheduling on Grid computing

The literature has shown interest in scheduling scientific workflows within a Grid computing environment. As presented in the context chapter, Grid computing is a paradigm for the management of a multi-cluster infrastructure. In this type of environment, resources (here PM or processors) can be reserved in order to perform tasks. In a Grid computing environment, the reserved resources do not have a virtualization layer, so tasks are executed on the PMs directly using the operating system of the PM.

Scheduling for one user

The algorithms presented in this section are used to schedule a single workflow belonging to a single user. This workflow can be the fusion of several workflows, but never belongs to several users, thus this does not raise any fairness issue - the fairness defining

a fair sharing of the resources between multiple users. Furthermore, in the following related work, all the required tasks are assumed to be homogeneous and to consume only one resource, namely a processing unit (or a processor). As each task consume only one processing unit, it is not necessary to consider how the resources are distributed among the different PMs, there is only a need to consider the speed of communication between processors.

In the literature, algorithms developed for Grid infrastructures can be divided into three main categories depending on the objectives they aim at maximizing [138]: *Best-effort based algorithms*, *Cost-based algorithms*, and *Energy aware algorithms*. The following sub-sections are listing scheduling algorithms with those objectives.

A. Best-effort based algorithms

Best-effort based algorithms are used when the objective is to minimize the overall workflow execution time, or in other words to minimize the makespan of the workflow execution. Different algorithms have been developed to solve this scheduling problem, with the aim of minimizing the makespan. In the following, the word makespan will be used to refer to the overall execution time of the workflow.

[73] and [131] present algorithms that use GA techniques. The authors of these two contributions present their mutation, crossover and selection functions for their GA algorithm for scheduling interdependent tasks on a single PM containing several processors (CPU cores). As all tasks are executed on the same PM, the communication phases (transmission of a file from a task to its successors) required in distributed infrastructure is not taken into account in these works. Indeed, all the tasks have access to the same file system (for example the same hard drive), and thus the files are available to any tasks at any moment.

Algorithms based on heuristics have also been studied in the literature. First, *Independent task scheduling* algorithm have been adopted to solve the problem of scheduling the scientific workflow in a Grid environment. In [130] the MYOPIC algorithm is introduced, and has been implemented and used in Grid systems such as Condor DAGMan [121]. The problem solved by this algorithm is the creation of a planning for the execution of a scientific workflow, taking into account the size of the communication between tasks and the speed of communication between reservable processors. In addition, the fact that task execution time may differ from one processor to another, due to the heterogeneity of the hardware components, is also taken into account in this work.

In [89] MIN-MIN and MAX-MIN are presented. These algorithms are originally de-

signed for the scheduling of parallel independent tasks. However, they can be used for the scheduling of workflows (*Independent task scheduling*) as for MYOPIC. The MIN-MIN algorithm first retreives the independent tasks of the workflow, and schedules them, starting its decision process with the task with the lowest execution time (on average on the reservable processors). The MAX-MIN algorithm is similar to the MIN-MIN algorithm, except that it chooses to schedule the task taking the highest execution time first. These algorithms have been implemented and used in workflow engines for Grid computing [33, 41]. A detailed study is presented in [42] showing that MIN-MIN is in average better than MAX-MIN in terms of makespan minimization.

Algorithms based on *List scheduling* have also been used in the literature. The algorithm known as HEFT (Heterogeneous Early Finish Time) [122] is a well-known *List scheduling* heuristic based algorithm. This algorithm computes a rank for each task of the workflow. The rank of a task is based on the rank of its successors, its length (average time required to complete the task on all available processors), and the size of the file that must be passed by the task to its successor. This rank is used to sort the tasks, with the task having the highest rank being scheduled first. In [140] the authors conducted experiments to evaluate the rank function used in the HEFT algorithm, and showed that it is not always the best heuristic function according to the type of workflow that has to be scheduled. In [48, 114] MIN-MIN and HEFT have been compared for the scheduling of different workflows, and HEFT has been shown to be better than MIN-MIN in most cases in terms of makespan minimization.

In [117] a branch and bound algorithm is presented for the scheduling of a scientific workflow on a set of processors. This algorithm takes into consideration the communication time by considering a speed of communication between the processors, and the size of the dependencies. By using a branch and bound algorithm, the authors are able to compute the best solution with the minimal makespan.

B. Cost-based algorithms

The *Cost-based* algorithms add a new metric to the scheduling problem: the cost associated with resource reservation. This new metric creates a new objective, which is cost minimization. By adding a new objective to the problem, it becomes a multi-objective optimization problem, which is more difficult to solve [94]. In the literature, most of the time, one of the two objectives is replaced by a constraint that must be satisfied in the solution computed by the algorithm.

First, some algorithms have opted for a deadline constraint instead of a makespan

minimization to simplify the problem and transform it to a single-objective problem. An algorithm for scheduling a scientific workflow in a Grid environment is presented in [76]. This algorithm takes into account a deadline for the workflow, and attempts to ensure that the workflow makespan will not exceed its deadline while minimizing the cost of execution. This algorithm first groups the tasks into partitions. Each partition contains tasks that must be executed sequentially (due to dependency constraints). Once the partitions have been computed, a deadline is assigned to each partition. Each partition is then scheduled on the resource (processor) that gives the lowest cost while ensuring that the partition's sub-deadline is respected. Since the tasks in a partition must be executed by the same resource, they are assumed to be homogeneous.

In [22], the authors present a deadline constraint algorithm. Unlike [76], this algorithm does not create task partitions, but assigns a sub-deadline to each task in the workflow. The deadline for each task is computed from the partial critical path of the workflow. The critical path of a workflow is the longest execution path from the entry tasks to the exit tasks, and the partial critical path is the longest execution path from a given task to the exit tasks of a workflow. Once sub-deadlines are assigned, each task is scheduled one by one by selecting the resource (processor) providing an execution time that meets the deadline with the lowest cost.

Instead of a deadline constraint in order to remove the makespan minimization, some researchers have opted for a budget constraint. In [25], the authors define an algorithm that schedules a workflow in a Grid environment with makespan minimization objective, and a budget constraint. This algorithm is a *List scheduling* algorithm that sorts the tasks according to the HEFT rank function. Then, it schedules the tasks one by one, selecting the resource (processor) that gives the best trade off between cost and efficiency, while guaranteeing that the remaining budget is sufficient for the execution of the remaining tasks.

C. Energy aware algorithms

In the context of Grid computing, there is a sub-part of the literature that focuses on optimizing the energy consumption of the infrastructure that executes the scientific workflows. The works we present in this section carry out this optimization during the scheduling phase.

Most of the work relies on the use of DVFS technique to select the frequency of the processors. In [93], the authors use a genetic algorithm with multi-objective optimization, namely makespan minimization and energy optimization, creating a Pareto frontier for the

scheduling of a workflow on a set of processors. In [84], the authors present an algorithm also based on GA, which solves the same problem. These two algorithms do not allow to change the processor frequency during execution, and give a solution where a frequency is attached to each one of them.

In [60] two algorithms based on HEFT are introduced, for the scheduling of a workflow in a Grid environment. These algorithms do not use the DVFS technique, but compute an estimate of the consumption of each physical machine when a task is scheduled on them (on one of the processor of a PM, a PM being considered as a set of processors). This estimation of the power consumption is based on a non-linear model presented in the same paper. This function computes the energy consumption based on the number of processor used inside a PM (processor with assigned tasks) at each instant. From this estimation, the first algorithm MOHEFT (Multi-Objective Heterogeneous Early Finish Time) computes a Pareto frontier composed of multiple solutions, with the goal of minimizing the makespan and minimizing energy consumption. The second algorithm Green-HEFT chooses for each task the resource (processor) that consumes the least energy, instead of the resource that gives the best makespan, as HEFT would do. Indeed, as the energy consumption of a PM is not linear in relation to its load, the energy consumption induced by the execution of a task would not be the same, depending on its location. The algorithm Green-HEFT is only used by the authors to compute the lower bound of the energy consumption, but gives a bad solution in terms of makespan as using only one node will always consume less energy than using multiple.

Scheduling for multiple users

Some works on workflow scheduling in a Grid computing have helped to reinforce fairness among users. Fairness can be defined as a fair sharing of physical resources between several users, proportional to their needs. In [24], the algorithm shares a set of resources (here processors, because the heterogeneity of tasks is not taken into account) between several workflows, each workflow belonging to a user. The algorithm considers a priority on each task. The priority of a task is based on its number of successors (recursively) and its rank (computed with the rank function of HEFT). The algorithm schedules first the tasks that have the fewest remaining successors. The idea is to give the opportunity to the new submitted small workflows to be executed before the end of the workflows already in progress, and to avoid the convoy effect (see Section 2.5.1). The goal of this algorithm is to minimize the makespan of all the workflows.

In [61], the authors consider the submission of multiple workflows belonging to multiple users in a Grid environment. This algorithm creates groups of tasks that must be executed by the same resource (once again, a processor). The idea of this algorithm is to find a trade-off between parallelism and fairness between users. Indeed, the more tasks in a group, the less parallelism there is for a given workflow, but the more resources are available for the execution of other workflows. As for [24], the objective of this algorithm is the minimization of the makespan of the set of workflows being executed. The algorithm presented in [61] is a dynamic algorithm that reconsiders the computed planning when a new submission arrives, in order to maximize fairness between already scheduled and newly submitted tasks.

Discussion on scheduling algorithms on Grid computing

#	Class	Related work
Heuristic based algorithm		
1	*Independent task scheduling*	[130] [89] [76] [22] [24]
2	*List scheduling*	[122] [25] [60]
3	*Iterative scheduling*	-
Meta-Heuristic based algorithm		
4	*Genetic algorithm*	[73] [131] [93] [84]
Exhaustive search algorithm		
5	*Breadth-first search*	[60]
6	*Backtracking*	[22]
7	*Branch and Bound*	[117]
Dynamic algorithm		
8	*Online*	[25]
9	*Reconsidering*	[61]

Table 3.1 – Class of the algorithm of the related work for Grid computing environment

As one can notice, the majority of algorithms found in the literature in the context of Grid computing do not take into account task heterogeneity. They assume that each task consumes exactly one unit of resource, namely a processor, for a certain period of time. This definition of the problem is very close to the Job Shop scheduling problem 2.5.2, with the only difference that it takes into account the dependencies between tasks and the communication time between tasks. This limitation cannot be removed naively, as resources are spread over several PMs and tasks cannot be splited, and the problem must gain a new dimension, and become closer to a bin-packing problem 2.5.1.

The second limitation that could be observed regarding the heterogeneity of tasks is the lack of consideration for software dependencies required for a given task. As presented in section 2.1, the tasks of scientific workflows can be very heterogeneous and may require different operating systems and libraries. However, since the resources offered by the Grid computing environment cannot run a specific operating system required by the user, these types of workflows cannot be executed in such an environment.

Table 3.2 presents the criteria that we aim at taking into consideration in the contributions of this thesis. This table lists the contributions presented from the related work that take into account these criteria.

#	Metric	Related work
Task heterogneity consideration		
1	*Software*	-
2	*Hardware*	-
Isolation		
3	*Isolation*	-
Energy		
4	*Energy optimization*	[76] [60] [84]
Fairness		
5	*Multiple users and Fairness*	[24] [61]
Uncertainty		
6	*Execution time*	[24] [61]
7	*Submission arrivals*	[24] [61]

Table 3.2 – Criteria of interest of the contributions of this thesis taken into account by Grid scheduling algorithms

Table 3.1 presents the classes of algorithms used in the related work on Grid computing environment.

3.1.4 Scheduling on Cloud computing

Cloud computing is the environment targeted by most of the latest research in the literature for the execution of scientific workflows. As presented in section 2.4, Cloud Computing is a paradigm for managing a multi-cluster infrastructure. In this type of environment, resources can be reserved to perform tasks. Regarding the scheduling of scientific workflows, the Cloud service that has been the most used is the IaaS (*i.e.*, Infrastructure as a Service). Thus, the planning (solution of the scheduling problem) lists

the VM instances (*e.g.*, type, number of instances, duration of the reservation, etc.), and associate to each task the VM instance that will execute them, as well as their start time.

Scheduling one workflow

The algorithms presented in this section are used to schedule a single workflow belonging to a single user. This workflow can be the fusion of several workflows, but never belongs to several users. In general, the works that can be found in the literature focus on scheduling from the user's point of view, when the Cloud environment is operated by a third party (e.g. public Cloud providers such as AWS [2]). Thus, the physical infrastructure of Cloud computing is seen as an unknown black box, and it is generally assumed that the Cloud provider is capable of providing infinite resources, and therefore the only limit is the user's budget. As a result, decisions on the placement of different virtual resources are left to an internal scheduler controlled by the Cloud provider, and fairness is therefore not taken into account in these works, as they locate the decision on the user side.

Unlike schedulers for Grid computing, scientific workflow schedulers for Cloud computing environments, that can be found in the literature, never intend to minimize only the makespan (single-objective problem). This could be achieved relatively easily by launching as many virtual resources as needed to run the parallel parts of the workflow, but this would result in a relatively expensive use of resources for the user. As a result, as illustrated in [23], most publications feature scheduling algorithms based on cost optimization. In the context of Cloud Computing, three types of cost-based scheduling algorithms can be distinguished: *Budget constrained*, *Deadline constrained* and *Energy aware*.

A. Budget constrained algorithms

The goal of this type of algorithm is to minimize the makespan of the workflow execution, when the user is limited by a budget that must be respected. Indeed, the provisioning of each virtual resource is charged to the user by the Cloud provider.

In [48] a budget has to be met according to a public IaaS Cloud offer. In public Cloud offers, provisioned VM instances are typically billed on an hourly basis, which means that one VM used for two hours will cost the same as two VMs (of the same type) used for one hour, and that a VM used for one minute will cost the same as a VM used for one hour. In order to stay within the user's budget, the scheduler must be able to determine how many virtual resources to use at any given time, and when to release them knowing that the current hour is already paid. In [48] an algorithm based on HEFT is presented. This

algorithm allocates the budget between the different tasks of the workflow, while ensuring that all tasks have sufficient budget to be properly executed.

In [132] an algorithm that respects a user's budget by minimizing overall execution time is considered. The algorithm assigns to each task the cost and execution time on each VM type. Then, the algorithm assigns to each task the VM type that costs the least. The remaining budget (the user's budget minus the cost of each task) is then allocated to a task, modifying its allocation and putting it on a faster VM. At each reassignment, the task reassignment that gives the best gain is selected; this gain is evaluated by calculating the difference between the execution time of the critical path before and after the reassignment. The reassignment step is repeated until the remaining budget no longer allows a task to be reassigned. In this work, each VM is billed to the user, not on an hourly basis, but based on the time it takes to complete the task (a VM being used to execute only one task).

In [26] a genetic algorithm is presented for the scheduling of one workflow inside a Cloud computing environment. This algorithm initializes the population of the genetic algorithm based on the HEFT algorithm, by generating K random solution, and adding the solution computed by the HEFT algorithm. In this work, a solution is a planning composed of a set of VMs with associated tasks, each VM being able to execute only one task at a time. Then the selection phase of the individuals is based on the makespan and the cost of each solution. The algorithm is stopped, once a solution that meets the budget is found.

B. Deadline constrained algorithms

This type of algorithm introduces a deadline to the workflow and tries to ensure that the workflow execution will not exceed the deadline. The objective of this type of algorithm is to minimize the cost of execution for the user.

In [21] a deadline based algorithm for scheduling one workflow on IaaS Cloud is presented. The IC-PCPD2 (Iaas Cloud Partial Critical Path with Deadlines Distribution) algorithm, which was originally dedicated to the Grid infrastructure in [22], prioritizes the cheapest resource (cheapest VM type) while trying to meet the workflow deadline. In this work, the resources are VMs, and these VMs are capable of executing only one task at a time. As with the Grid version, this algorithm computes a deadline for each task, allowing it to determine the cheapest resource that can be provisioned to ensure that this deadline will not be exceeded.

In [116], an algorithm is presented for scheduling a scientific workflow with a deadline

constraint. The algorithm computes a sub-deadline for each task. Then, a pool of VMs is maintained, by varying the number of VMs it contains. Each VM holds a list of tasks that will be executed by it, where a VM can only execute one task at a time. The number of VMs remaining in the pool is computed by a function on the number of ready tasks, and the interest to launch or kill a VM instance (due to cost consideration). The goal of the algorithm is to minimize the client's budget while respecting the overall workflow deadline.

In [133] a *List scheduling* algorithm is presented for the scheduling of scientific workflow within heterogeneous VM. Unlike the algorithm that have been presented so far, the algorithm in [133] takes into account the heterogneity of the hardware capabilities of the VM type. Indeed, the authors consider that a VM is capable of executing as much task in parallel as it contains VCPUs. The algorithm starts by creating partition of tasks, each partition being the tasks that have to be executed sequentially (as for the IC-PCPD2 algorithm in [21]), and assign to each partition a deadline. It then, schedules the partition on the VMs by choosing the VM having the most number of idle (unused) VCPUs, in order to maximize the utilization rate of the provisioned VM instances, and minimize the number and the duration of the instantiated VMs, and thus the overall cost. As in [21], the tasks composing a partition are supposed homogeneous as they will be executed by the same resource (here a VCPU of a VM instance).

C. Energy aware algorithms

As with scheduling in the Grid computing environment, research has been conducted to minimize energy consumption. In these works, the minimization of the cost (or the user budget) is replaced by an objective of minimizing the energy consumption.

In [72], the authors propose an algorithm to optimize the energy consumption induced by the execution of a workflow within an IaaS Cloud environment. The proposed approach uses DVFS technique, associating a frequency to each provisioned VM. The algorithm first assigns a deadline to each task, allowing it to determine the VM instance that will be used. To each VM instance, the algorithm also assign a frequency, this frequency being the lowest possible frequency guaranteeing that the deadline of the task executed by the VM instance is met.

In [68], a placement policy is presented for executing scientific workflows within VMs on an infinite set of PMs, with the goal of minimizing power consumption while meeting a user-defined deadline. As specified, the proposed approach in [68] is a placement algorithm, and not a scheduling algorithm. Indeed, the authors assume that they have no information

about the time required by a task to be executed. Thus, as no temporal information is known by the algorithm, no prediction can be performed and thus the problem is a placement problem, and is really close to the bin packing problem (cf. Definition 2.5.1), with the only difference that the PMs have heterogeneous capacities. The placement algorithm is relaunched each time a task is finished in order to place its successor tasks. The idea of the placement policy presented in [68] is based on the consolidation technique; therefore, it places VMs on the PMs with the lowest energy consumption and shuts down the PMs that are not being used. In this work, each task requires a number of VMs in the infrastructure that need to be placed, to be properly executed. The VMs running a task can be scattered over several PMs, however each VM reserve the same amount of resources.

Scheduling multiple workflows

In [88] is presented a solution for scheduling multiple workflows with unpredictable random arrivals, and uncertain task execution times on an IaaS Cloud environment. The goal is to ensure that the deadline for each workflow is met, while minimizing the cost of renting the VM in the Cloud environment for a given user. All workflows are assumed to belong to the same user, so all resources (VMs) made available are capable of executing the task of every workflows (with the exception of software dependency issues), as no isolation issues are raised. In addition, the budget is common to all workflows. In this work, a *Independent task scheduling* algorithm is proposed. The algorithm first associate to each task a deadline, and then schedule them one by one, starting its decision process with the tasks having no dependency. The algorithm associates to each VM a rental cost (based on a hourly billing), and chooses for each task the VM that will minimize the overall cost and guarantee the deadline.

In [58], a genetic algorithm to solve the problem of scheduling multiple workflows with random arrivals is detailed. The authors present the three functions of a GA, for scheduling workflows under uncertain arrivals. By using a genetic algorithm, the authors minimize the user's budget, while respecting the deadlines of the different workflows. In this paper, only ready tasks (tasks whose parent tasks have been completed) are taken into account in the scheduling phase, which is called periodically, thus taking into account the uncertainty of the task execution time. As for [88], the objective of this algorithm is to minimize the budget of a single user, wanting to execute several workflows.

Discussion on scheduling algorithms on Cloud computing

As can be observed, the majority of solutions that can be found in the literature, provide algorithms for scheduling workflows belonging to a single user. They consider the Cloud environment as a third party, capable of providing an infinite number of resources, and therefore the only limitation is the budget of the user. As a result, fairness among multiple users is never taken into account, as it is assumed to be handled by the Cloud provider that manages the underlying physical infrastructure. In addition, as the solutions are only managing virtual resources, energy consumption cannot be fully addressed. The works presented in [68] and [72], consider a Cloud environment that is relatively different from the real Cloud environment (VMs are associated to one CPU core, and does not require any provisioning time), in addition, in [68] the decision of placement of the VMs within the infrastructure is made on the Cloud provider side, thus by taking into consideration the physical infrastructure itself.

Table 3.3 presents the criteria that we aim at taking into consideration in the contributions of this thesis, as far as scheduling is concerned. This table lists the contributions presented from related works, which take these criteria into account.

#	Metric	Related work
Task heterogneity consideration		
1	*Software*	[21] [68] [88] [132] [58] [48] [116] [26]
2	*Hardware*	-
Isolation		
3	*Isolation*	[21] [68] [88] [132] [58] [48] [116] [26]
Energy		
4	*Energy optimization*	[68] [72]
Fairness		
5	*Multiple users and Fairness*	-
Uncertainty		
6	*Execution time*	[48] [88] [58]
7	*Submission arrivals*	[88] [58]

Table 3.3 – Criteria of interest of the contributions of this thesis taken into account by Cloud scheduling algorithms

Table 3.4 presents the class of the algorithms used in a Cloud computing environment that can be found in the literature.

#	Class	Related work
Heuristic based algorithm		
1	*Independent task scheduling*	[21] [68] [88]
2	*List scheduling*	[48] [116] [72] [133]
4	*Iterative scheduling*	[132]
Meta-Heuristic based algorithm		
4	*Genetic algorithm*	[58] [26]
Exhaustive search algorithm		
5	*Breadth-first search*	-
6	*Backtracking*	-
7	*Branch and Bound*	-
Dynamic algorithm		
8	*Online*	[58]
9	*Reconsidering*	[88]

Table 3.4 – Class of the algorithm of the related work for Cloud computing environment

3.1.5 Discussion

We have seen in the previous subsections the related work algorithms that resolve the problem of scheduling scientific workflows in the Grid and Cloud environment. We have seen that the Grid computing environment offers the possibility to take into consideration three of our criteria: energy optimization, fairness and uncertainty.

The uncertainty of execution and submission is a criteria that can be taken into consideration in both Grid and Cloud environment, as it does not require any specific management of resources, but only a specific algorithm. In the literature, two different types of algorithms are suggested: online and dynamic algorithms. Online algorithms do not consider the future, and only schedule the tasks that are independent, and that can be run immediately. However, as a scientific workflows contains communications phases, we consider important to be able to rapidly be able to know where to send the files created by a task for its successors tasks, in order to avoid delays due to synchronizations.

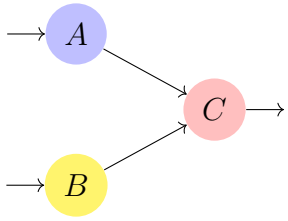


Figure 3.1 – Example of workflow with synchronization task

This kind of synchronization can be explained by Figure 3.1 and Figure 3.2. In the example, the task C has two parent tasks A and B. If we decide to schedule the task C only when it becomes independent, we would have to wait the end of the task A and the task B before taking any decision. However, if the task A is shorter than the task B, we have to wait until the task B is finished before sending the files created by A to the task C (as we do not know where to send the files), introducing delays as it can be observed in Figure 3.2. Even so, dynamic algorithms may introduce useless replications. Indeed, if the task C is rescheduled on a different resource after the transmissions of the files of the task A, the files would have to be sent again.

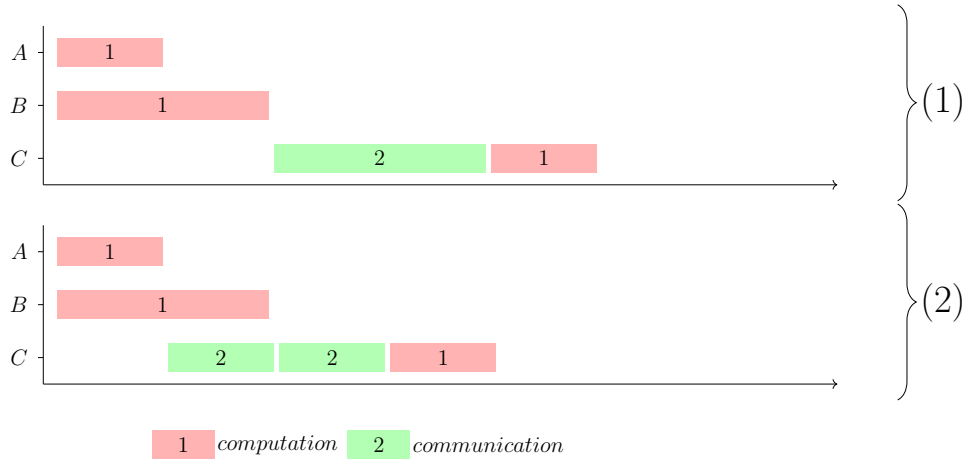


Figure 3.2 – Representation of delay introduced by synchronization with online (1) and with predictive (2) algorithms

The consideration of the two criteria energy and fairness, is made possible, in the Grid environment, by taking into account the physical infrastructure, and making management decision based on the real usage of this infrastructure. On the other side, in the Cloud environment, such management is impossible because of the nature of the environment that only gives the possibility to manage virtual infrastructure. The physical infrastructure itself is hidden to the users, and consequently it is difficult to predict the impact of the management of the virtual machines of one user on the infrastructure considering the two criteria energy and user fairness.

However, unlike the Grid environment, the Cloud environment allows to consider the two following criteria: task heterogeneity and isolation. Indeed, by providing access to VMs, the users are able to choose the OSs and softwares they want, and therefore to execute highly heterogeneous tasks. As we have seen in the previous subsections, the majority

of the work carried out on the Grid environment also does not take into consideration the heterogeneity of the tasks in term of hardware requirements. This limitation, in contrast to software heterogeneity, can be taken into account inside a Grid environment by adding a new dimension to the scheduling problem and considering space, and therefore a combination of the bin-packing (Def. 2.5.1) and the scheduling problems. The execution on a Cloud environment may also take this heterogeneity into account by considering different types of provisionable VMs (with different hardware capabilities).

The isolation criteria, is made possible easily in the Cloud environment, as every users provision VMs for the tasks they have to execute, and do not have access to the VMs of the other users. This isolation, is not as easy in the Grid environment. Indeed, to make isolation possible in a Grid environment, a full PM would have to be reserved for one user, thus leading to under used resources when the user does not need that much resources. As one can note, this would also lead to a decrease of the fairness.

Conclusion

In the contribution presented in Chapter 4 and Chapter 5, we detail scheduling algorithms for scientific workflows by locating the decision process on the Cloud provider side. The idea is to take advantage of both Grid and Cloud computing world, by taking into account the management of the physical infrastructure such as in a Grid environment, and giving access to virtual resources to the users. By considering the physical infrastructure during the scheduling process, the energy and the fairness criteria can be addressed, as well as the hardware heterogeneity of the tasks. On the other side, by providing virtual resources (such as VMs), the software heterogeneity and the isolation criteria can also be addressed. Furthermore, the algorithm presented in Chapter 5 is a dynamic algorithm that reconsiders the computed planning when new submission arrives, and that takes into consideration the uncertainty of the execution and communication time.

3.2 Scientific workflow execution

In the previous section, we have seen the related work on scientific workflow scheduling. In this section, is presented the state of the art regarding the automatic execution of scientific workflows. This section is divided in three parts. First, we will present the criteria in which we are interested in this thesis, in order to compare them with the related work. Then, we will present the related work around the automatic execution of workflows, and finally the last part will discuss the limitations of the state of the art regarding the criteria we aim at considering.

3.2.1 Criteria of interest

This subsection presents the criteria we want to consider in this thesis in regard to the workflow execution. In our context, multiple users submit workflow for execution on a Cloud computing environment. The Cloud provider is then giving access to a new kind of service dedicated to scientific workflows. This service can be divided into two main parts, the scheduling and the application of the planning computed by the scheduling algorithm. We can list our criteria as follows :

Automatic execution. In order to execute a scientific workflow, there are six operations that one must be able to carry out:

(a) determine the resources needed to execute the tasks of the workflow;

(b) reserve and release a resource;

(c) install on a resource the set of software dependencies required by workflow tasks;

(d) give the resources access to the input files of the tasks;

(e) execute the tasks of the workflow on a resource;

(f) retrieve the results of the execution.

The first operation (a) (*i.e.*, determining on which resource to execute the tasks of the workflow) is the resolution of a scheduling problem and has been discussed in the previous section 3.1. The operation (b) refers to the possibility to interact with the environment of execution in order to reserve a resource for the execution of one or multiple tasks (*e.g.*, be able to provision a VM in a Cloud environment). The operation (c) refers to the capacity to install all the required software dependencies (*e.g.*, OS, libraries, etc.) on a given resource, in order to prepare it for the execution of the tasks. The operations (d)

and (f) are file operations. Operation (d) refers to the need for the workflow execution, to transfer the files created by one task to its successor tasks, when the operation (f) refers to the capacity of giving access to the result files of the workflow execution to the user. The last operation (e) is the operation that consists in executing a task on a reserved and correctly configured resource. To summarize, the criteria in which we are interested is the automatization of those six operations, in order to follow a planning that has been computed by a scheduling algorithm without requiring the involvment of an human being.

Resource management. One of the criteria of interest of our contribution is to have a dedicated resource management. We have seen in the previous section around the scheduling of scientific workflows, that by locating the decision process on the Cloud provider side, better optimization can be performed in term of physical infrastructure usage. This criterion refers to the possibility of using a physical infrastructure with a dedicated service for the execution of scientific workflows belonging to multiple users, while being able to use algorithms that can make optimization of the infrastructure usage itself, which would be impossible with hidden physical infrastructure due to virtualization. This criterion can be divided in two different sub criteria: Elastic resources, and dedicated resource management. The elastic resource criterion is the capacity of reserve resources for a user, only when they are required and be able to release them as soon as possible, to make them usable for other users. When this criterion is mainly correlated to the decision of the scheduling algorithm, the environment of execution itself must be able to give access to elastic resources. The other criterion (dedicated resource management), is the capacity of providing resources that are designed for the execution of scientific workflow, without stacking virtualization technologies.

Modularity. The solution we aim at providing is intended to Cloud provider so as to offer a new type of service for the execution of scientific workflows. Cloud providers must be able to customize the way they want to manage their dedicated infrastructure, and which virtualization technologies to use. For this reason, one of our criteria is to give the ability to the Cloud provider to change the scheduling algorithm to use, as well as the ability to change the virtualization technology, with minimal code development. Hence mechanisms to enhance modularity are expected.

Separation of concerns. In our context, we can distinguish two different types of actors. The users who want to execute scientific workflows, that we will call the end-users, and the Cloud providers. These two types of actors have different concerns. On the one hand

the end-users are worrying about the development of the scientific workflows, and the results of the execution, and do not want to manage complex infrastructures nor provision resources; on the other hand, Cloud providers are worrying about the management of their infrastructures. Consequently, this criteria of separation of concerns is divided into two sub objectives : Providing a service with an hidden management for end-users, and providing a good turnkey solution for Cloud providers.

3.2.2 Scientific workflow execution systems

Historically, the execution of scientific workflows focused on the Grid infrastructure [59, 137]. In these contributions, when the end-users wanted to run workflows, they had to perform all the operations (operation (a) to (f)) manually. To avoid this and to facilitate the use of the scientific workflow, a lot of work has been done to automate workflow management and execution. In recent years, Cloud infrastructure are more likely to be targeted by workflow engines, for maintenance and cost reason. Indeed, in many cases private infrastructures are way more costly than on demand resources reserved on a IaaS public Cloud, and also requires more maintenance. Thus, in this section, the focus will be made on the execution of scientific workflows in Cloud environment. Two services are mainly targeted in the literature, the IaaS and the FaaS (cf. Section 2.4.3).

The first set of operations that have been automated are the operations (e), (d) and (f), *i.e.*, the automatic execution of tasks, and the file management operations. These operations are typically performed by what is called a *workflow engine*. Pegasus [56] and Hyperflow [27] are two famous engines that are to be deployed on resources reserved by the end-user. These resources can be either physical machines, as in a Grid environment, or virtual machines reserved in an IaaS in the Cloud. Both these workflow engines rely on a scheduler to determine which resource will execute the tasks (operation (a)).

The operation (c) (*i.e.*, installation of software dependencies) is also automated by all recent workflow engines of the literature. Indeed, these engines are nowadays all able to execute the tasks of the workflow inside containers to handle their software dependencies [123]. One can note that to handle file management operations (d) and (f), a distributed file system is generally deployed [27, 56] in order to provide access to the file from all resources at any times. Then the dependencies (transfers of the files) is managed by the distributed file system itself, more than by the workflow engine.

To be able to use Pegasus or Hyperflow, an end-user has to reserve a cluster of VMs in a IaaS Cloud service, install the workflow engine on the obtained virtual resources

and then launch the execution of the workflow with the engine. One may note that the operation (b) is carried out by the end-user in this case.

Some workflow engines leverages the use of FaaS (Function as a Service) Cloud service [4, 7, 77, 81]. The FaaS offers the opportunity to define functions (small computing entities using specific libraries), that will be executed in a serverless environment. In the FaaS, the deployment and management of resources are handled by the Cloud provider, and are hidden from the end-user. Basically, the resources provisioned in a FaaS are containers inside virtual machines. By using the FaaS, the resources are provisioned when required, and released when not used, therefore enhancing the elasticity of the solution. It can be said that the six operations of a workflow execution are automated by the workflow engines that leverage the FaaS paradigm.

3.2.3 Discussion

In our opinion Pegasus and Hyperflow engines suffer from two main issues from the end-user viewpoint, particularly when using Cloud infrastructures. On the one hand, they require from the end-user the ability to deploy an infrastructure by themselves, assuming that the end-user composing the workflow, *i.e.*, the scientist, is an expert in resource management. Moreover, they also expect from the end-user to manage the various dependencies required by the workflow engine to be properly configured and executed. In addition to being difficult and time-consuming, this limits the possibility of re-using the efforts between several users. Indeed, in addition to the workflow specification, deployment and configuration scripts have to be shared but must be adapted to different infrastructures and user requirements. As for Pegasus and Hyperflow, the workflow engines targeting a FaaS environment, require from the user to be set up and configured, and thus still require some management from the end-user. If DevOps tools could help in this task (*e.g.*, Terraform [20], Ansible [3], Puppet [18] etc.) this remains an important time and technical drawback for scientists.

On the other hand, as the end-users are responsible for resource provisioning on the Cloud, (in the case of Pegasus and Hyperflow), they have to determine the number of resources that will be needed for the overall execution of the workflow. This is a very difficult task that can lead to under-used resources. Furthermore, the resource usage of a workflow may vary during its execution lifetime (*e.g.*, the number of parallel tasks at each step is variable), and therefore a non elastic resource reservation seems inappropriate. This illustrates limitations in some engines of the related work on separation of concerns

#	metric	Related Work
	Automatic execution	
1	Software dependencies management (c)	[4] [7] [27] [56] [77]
2	Tasks execution (e)	[4] [7] [27] [56] [77]
3	File management (d) (f)	[4] [7] [27] [56] [77]
	Virtualization	
4	Elastic resources	[7] [4] [77]
5	Dedicated resource management	-
	Modularity	
6	Modular virtual resources (Heterogeneity)	[4]
7	Modular scheduler	-
	Separation of concerns	
8	Hidden management for the end-user	[4] [7]
9	Turnkey solution for the Cloud provider	-

Table 3.5 – Capabilities of Cloud-oriented workflow engines of the literature.

between the end-user and the Cloud provider, and on the elasticity of resource provisioning as illustrated in Table 3.5.

In addition, the workflow engines using a FaaS service, such as Argo [7] and Apache Airflow [4], by using an already deployed Cloud service (Kubernetes, AWS Lambda, etc.) assume that the management of the physical resources is already done by the Cloud provider, and only manage virtual resources. However, the Cloud provider is not aware of the kind of application running on its infrastructure, and thus cannot make dedicated optimization (fair sharing of the infrastructure between multiple user, energy optimizations, etc.). Indeed, such engines are dedicated for the execution of a workflow belonging to one user, and therefore make optimizations for this user (makespan minimization, cost optimization, etc.). By locating the decision process on the Cloud provider side, and by providing information about all the tasks that are to be executed on the infrastructure, and information about the topology of the physical infrastructure, the Cloud provider would be able to have a dedicated management of the infrastructure, and make decision based on the metric it wants to optimize, such as energy optimization or fairness.

The Cloud services provided by Cloud computing environment often stack different layers of virtualization (containers inside virtual machines) that can lead to performance loss. Yet, the performance is an important matter in the case of scientific workflows. Indeed, from the end-user viewpoint, as workflows are complex applications and can be time consuming, an efficient execution is preferred.

Furthermore, as workflow tasks generally are CPU intensive, and consume a lot of

memory, traditional strategies to consolidate resources in the Cloud are therefore not desirable as they create performance interference between hosted tasks. From the Cloud provider viewpoint this performance loss can be translated into a diminution of the Quality of Service that limit the adoption of their solution by scientists. This illustrates limitations in the related work to get resource management strategies dedicated to workflow execution, thus impacting the execution performance. This is summarized in Table 3.5 by *Dedicated resource management*.

A final limitation can be highlighted in the existing workflow engines: their lack of modularity. Indeed, none of them offer a way to easily introduce new kind of virtualization mechanisms nor new schedulers (*e.g.*, Pegasus is based on a scheduler that cannot be changed without a major modification of the engine itself). These limitations are also summarized in Table 3.5.

Conclusion

The contribution presented in Chapter 7 is a new service dedicated for the execution of scientific workflows. This service is presented as a turnkey solution for Cloud provider, and aims at achieving the separation of concern between the two types of actors that are the end-users and the Cloud providers. This service being installed on a multi-cluster infrastructure as would be any Cloud service, aims at enabling the opportunity for scheduling algorithm to efficiently consider infrastructure management. In addition, this service is designed to be modular, such that the scheduling algorithm and the virtualization mechanism could be easily changed with minimal code changes.

Part I

Workflow scheduling algorithms for Cloud providers

Chapter 4

OnlyUsedNodes : A workflow scheduling deadline-based algorithm for energy optimization

This chapter presents a new scheduling algorithm for Cloud providers that aims to reduce the energy consumption of the infrastructure. To this end, the algorithm attempts to minimize the number of physical machines required to execute a set of workflows (i.e. a workload). Indeed, one of the main operating cost of a Cloud computing provider is the electrical consumption. This consumption can be reduced by limiting the number of under-used physical machines [43, 79].

Contents

4.1 Introduction

Generally, users who develop scientific workflows are more interested in getting the result at a predictable time rather than as quickly as possible [127]. The scheduling algorithm OnlyUsedNodes, presented in this chapter, uses the notion of deadline constraint and the objective of energy minimization. We have seen in the context chapter, that PMs are more energy efficient when highly loaded, due to the non-linear power consumption of their CPUs.

Intuitively, when trying to reduce the makespan of a workflow, it is necessary to use a large number of PMs, which is costly for the Cloud provider. Indeed, since energy consumption cannot be represented by a linear function [74, 135] defined by the number of used physical resources and the running time, the use of a large number of PMs for a short time is more consuming than the use of a reduced number of PMs for a slightly longer time interval.

Conversely, if users can extend their deadlines instead of looking for the best possible makespan, a better energy optimization (*i.e.*, number of PMs used) can be achieved, resulting in a cost reduction for the Cloud provider. This cost reduction can then be reflected in the rental prices by a business model. One can note that such business model is above the scope of this contribution but it seems realistic to, at least partially, pass on the savings made by the Cloud provider to the user in the rental price, in order to reward the users who submits workflows with flexible deadlines.

In this chapter we consider a Cloud infrastructure from the provider point of view with the possibility of provisioning virtual machines (VMs) for two reasons. On one hand, the heterogeneity of the tasks composing the workflows makes it mandatory to load different operating systems (cf Section 2.1). On the second hand, a multi-user workload is considered, thus, strong isolation, for security reasons, must be guaranteed which is made easier by the VMs.

This chapter presents the following contributions: (1) a detailed model of the scheduling problem under consideration; (2) an adaptation of the HEFT algorithm, namely OnlyUsedNodes, that takes into account both virtualization and deadlines and that minimizes the number of PMs used to plan a workload. The remainder of this chapter is organized as follows. Section 4.2 details the problem modeling, Section 4.3 presents our

new algorithm OnlyUsedNodes. Finally, the section 4.4 concludes this chapter.

4.2 Problem modeling

In this section, we present the model that describes our scheduling problem. The problem we are addressing is to create a planning containing all the tasks of the workflows being submitted at a given instant with a deadline. Every tasks are to be executed by virtual resources, that are used to satisfy the dependencies of the tasks and their resource requirements. In the computed planning the capacities of the physical machines are respected, as well as the deadlines of the workflows. The objective of the algorithm is to minimize the cloud provider's energy consumption by reducing the number of physical machines used to execute the workflows. All the symbols presented in this section are summarized in Table 4.2.

4.2.1 Applications and execution environment

Workflow definition - Let $\mathcal{J}$ be the set of tasks to be scheduled and executed. These tasks make up the different workflows. Each workflow $w \in \mathcal{W}$ can be represented as a DAG - **D**irected **A**cyclical **G**raph $G = (T, D)$, where $T \subset \mathcal{J}$ and D is the set of data dependencies between tasks. Each edge of the graph $d \in \mathcal{D}$ is weighted, and its weight represents the size of the data to be transferred from one task to another and is denoted $d_{i \to j}$ for a task i and $j \in \mathcal{J}$. Each task has constraints that can be divided into two categories: the hardware constraints that can be quantified, such as the number of CPU cores, the amount of memory; and the software constraints (OS, library, etc.). In this chapter, it is assumed that the number of instructions to execute a task is known. Such information can be retrieved by sampling. Let $succ_j$ and $pred_j$ be the list of respectively the successors and the predecessors of the task $j \in \mathcal{J}$, such that $\forall\ p \in succ_j,\ \exists\ d_{j \to p} \in D$ and $\forall\ p \in pred_j,\ \exists\ d_{p \to j} \in D$.

Execution environment - The targeted environment is a multi-cluster infrastructure, namely a set of clusters of PMs, where each PM is running an hypervisor, giving the ability to provision different types of VMs on each one of them. $\mathcal{V}$ denotes the set of VMs - in the planning - that are to be provisioned within the execution environment. Virtual machines are provisioned from images, which have different hardware and software capacities. Let

$\mathcal{N}$ be the set of compute nodes (physical machines). A node is associated with a given cluster.

The bandwidth between clusters is assumed to be heterogeneous, and therefore the bandwidth between the nodes of several different clusters is also heterogeneous. Let $bw^{\mathcal{N}}_{n\leftrightarrow m}$ be the bandwidth between the nodes n and $m \in \mathcal{N}$.

Let $speed^{\mathcal{N}}_n$ be the speed of the node $n \in \mathcal{N}$, as the number of instructions per CPU core per instant (*e.g.*, seconds). No over-provisioning of the nodes is considered in this chapter; consequently one CPU core is reserved for one VCPU, 1MB of virtual memory is reserved for 1MB of memory, etc.

4.2.2 Software and Hardware constraints

Let $H_{t,n} =< h_{t,n,v}, \ldots, h_{t,n,|\mathcal{V}|} >$ be a vector, for each node $n \in \mathcal{N}$, and for each instant $t \in \mathbb{N}$, where $h_{t,n,v} = 1$ if and only if the VM $v \in \mathcal{V}$ is hosted by n at instant t, and $h_{t,n,v} = 0$ otherwise. Similarly let $E_{t,v} =< e_{t,v,j}, \ldots, e_{t,v,|\mathcal{J}|} >$ be a vector, for each $v \in \mathcal{V}$, and for each instant $t \in \mathbb{N}$, where $e_{t,v,j} = 1$ if and only if $j \in \mathcal{J}$ is executed by v at instant t, and $e_{t,v,j} = 0$ otherwise. Let $\mathcal{C}$ be the set of different hardware capacities (*e.g.*, RAM, CPU, HDD, etc.). For each capacity $k \in \mathcal{C}$, three vectors are defined :

— $\mathcal{C}^{\mathcal{N}}_k$, of size $|\mathcal{N}|$, which represents the capacity k provided by each nodes $n \in \mathcal{N}$, such that $\mathcal{C}^{\mathcal{N}}_k(n)$ is the capacity k that $n \in \mathcal{N}$ can supply.

— $\mathcal{C}^{\mathcal{V}}_k$, of size $|\mathcal{V}|$ represents the required amount of resource k needed by each $v \in \mathcal{V}$, such that $\mathcal{C}^{\mathcal{V}}_k(v)$ is the amount of resource k reserved by $v \in \mathcal{V}$.

— $\mathcal{C}^{\mathcal{J}}_k$, of size $|\mathcal{J}|$, defines the amount of resource k required by each $j \in \mathcal{J}$.

For each capacity $k \in \mathcal{C}$, the two following constraints are defined in such a way that over-provisioning is not considered both for physical and virtual machines:

$$\mathcal{C}^{\mathcal{V}}_k \cdot H_{t,n} \leq \mathcal{C}^{\mathcal{N}}_k(n) \ \forall n \in \mathcal{N}, \ \forall t \in \mathbb{N} \tag{4.1}$$

$$\mathcal{C}^{\mathcal{J}}_k \cdot E_{t,v} \leq \mathcal{C}^{\mathcal{V}}_k(v) \ \forall v \in \mathcal{V}, \ \forall t \in \mathbb{N} \tag{4.2}$$

The software requirements of the tasks are also taken into account. Thus, let $\mathcal{S}$ be the set of software requirements (*e.g.*, OS, library, language, etc.). For each software requirement $s \in \mathcal{S}$, a task requiring the software s must be executed by a virtual machine possessing the software s.

4.2.3 Temporal dependency constraints

Let $speed^{\mathcal{V}}_{t,v}$ be the speed of $v \in \mathcal{V}$ at instant t - let us remind that the speed is in number of instructions per core (here per VCPU). As no over-provisioning is considered, the deterioration of the VM speed is assumed to be low [101], and considered to be 5% of the host node speed. Thus, $speed^{\mathcal{V}}_{t,v} = speed^{\mathcal{N}}_{n} \cdot 0.95$ if and only if $h_{t,n,v} = 1$. For simplicity in the rest of the chapter, the speed of the VM v is time-independent and denoted as follows $speed^{\mathcal{V}}_{v}$, because a VM is associated to one and only one node (no migration). In this chapter we assume, that the length of the tasks are sufficiently short, so that a migration of a VM would provide no gain as it would imply time delay. However, the migration of the VM could be an interesting orientation for a future work.

In this contribution, VMs are dynamically provisioned on demand to manage the specific constraints of each task. A VM needs a certain amount of time to start and be ready to perform tasks. Let $start^{\mathcal{V}}_{v}$ be the instant when $v \in \mathcal{V}$ initiates its powering on. Let $boot^{\mathcal{V}}_{v}$ be the number of instructions to run before $v \in \mathcal{V}$ is ready for usage. And let $ready^{\mathcal{V}}_{v}$ be the instant when $v \in \mathcal{V}$ is ready to compute tasks, such that $ready^{\mathcal{V}}_{v} = start^{\mathcal{V}}_{v} + \frac{boot^{\mathcal{V}}_{v}}{speed^{\mathcal{V}}_{v}}$. We define the instant when a task can start as follows:

$$\forall j \in \mathcal{J} \quad start^{\mathcal{J}}_{j} = \max_{p \in pred_j} (end^{\mathcal{J}}_{p} + \max(\frac{d_{p \to j}}{bw^{\mathcal{N}}_{host^{\mathcal{J}}_{p} \leftrightarrow host^{\mathcal{J}}_{j}}}, ready^{\mathcal{V}}_{v})), \tag{4.3}$$

where v is the VM executing the task j, $host^{\mathcal{J}}_{j}$ refers to the node hosting the VM executing the task $j \in \mathcal{J}$, $pred_j$ is the list of predecessors of the task j, $d_{p \to j}$ is the size of the dependency between the task p and the task j, $end^{\mathcal{J}}_{p}$ (see Equation 4.5) is the instant of the end of execution of the task p and $bw^{\mathcal{N}}_{n \leftrightarrow m}$ is the bandwidth between two physical nodes, a clarification of the communication is given in the next subsection.

The execution time of a task j depends on its weight W_j (number of instructions) divided by the speed of the resource that will execute the task (See Equation 4.4). This execution time is used to compute the end time of the task (See Equation 4.5).

$$\forall j \in \mathcal{J} \quad exec^{\mathcal{J}}_{j} = \frac{W_j}{speed^{\mathcal{N}}_{host^{\mathcal{J}}_{j}} \cdot 0.95} \tag{4.4}$$

$$\forall j \in \mathcal{J} \quad end^{\mathcal{J}}_{j} = start^{\mathcal{J}}_{j} + exec^{\mathcal{J}}_{j} \tag{4.5}$$

Finally, each workflow $w \in \mathcal{W}$ has a deadline $dead^{\mathcal{W}}_{w}$ such that all tasks forming the workflow w must be finished before $dead^{\mathcal{W}}_{w}$.

4.2.4 Communications

We have seen in Equation 4.3, that a task cannot be started before the files of its predecessors are successfully transferred to the node that will host the VM that will execute the task.

In this chapter the network used to transfers the files from one task to another is the physical network that links the nodes together, and not a virtual network that is used by the VMs. Indeed in this contribution, VMs are only considered as computing resources, and therefore are only used to execute the tasks. As a result, communications between different tasks are directly performed between nodes without virtualization layer. We assume that communications require a very low CPU load [31], so this load is not taken into account when computing the resource usage ($\mathcal{C}^{\mathcal{N}}_{core}$) of the node. This assumption made possible an overlap of communications and computations, which improves the quality (*e.g.*, execution time) of workflows executions.

Figure 4.1 illustrates this claim with an example where two users are sharing the same node. In this example, the first user uses a VM that is consuming almost all the resources of the node (CPU cores for example), that we will call $v1$. This VM makes impossible to start a new VM on the node without over-provisioning, and therefore the second user has to wait the end of $v1$ to start a new VM $v2$ and execute the tasks.

In the first scenario (at the top of Figure 4.1), communications are performed within the VMs. In this case, the execution is fully sequential. Indeed, the VM $v1$ must be stopped before the second VM $v2$ can be started, as a result the communications of the second user are also waiting the end of $v1$ (as they are performed within $v2$). In the second scenario, on the contrary, since communications are made outside the VMs, they are performed in parallel with the execution of the tasks of the other users, and in parallel with the booting process of the VMs.

4.2.5 Cost modeling

Our contribution aims at minimizing the operational costs of the Cloud computing provider. We assume that this cost is directly correlated to the data center's power consumption. It has already been discussed in the context chapter (cf Section 2.6.2), that the CPU's energy consumption is not a linear function defined by the load and the running time. Therefore, reducing the number of nodes for a longer period of time is not equivalent to using more nodes for a shorter period.

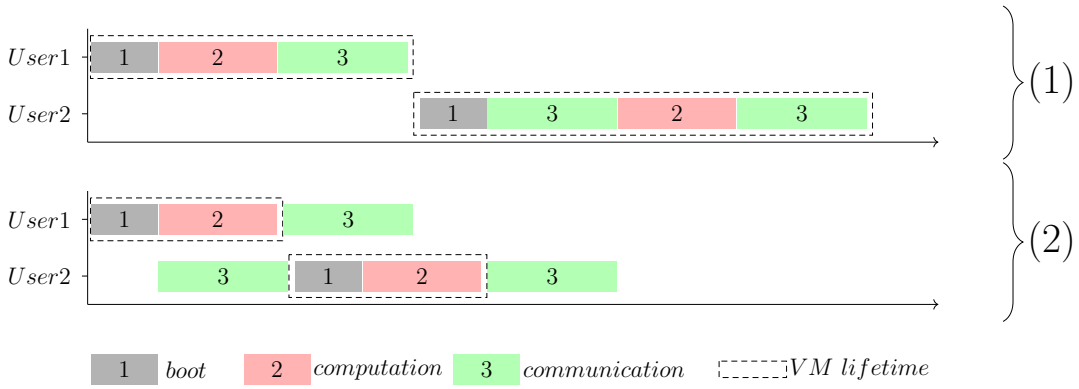


Figure 4.1 – Representation of the gain between communication done on the VMs (1) and on the node (2)

To properly model the cost and therefore the gain of our solution, the consumption of a node is defined accordingly to [135] and as follows:

$$consumption_n = \sum_{t \in \mathbb{N}} Pmax_n + (\frac{Pidle_n - Pmax_n}{\ln 0.01}) \cdot \ln cpu\_load_{t,n}, \tag{4.6}$$

$$cpu\_load_{t,n} = \frac{\sum_{v \in \mathcal{V}} (\mathcal{C}^{\mathcal{J}}_{core}(v) \cdot E_{t,v}) \times h_{t,n,v}}{\mathcal{C}^{\mathcal{N}}_{core}(n)}, \tag{4.7}$$

where $Pidle_n$ is the idle consumption of the node, $Pmax_n$ is the power consumption of the node when fully used. The *cpu_load* computed by Equation 4.7 is the percentage of cores used on a CPU at a given time. The value 0.01 is an arbitrarily small value to keep the logarithm calculable, and represents the minimal *cpu_load*.

One can note that we have conducted experiments on the Ecotype cluster [13] (SeDuCe [103] platform) that perfectly match this model (see Figure 2.9).

4.2.6 Objective

The objective considered in this contribution is to minimize the number of nodes required to execute a set of workflows and consolidate the already used nodes, in order to reduce the energy consumption of the infrastructure, and by extension the cost of the Cloud provider.

Symbol	Definition
Workflow	
$w \in \mathcal{W}$	A workflow
$j \in \mathcal{J}$	A task
$T_w \subset \mathcal{J}$	The list of tasks of the workflow $w \in \mathcal{W}$
$d_{i \to j} \in \mathcal{D}$	A dependency between two tasks i and $j \in \mathcal{J}$
$dead_w^{\mathcal{W}}$	The deadline of the workflow $w \in \mathcal{W}$
$succ_j$	The list of successor tasks of the task $j \in \mathcal{J}$
$pred_j$	The list of predecessor tasks of the task $j \in \mathcal{J}$
Execution environment	
$n \in \mathcal{N}$	A node
$v \in \mathcal{V}$	A VM
$bw_{n \leftrightarrow m}^{\mathcal{N}}$	the bandwidth between the node n and $m \in \mathcal{N}$
$speed_n^{\mathcal{N}}$	the speed of the node $n \in \mathcal{N}$
$speed_v^{\mathcal{V}}$	the speed of the VM $v \in \mathcal{V}$
Software and Hardware constraints	
$k \in \mathcal{C}$	A capacity (*e.g.*, RAM, CPU, HDD, etc.)
$\mathcal{C}_k^{\mathcal{N}}(n)$	The amount of capacity k that the node $n \in \mathcal{N}$ can supply
$\mathcal{C}_k^{\mathcal{V}}(v)$	The amount of capacity k reserved by the VM $v \in \mathcal{V}$
$\mathcal{C}_k^{\mathcal{J}}(j)$	The amount of capacity k reserved by the task $j \in \mathcal{J}$
$s \in \mathcal{S}$	A software dependency (OS, library, language, etc.)
Temporal constraints	
$start_v^{\mathcal{V}}$	The instant of powering on of the VM $v \in \mathcal{V}$
$boot_v^{\mathcal{V}}$	The number of instructions to run in the boot process of the VM $v \in \mathcal{V}$
$ready_v^{\mathcal{V}}$	The ready instant of the VM $v \in \mathcal{V}$
$start_j^{\mathcal{J}}$	The start instant of the task $j \in \mathcal{J}$
$end_j^{\mathcal{J}}$	The end instant of the task $j \in \mathcal{J}$
$exec_j^{\mathcal{J}}$	The execution time of the task $j \in \mathcal{J}$
Assignment	
$H_{t,n}$	The vector of hosted VM on the node $n \in \mathcal{N}$ at the instant $t \in \mathbb{N}$
$E_{t,v}$	The vector of executing tasks on the VM $v \in \mathcal{V}$ at the instant $t \in \mathbb{N}$
$host_j^{\mathcal{J}}$	The node $n \in \mathcal{N}$ hosting the VM that executes the task $j \in \mathcal{J}$

Figure 4.2 – Table of symbols of the model of section 4.2

4.3 OnlyUsedNodes algorithm

In this section, we present our deadline aware scheduling algorithm OnlyUsedNodes based on the HEFT algorithm. As part of our algorithm, a set of workflows - each associated with a deadline - is submitted at a given time to the Cloud provider. The algorithm is launched following a regular clock and takes into account the VMs already

hosted on the Cloud infrastructure. The algorithm computes a planning, that is containing all the tasks of the submitted workflows. The planning corresponds to an instance of the vectors H and E presented in the model section, where all the constraints are respected.

4.3.1 Priorities and deadlines

In HEFT, a priority is calculated for each task j of each workflow such that tasks with higher priorities are scheduled first. This priority is established according to the average completion time (on all possible nodes) of a task, denoted $\overline{exec_j^{\mathcal{J}}}$, as well as its average communication time (between all possible nodes), denoted $\overline{com_{j\to p}^{\mathcal{J}}}$. The rank of a task is shown in Equation 4.8.

$$rank_j^{\mathcal{J}} = \overline{exec_j^{\mathcal{J}}} + \max_{p \in succ_j} \left(\overline{com_{j\to p}^{\mathcal{J}}} + rank_p^{\mathcal{J}} \right) \tag{4.8}$$

Figure 4.3 presents an example of workflow and Table 4.1, lists the weight W_j, of each task of the example. This table also presents the rank of each task, assuming that the infrastructure is composed of two nodes with a speed of respectively 2GFlops (floating operations per second) and 1.8GFlops, and that the bandwidth between the nodes and between a node and itself is 10Mbps. This table presents the average execution time of each task (based on its weight), the average communication time of each task to its successors, and the rank of each task. One can note that the rank of A is based on the rank of B, due to the fact that $rank_B^{\mathcal{J}} + \overline{com_{A\to B}^{\mathcal{J}}}$ is greater than $rank_C^{\mathcal{J}} + \overline{com_{A\to C}^{\mathcal{J}}}$.

	A	B	C	D	E	F
W_j	10×10^9	20×10^9	10×10^9	25×10^9	13×10^9	45×10^9
$\overline{exec_j^{\mathcal{J}}}$	5	11	5	13	7	24
$\overline{com_{j\to p}^{\mathcal{J}}}$	{B: 3, C: 5}	{D: 8}	{E: 9}	{F: 8}	{F: 10}	{}
$rank_j^{\mathcal{J}}$	72	64	55	45	41	24

Table 4.1 – Weight and ranks of the example workflow of Figure 4.3

As illustrated in the main algorithm of HEFT in Algorithm 1, once all the tasks are sorted by rank (computeRankList function), the scheduling algorithm is launched. This algorithm will be detailed later in Section 4.3.3.

The first operation performed by OnlyUsedNodes, as indicated in Algorithm 2, is to order the workflows by difficulty (function sortWorkflows). This difficulty is defined by the difference between the deadline and the average execution time of the critical path of the workflow (on all possible nodes) accordingly to Equation 4.9. The average critical

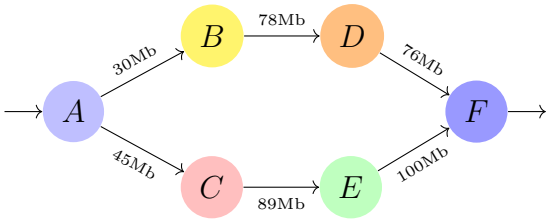


Figure 4.3 – Example of scientific workflow

Algorithm 1 HEFT algorithm

```
function HEFT (tasks, nodes)
    task_list ← computeRankList (tasks, nodes)          ▷ Eq. (4.8)
    for all t ∈ task_list do
        scheduleEarliest(t, nodes)
```

path is already computed by the rank of the entry tasks of the workflow (see Equation 4.8). In the example of Figure 4.3 and Table 4.1, the difficulty of the workflow would be 28, if its deadline is 100 seconds.

$$difficulty_w = dead_w^{\mathcal{W}} - \max_{j \in \mathcal{J}} \left(rank_j^{\mathcal{J}} \right). \tag{4.9}$$

Then, OnlyUsedNodes processes each workflow by decreasing difficulty unlike HEFT that directly ranks all tasks of all submitted workflows.

Algorithm 2 OnlyUsedNodes algorithm

```
function OnlyUsedNodes (workflows, deadlines, nodes)
    workflow_list ← sortWorkflows (workflows, deadlines, nodes)
    for all w ∈ workflow_list do
        OnlyUsedNodes-workflow(w.tasks, nodes, deadlines[w])
```

For each workflow to schedule, the function OnlyUsedNodes-workflow shown in Algorithm 3 is called. The first step of this function is to calculate the priority of each task (computeRankList function) of the input workflow. The second step is to compute the deadlines for each task of the current workflow. As already explained, in OnlyUsedNodes, the user submits a workflow with a global deadline, including all the tasks that make up the workflow. We also consider (as already explained in Section 4.2) that the number of instructions necessary to perform each task is given when the workflow is submitted to the scheduler. It is therefore possible to compute a time limit per task that must not be exceeded in order to meet the overall initial deadline of the workflow. Each task deadline will be used to avoid a full exploration of a branch of the search space that will necessarily lead to an overdue global deadline.

Algorithm 3 OnlyUsedNodes single workflow scheduling

function OnlyUsedNodes-workflow $(tasks, nodes, deadline)$
  $task\_list \leftarrow$ computeRankList $(tasks, nodes)$ ▷ Eq. (4.8)
  $dead\_list \leftarrow$ computeDeadlines $(tasks, nodes, deadline)$ ▷ Eq. (4.10)
  $ons \leftarrow$ filterUsedNodes$(nodes)$
  $offs \leftarrow$ filterUnusedNodes$(nodes)$
  return scheduleBackTrack$(0, 0, task_list, dead_list, ons, offs, false)$

	A	B	C	D	E	F
$min\ exec_j^{\mathcal{J}}$	5	10	6	12	7	23
$min\ com_{j\to p}^{\mathcal{J}}$	{B: 3, C: 5}	{D: 8}	{E: 9}	{F: 8}	{F: 10}	{}
$dead_j^{\mathcal{J}}$	36	49	51	69	67	100

Table 4.2 – Deadlines of the tasks of the example workflow of Figure 4.3 with a global deadline of 100 seconds

The deadline of the tasks is computed in the computeDeadlines function of Algorithm 3. To compute the deadline of each task, it is necessary to start with the exit tasks of the workflow (tasks with no successors). Indeed, for the exit tasks, the time not to be exceeded is the overall deadline of the workflow. Then for any other task $j \in \mathcal{J}$ that has successors, the deadline is represented by Equation 4.10, where $succ_j \subset T \subset \mathcal{J}$ is the set of successor of j. The idea is that the deadline of a task is the moment when, even if the fastest node is used for the execution of the successors, the overall deadline of the workflow cannot be guaranteed. Table 4.2 presents the deadline of each task of the example workflow of Figure 4.1 with the same infrastructure as in Table 4.1.

$$\forall j \in \mathcal{J} \text{ where } |succ_j| > 0,$$
$$dead_j^{\mathcal{J}} = \min_{p \in succ_j} \left(dead_p^{\mathcal{J}} - \min_{n,m \in \mathcal{N}} \left(\frac{W_p}{speed_m^{\mathcal{N}} \cdot 0.95} + \frac{d_{j\to p}}{bw_{n\leftrightarrow m}^{\mathcal{N}}} \right) \right) \tag{4.10}$$

Finally, the OnlyUsedNodes algorithm maintains two lists of physical machines (ons and $offs$ lists) so that the scheduling algorithm knows which PMs will be used to execute tasks, and which ones will not be used. The rest of this section details the scheduling algorithm of OnlyUsedNodes and HEFT.

4.3.2 Backtrack scheduling algorithm

OnlyUsedNodes tries to minimize the number of nodes used to schedule the workflows of a workload. To this end, our solution extends the HEFT algorithm with a partial backtracking heuristic. This heuristic consists in trying to perform scheduling on nodes that already have planned task execution and VMs provisioning. As soon as this attempt

fails, because the tasks deadlines are no longer met, a backtrack is performed and a new node is considered. This new algorithm is defined in the recursive function SCHEDULEBACKTRACK of Algorithm 4 (initially called in Algorithm 3). This function takes as input the *id* of the first task (*backTo*), the *id* of the current task, the task list of the current workflow, the list of tasks deadlines, the list of used nodes and the list of unused nodes and the last input *force* that will be explained later.

Algorithm 4 ONLYUSEDNODES backtrack scheduling

```
function SCHEDULEBACKTRACK (backTo, id, tasks, deads, ons, offs, force)
    task ← tasks [id]
    deadline ← deads [id]
    if not force and SCHEDULEEARLIEST(task, ons, deadline) then
        if SCHEDULEBACKTRACK(backTo, id+1, tasks,
                             deads, ons, offs, False) then
            return True
    if id = backTo then
        if SCHEDULEEARLIEST(task, ons + offs, deadline) then
            new_ons ← FILTERUSEDNODES(ons + offs)
            new_offs ← FILTERUNUSEDNODES(offs)
            if SCHEDULEBACKTRACK(backTo+1, id+1, tasks, deads,
                                 new_ons, new_offs, |new_ons| == |ons|) then
                return True
    return False
```

Algorithm 4 is the main part of the ONLYUSEDNODES algorithm. If the deadline constraint cannot be met by considering used nodes only, the algorithm backtracks to the first task that was not scheduled on an unused node. This task is the task whose *id* is *backTo*. It may happen that an already used node offers a better makespan than an unused node, and thus that the list *new_ons* and the list *ons* would be the same. However, the scheduling had failed with this number of used nodes, so it is not necessary to wait for the first step (scheduling only on the used nodes) to fail. The input parameter *force* is used to this end, and is evaluated to false, only when a new node has been considered for the scheduling of a task, thus avoiding useless backtracking.

It can be noted that in our scheduling algorithm, the maximum number of backtracks is equal to the number of unused nodes at entry point. ONLYUSEDNODES manages each workflow one after the other by decreasing difficulty, unlike HEFT which handles all the tasks of all workflows in a single sorted list. In ONLYUSEDNODES this cannot be done, because the backtrack part would go back to tasks that have little impact on the task that has not been scheduled. For this reason, if all tasks are processed together, all nodes will be used and a result close to HEFT will be observed but with worse complexity. For

this reason, OnlyUsedNodes schedules the workflows one by one, sorted by decreasing difficulty.

4.3.3 Resource selection

Another essential part of the OnlyUsedNodes algorithm is the resource selection function scheduleEarliest. In HEFT, this resource selection phase does not take into account the heterogeneity of the PMs composing the infrastructure, and assumes that each task consume exactly one CPU core. Consequently, in the HEFT algorithm, the resource selection is relatively trivial and consists in choosing the CPU core that will give the lowest makespan. In our context, the tasks require different amount of resources, and the resources are spread across PMs, and in addition VMs must be considered.

The function scheduleEarliest is called each time a task is to be scheduled by the OnlyUsedNodes algorithm, and is detailed in Algorithm 5. Obviously, this function takes deadlines into account,

Algorithm 5 OnlyUsedNodes resource selection

```
function scheduleEarliest (task, nodes, deadline)
    start ← 0
    bestPlace ← (None, 0, 0, deadline + 1)
                                  ▷ A place is a tuple (node, start, duration, end) for a task
                                    ▷ deadline + 1 means that the deadline is not respected
    start ← GetMaximumEnd(pred_task)
    for all n ∈ nodes do
        exec ← ComputeExecTime(task, n)                                   ▷ Eq. (4.4)
        place ← getPlaceOnNode(n, task, start, exec, deadline)
                                  ▷ Eq. (4.1, 4.2)
        if place.end < bestPlace.end then bestPlace ← place
    if bestPlace.node ≠ None then
        resourceProvisioning(bestPlace.node, task,
                                  bestPlace.start, bestPlace.duration)
```

This resource selection phase is specific to the problem modeled in Section 4.2. Due to virtualization and software constraints, virtual machines must be provisioned. A VM is provisioned for only two reasons: (1) the VMs already used by the owner of the current workflow cannot meet the requirements of the current task to be scheduled (or the user does not have one), and for isolation and safety reasons VMs of other users cannot be used; (2) by starting a new VM the quality of the schedule (makespan) is enhanced.

The function GetPlaceOnNode called in Algorithm 5 is used to find a suitable location to execute a task inside an existing or a new VM onto a specific node (physical

machine). First, this function looks for a placement on the existing VMs of the user that minimizes the ending time of the task. Then, the function selects a new VM (that takes into account the constraints of the task) and calculates the ending time of the task on this new VM (taking into account its boot time). The place inside a VM that offers the earliest ending time is selected. The placement found inside a VM may extend the lifetime of that VM, thus the function must also know the capacities of the node hosting the VM.

Finally, in Algorithm 5, the function RESOURCEPROVISIONNING reserves the place on the node, updates all capacities information, and eventually stores the new selected VM. A reservation being the recording of the task, the VM and the temporal information (start instant, estimated end time, etc.) in the planning that is computed by the scheduling algorithm. One can note that this reservation may be released when the ONLYUSEDNODES algorithm backtrack on previous task scheduling.

A variation of the HEFT algorithm can be defined easily by replacing its resource selection function by the function SCHEDULEEARLIEST function of the ONLYUSEDNODES algorithm. Thus, a variation, that will be called V-HEFT in the rest of this chapter, is defined, and takes into account the heterogeneity of the task requirements, the heterogeneity of the PMs, and the provisioning of VMs. The V-HEFT algorithm will be used in the evaluation section, in order to compare the ONLYUSEDNODES and the HEFT algorithm in fair conditions.

4.3.4 Complexity

The complexity of the function GETPLACEONNODE is common to both V-HEFT and ONLYUSEDNODES algorithms, thus this function is not taken into account for the complexity. The complexity of V-HEFT is a function of the number of tasks in the workflow and the number of nodes on which the scheduling solution will be done. Therefore, its worst-case complexity V-HEFT is $\mathcal{O}(|\mathcal{J}| \times |\mathcal{N}|)$, and is exactly equals to its average-case complexity.

The complexity of our new algorithm ONLYUSEDNODES, is different due to the partial backtracking. The worst case, is when all the nodes are needed to schedule the workflow and when the backtrack is performed when reaching the last task of the workflow. The worst-case complexity is then $\mathcal{O}(\sum_{i=1}^{|\mathcal{J}|}(|\mathcal{N}| + \sum_{k=1}^{i} k))$. However, this complexity is on average far lower than the worst case, as only used nodes are explored, and for most of the tasks the number of used nodes may be low. Section 6.1 validates this claim.

4.4 Conclusion

This chapter tackles the scheduling of heterogeneous scientific workflows while minimizing the energy consumption of Cloud providers. The OnlyUsedNodes algorithm has been presented as a solution to this scheduling problem. OnlyUsedNodes adds deadlines to workflows so that the number of PMs needed to run the workload is reduced, as well as the energy consumption. This algorithm is based on v-HEFT, a variant of HEFT that takes virtualization into account. In Chapter 6 The OnlyUsedNodes algorithm is compared to v-HEFT to assess the impact of the deadlines on the energy consumption of the Cloud provider. Experiments on real infrastructure have been conducted, in order to evaluate this algorithm.

In the next chapter, another algorithm is presented. This algorithm operates in a different context, where instead of being static, the problem is dynamic because the users can submit workflows at random times. Therefore, the algorithm that is presented in the next chapter is a dynamic algorithm, which takes into account the uncertainty of arrival. It also uses the deadline of the workflows, not only for energy optimization, as the OnlyUsedNodes algorithm does, but also to ensure fairness between users.

Chapter 5

NearDeadline : Dynamic Multi-User Workflow Scheduling Algorithm for Fairness and Energy Optimization

This chapter introduces a new scheduling algorithm for Cloud providers. This algorithm aims to guarantee fairness among users, who submit a scientific workflow for execution at a uncertain point of time on the execution environment, as well as to reduce the energy consumption. The algorithm also takes into account the uncertain execution time of the tasks and uncertain boot times for the virtual machines to be properly provisioned and operational.

Contents

5.1 Introduction

We have seen in the Chapter 4 an algorithm that creates a planning for the execution of scientific workflows belonging to multiple users, from the Cloud provider side. By locating the scheduling decision on the Cloud provider side, a better optimization of the infrastructure can be achieved. However, in a public execution environment multiple users are in competition for a set of resources, on which they submit different applications with uncertain task executions at unpredictable random arrivals. Therefore a static algorithm - as detailed in Chapter 4 - is no longer sufficient, as the problem evolves and the planning has to be updated accordingly.

As far as we know there is no work on Grid or Cloud computing infrastructures that aims at solving the following problematics: (1) uncertainty, that is to say juggle with the unpredictable random arrivals of workflows, uncertain task execution times, and uncertain VM boot times; (2) user fairness, in other word, be able to schedule a multi-user workload with a fair sharing of the available resources; (3) energy minimization, *i.e.*, minimizing the energy consumption of the Cloud infrastructure.

In this chapter, as for the first contribution detailed in Chapter 4, we consider a Cloud infrastructure from the provider point of view with the possibility of provisioning virtual machines (VMs) for two reasons. On one hand, the heterogeneity of the tasks composing the workflows makes it mandatory to load different operating systems (cf Section 2.1). On the second hand, a multi-user workload is considered, thus, strong isolation, for security reasons, must be guaranteed which is made easier by the VMs.

In addition, in this chapter each workflow is submitted with a deadline. This deadline is used to define the fairness between the users, where an execution is considered fair if all the deadlines of the workflows are guaranteed. This chapter presents the following contributions: a dynamic scheduling algorithm for Cloud provider, named NearDeadline. This algorithm, by taking into account the deadlines, schedules the workflows of all users on a set of PMs in order to maximize the user fairness and minimize the energy consumption. The algorithm is able to dynamically update the planning it has created, when new submissions of workflows arrive.

The rest of the chapter is organized as follows: Section 5.2 gives a detailed model of the problem. Then Section 5.3 presents our new algorithm NearDeadline. Finally, the section 5.4 concludes this work.

5.2 Modeling and Problem Formulation

This section presents a model that describes the scheduling problem: schedule all the tasks of submitted workflows on virtual resources in order to satisfy task dependencies and their resource needs, while respecting the capacity constraints of the physical machines. The problem modeling of this chapter is close to the problem modeling presented in Section 4.2, but have some differences due to the dynamic nature of the problem, and the objectives that are not the same. Our objectives are to maximize the user fairness by respecting the deadline of each users, and to minimize the energy consumption of the physical machines. All the symbols presented in this section are summarized in Table 5.1.

5.2.1 Workflow definition

A scientific workflow can be defined as a DAG (Directed Acyclic Graph) $G = (T, D)$ were each vertex $j \in T$ represents a task and each arrow $d \in D$ represents a data dependency between two tasks. For simplicity of further notation, let $\mathcal{J}$ be the set of all tasks composing the submitted workflows over the time. For each task is associated the following properties: (1) its needs in term of computing resources (such as the quantity of memory it requires or the number of CPUs, etc.); (2) its software requirements (basically the operating system, libraries and packages needed to execute the task); (3) its execution time represented by two metrics: for a task $j \in J$, μ_j is the number of instructions needed to complete j in average, and σ_j is the standard deviation of the execution time. Such information can be retrieved by sampling, and in this chapter are assumed to be known.

In addition, a weight representing the size of the data that has to be transferred from a task to another, denoted $d_{i \to j} \in D$ between $i \in T$ and $j \in T$. It can be noted that a task can transfer the same file to multiple successor tasks, and that a task can get multiple files from the same ancestor task.

Each workflow is associated to a user. This user is the only one that can have access to the files created by the tasks of the workflow. Therefore isolation must be guaranteed, and for this reason, the VM provisioned for the execution of the task of one user, cannot be use for the execution of the task of another user.

5.2.2 Infrastructure definition

The considered infrastructure is composed of multiple nodes (physical machines), denoted $n \in \mathcal{N}$, where $\mathcal{N}$ is the set of all nodes. For each node are attached some properties: (1) its computing capabilities, (the quantity of memory, the number of CPUs, etc.); (2) its speed represented by the processor frequency (per CPU), denoted $speed_n^{\mathcal{N}}$ for $n \in \mathcal{N}$. Let $bw_{n \leftrightarrow m}^{\mathcal{N}}$ be the bandwidth between two nodes n and $m \in \mathcal{N}$.

Unlike the VM of the Chapter 4 where the VMs where provisioned according to VM templates, any VM can be provisioned on nodes, with any size and with any given operating system. Let $\mathcal{V}$ be the set of all VMs launched on the nodes during the execution from the start. For each vm $v \in \mathcal{V}$, are attached two properties: (1) its hardware capacities, in term of VCPUs and memory; (2) its software capacities embedded in its image, basically the installed operating system, packages and libraries. It is ensured that on a given node, at every moment, the number of VCPUs and memory used by the hosted VMs does not exceed its capacities. As VMs are used to execute tasks, it is also ensured that at every moment the sum of the needs of the hosted tasks does not exceed the capacities of that VM. In other words, in this chapter over-provisioning is forbidden on both nodes and VMs levels. Such capacity constraints can be modeled as a classical bin packing problem. A more formal definition of this problem has been presented in the previous chapter 4.2. As over-provisioning is not allowed in this work the deterioration of VM computing speed is assumed to be low [101], and considered to be 5% of the host node speed. Consequently as every task is running in a VM, for simplicity, we will consider that the speed of all node is actually 95% of their real speed.

5.2.3 Scheduling problem

Unlike the scheduling problem resolved in the Chapter 4 that was a static problem, that consisted of scheduling a set of workflows submitted at the same time, the problem addressed in this chapter is dynamic. As presented in the related work chapter (see Section 3.1.2), a dynamic problem is a problem that may evolve over time, and therefore the planning (solution of the problem) must evolve accordingly to still be optimal. Indeed, new submitted workflows may have an higher priority than already scheduled ones - tighter deadline - and therefore the computed planning must be rethought. In the problem addressed in this chapter workflows can be submitted by users at any moment. A workflow $w \in \mathcal{W}$ is submitted at a given time $submit_w$, with a deadline denoted $dead_w^{\mathcal{W}}$, by a given user $user_w$.

Let $host_j^{\mathcal{J}}$ be the node hosting the VM that execute the task $j \in \mathcal{J}$, and $host_v^{\mathcal{V}}$ be the node hosting the VM $v \in \mathcal{V}$. For all tasks $j \in \mathcal{J}$, are defined height temporal notations :

Notation	Meaning
$ES_j^{\mathcal{J}}$	The estimated start time of the task
$EE_j^{\mathcal{J}}$	The estimated end time of the task
$EL_j^{\mathcal{J}}$	The estimated length of the task
$ETT_{i\to j}^{\mathcal{J}}$	The estimated transfer time between $i, j \in \mathcal{J}$
$AS_j^{\mathcal{J}}$	The actual start time of the task on the node $n \in \mathcal{N}$
$AE_j^{\mathcal{J}}$	The actual end time of the task on the node $n \in \mathcal{N}$
$AL_j^{\mathcal{J}}$	The actual length of the task
$ATT_{i\to j)}^{\mathcal{J}}$	The actual transfer time between $i, j \in \mathcal{J}$

Let $succ_j \subset \mathcal{J}$ be the set of tasks that succeed $j \in \mathcal{J}$, also called the set of successors. For each task $j \in \mathcal{J}$:

$$\forall p \in succ_j, ES_p^{\mathcal{J}} \geq EE_j^{\mathcal{J}} + ETT_{j\to p}^{\mathcal{J}} \tag{5.1}$$

As one may have noticed the execution time of a task follows a Gaussian distribution $\mathcal{N}(\mu, \sigma)$, unlike the tasks presented in Chapter 4. Indeed, in this chapter the task executions are considered uncertain, and may vary depending on different factors (*e.g.*, depending on input data). Thus a distinction between the estimated times and the actual times of the tasks is made in this chapter, and was not made in Chapter 4.

Representing the uncertainty of the task execution times by a Gaussian distribution is a common approach in the state of the art [48, 58, 88]. The estimated execution time of a task $j \in \mathcal{J}$ with the Gaussian distribution $\mathcal{N}(\mu, \sigma)$, on a node $n \in \mathcal{N}$ with the uncertainty $x \in]0; 1[$ is as follows:

$$F_x(\mu, \sigma) = \int_{-\infty}^{x} \frac{e^{\frac{-(x-\mu)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}} \tag{5.2}$$

$$EL_j^{\mathcal{J}} = \frac{F_x(\mu_j, \sigma_j)}{speed_{host_j^{\mathcal{J}}}^{\mathcal{N}}} \tag{5.3}$$

$$EE_j^{\mathcal{J}} = ES_j^{\mathcal{J}} + EL_j^{\mathcal{J}} \tag{5.4}$$

The estimated transfer time is defined according to the following equation :

$$ETT^{\mathcal{J}}_{j\to i} = \frac{d_{j\to i}}{bw^{\mathcal{N}}_{host^{\mathcal{J}}_j \to host^{\mathcal{J}}_i}} \tag{5.5}$$

Likewise, after a set of experiments on a real infrastructure we have observed that the VM boot time is uncertain. Indeed, the boot time of a VM is not static and can differ depending on the stress that is applied to different resources of the node that is hosting it. This observation has been highlighted in many works [97, 134, 136]. Thus, we assume that the VM boot time follows a Gaussian distribution composed of two properties, $\mu_{v,n}$ and $\sigma_{v,n}$, respectively the mean time and the standard deviation of the boot of $v \in \mathcal{V}$ on a node $n \in \mathcal{N}$. These two properties depend on the operating system of a given VM, and the node that host it. Furthermore, they can be retrieved by sampling, and are assumed to be known in this chapter. One can note that these properties are more static than the task execution time uncertainty, as possible bootable operating systems are bounded and determined by the Cloud provider, while tasks are unknown and chosen by users. Consequently, the administrator can run benchmarks for each OS to get the average boot times of a VM, and its standard deviation.

For each VM $v \in \mathcal{V}$, three different temporal notations are introduced :

Notation	Meaning
$EP^{\mathcal{V}}_v$	The estimated instant of the VM provisioning
$ES^{\mathcal{V}}_v$	The estimated ready time of the VM
$EE^{\mathcal{V}}_v$	The estimated end time of the VM

Let $\mathcal{J}_v \subset \mathcal{J}$ for $v \in \mathcal{V}$ be the set of tasks to execute on v, and $host^{\mathcal{V}}_v \in \mathcal{N}$ the hosting node for v. For each VM $v \in \mathcal{V}$, the following temporal constraints are defined, with $host^{\mathcal{V}}_v \in \mathcal{N}$ the node hosting the VM v:

$$ES^{\mathcal{V}}_v = \min_{j\in\mathcal{J}_v}(ES^{\mathcal{J}}_j) \tag{5.6}$$

$$EE^{\mathcal{V}}_v = \max_{j\in\mathcal{J}_v}(EE^{\mathcal{J}}_j) \tag{5.7}$$

$$EP^{\mathcal{V}}_v = ES^{\mathcal{V}}_v - F_x(\mu_{v,host^{\mathcal{V}}_v}, \sigma_{v,host^{\mathcal{V}}_v}) \tag{5.8}$$

5.2.4 Fairness objective

One of the objective of this work is to ensure the fairness between the different users of the workflow platform. The fairness is represented by the respect of the deadline of each workflow, and thus this objective can be defined as the minimization of the maximum of the deadline violations. The deadline violation of a workflow is defined by Equation 5.10 and the associated objective by Equation 5.11. The deadline violation is computed with the deviation, that is equal to the makespan of the workflow (maximum actual end time of the tasks of the workflow) minus the deadline. The value obtained by Equation 5.12 will be used in the evaluation section as a comparison metric, however the maximization of this metric is not an objective of this contribution.

$$deviation_w = \max_{j \in T_w} (AE_j^{\mathcal{J}}) - dead_w^{\mathcal{W}} \tag{5.9}$$

$$violation_w = \begin{cases} deviation_w & If\ deviation_w > 0 \\ 0 & Otherwise \end{cases} \tag{5.10}$$

$$Minimize(\max_{w \in \mathcal{W}}(violation_w)) \tag{5.11}$$

$$ahead_w = \begin{cases} deviation_w & If\ deviation_w < 0 \\ 0 & Otherwise \end{cases} \tag{5.12}$$

where $T_w \subset \mathcal{J}$ is the set of tasks of the workflow w, and $AE_j^{\mathcal{J}}$ is the actual end time of the task $j \in T_w$.

5.2.5 Energy objective

We have seen in the context chapter (cf. Section 2.6.2) that the energy consumption of a node is not linear to its load, and is more energy efficient when highly loaded than low loaded (see Equation 4.6). For this reason, to optimize the energy consumption, in this work the objective is to use the nodes at the maximum of their capacity by regrouping the tasks on the same nodes.

Symbol	Definition
Workflow	
$w \in \mathcal{W}$	A workflow
$j \in \mathcal{J}$	A task
$T_w \subset \mathcal{J}$	The list of tasks of the workflow $w \in \mathcal{W}$
$d_{i \to j} \in \mathcal{D}$	A dependency between two tasks i and $j \in \mathcal{J}$
$dead_w^{\mathcal{W}}$	The deadline of the workflow $w \in \mathcal{W}$
$succ_j$	The list of successor tasks of the task $j \in \mathcal{J}$
$pred_j$	The list of predecessor tasks of the task $j \in \mathcal{J}$
$submit_w$	The instant of sumbission of the workflow $w \in \mathcal{J}$
$user_w$	The user to whom the workflow $w \in \mathcal{W}$ belongs
Execution environment	
$n \in \mathcal{N}$	A node
$v \in \mathcal{V}$	A VM
$bw_{n \leftrightarrow m}^{\mathcal{N}}$	the bandwidth between the node n and $m \in \mathcal{N}$
$speed_n^{\mathcal{N}}$	the speed of the node $n \in \mathcal{N}$
Workflow temporal notations	
$deviation_w$	The deviation compared to the deadline of the workflow $w \in \mathcal{W}$
$violation_w$	The deadline violation of the workflow $w \in \mathcal{W}$
$ahead_w$	The ahead time of the workflow $w \in \mathcal{W}$
Task temporal notations	
$ES_j^{\mathcal{J}}$	The estimated start time of the task
$EE_j^{\mathcal{J}}$	The estimated end time of the task
$EL_j^{\mathcal{J}}$	The estimated length of the task
$ETT_{i \to j}^{\mathcal{J}}$	The estimated transfer time between $i, j \in \mathcal{J}$
$AS_j^{\mathcal{J}}$	The actual start time of the task
$AE_j^{\mathcal{J}}$	The actual end time of the task
$AL_j^{\mathcal{J}}$	The actual length of the task
$ATT_{i \to j}^{\mathcal{J}}$	The actual transfer time between $i, j \in \mathcal{J}$
VM temporal notations	
$EP_v^{\mathcal{V}}$	The estimated instant of the VM provisioning
$ES_v^{\mathcal{V}}$	The estimated ready time of the VM
$EE_v^{\mathcal{V}}$	The estimated end time of the VM
Assignment	
$host_v^{\mathcal{V}}$	The node hosting the VM $v \in \mathcal{V}$
$host_j^{\mathcal{J}}$	The node hosting the VM executing the task $j \in \mathcal{J}$

Figure 5.1 – Table of symbols of the model of section 5.2

5.3 Deadline based dynamic algorithm

In this section, is introduced our new deadline-aware scheduling algorithm NearDeadline. Our algorithm is built such that workflows belonging to different users - each associated to a deadline - can be submitted at any time to the workflow execution platform. The algorithm is launched when new submissions are performed, with a configurable submission window (*i.e.*, schedule all the workflows that have been submitted during a window of t seconds). In the literature, two types of algorithm exist in order to manage a dynamic problem: dynamic algorithms and online algorithms. In this contribution we have opted for a dynamic algorithm, for the reasons that are discussed in Section 3.1.5 (unwanted synchronizations).

Let us recall the two definitions: (1) the *workload* is the set of all submitted workflows; (2) an *expected planning* is generated by the scheduling algorithm and contains all the tasks, files and VMs running or expected to run in the future. It is similar to a planning (cf Section 4.3) with the exception that it may be updated by the scheduler. The algorithm takes into account the current *expected planning* and updates it, by adding the new submitted workflows, and reconsidering the already scheduled ones.

5.3.1 Priorities and deadlines

When a new workflow is submitted to the platform, the algorithm first computes its difficulty based on its deadline and its estimated execution time computed from its critical path (*i.e.*, set of tasks responsible for its overall execution time). The difficulty of a workflow is computed by Equation 5.13. The workflow with the lowest rank is handled first.

$$rank_w^{\mathcal{W}} = dead_w^{\mathcal{W}} - \max_{j \in T_w}(priority_j^{\mathcal{J}}) \tag{5.13}$$

$$priority_j^{\mathcal{J}} = \overline{ET_j^{\mathcal{J}}} + \max_{p \in succ_j} \left(\overline{ETT_{j \to p}^{\mathcal{J}}} + priority_p^{\mathcal{J}} \right) \tag{5.14}$$

The priority of a task presented in Equation 5.14, is used to compute the execution time of the critical path of the workflow. For each task $j \in \mathcal{J}$ its priority corresponds to the partial execution time of the workflow from this task. It is established according to the average completion time of the task j on all possible nodes, denoted $\overline{ET_j^{\mathcal{J}}}$, as computed in Equation 5.15, as well as its average communication time (between all possible

nodes) denoted $\overline{ETT^{\mathcal{J}}_{j \to p}}$, for $j, p \in \mathcal{J}$. One may note that the priority computed by NearDeadline is the same as the rank computed by the HEFT algorithm as presented in Section 4.3.

$$\overline{ET^{\mathcal{J}}_{j}} = \frac{\sum_{n \in \mathcal{N}} \frac{F_x(\mu_j, \sigma_j)}{speed^{\mathcal{N}}_n}}{|\mathcal{N}|} \tag{5.15}$$

After computing the rank of each submitted workflow (if multiple workflows are submitted at the same time), the NearDeadline algorithm sorts them by increasing rank, and schedule them one by one. As presented in Algorithm 6, the scheduling of a workflow is made in two different phases, the first one tries to schedule each task of the workflow, as near as possible to their respective deadline. This first scheduling is the function scheduleWorkflowND. In the second phase, if this schedule fails, the workflow moves in panic mode and will be scheduled in best effort. The rest of this section follows these two phases.

Algorithm 6 NearDeadline algorithm

```
function NearDeadline(workflows, node)
    Q ← workflows
    while |Q|! = 0 do
        panic ← None
        Q ← SortByRank(Q)
        for w ∈ Q do
            if not scheduleWorkflowND(T_w, nodes) then
                Invalidate(j), ∀ j ∈ T_w
                panic ← w
                break
        if panic ≠ None then
            (Q', P') ← RemoveAllUnder(panic, nodes)
            P ← SortByRank(panic + P')
            Q ← Q + Q'
            for all w ∈ P do
                scheduleTaskBestEffort(T_w, nodes)
```

5.3.2 Scheduling near the deadline

The first scheduling that is performed on a workflow is made by the function scheduleWorkflowND, presented in Algorithm 7. The idea is to free resources as most as possible before the deadline in case of future submissions with shorter deadlines. The Figure 5.2 shows an example illustrating this idea, where a first workflow is submitted with

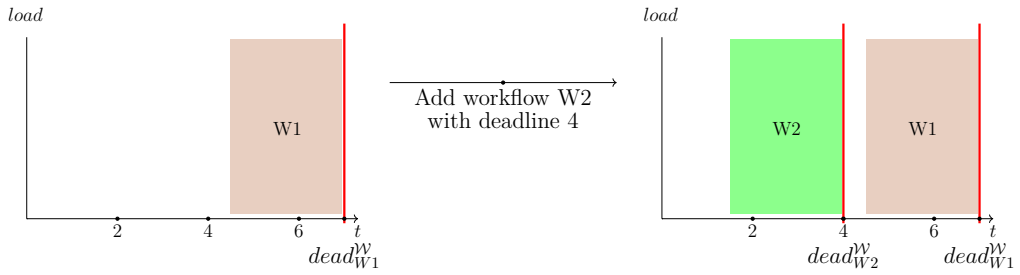


Figure 5.2 – Scheduling near the deadline to have free resources

a smooth deadline, and afterwards a second workflow is submitted with a more difficult deadline. The y-axis models an imprecise vision of the load on the infrastructure as the percentage of used resources. One can note in this example that the current workload doesn't need to be reconsidered to place the new workflow.

Another idea could have been to schedule the workflows using a small amount of resources, as presented in the Figure 5.3. But as discussed in Section 5.2, the energy consumption is not linear to its executing load. Thus, using this idea would lead to under used nodes during long period of time, which would be much worse in term of energy consumption than using nodes during small period and leave them in idle mode the rest of the time.

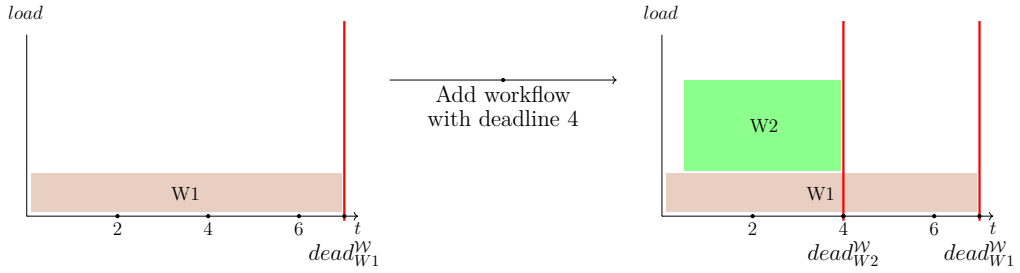


Figure 5.3 – Scheduling with low resources to have free resources

Algorithm 7 Single workflow scheduling near its deadline

```
function scheduleWorkflowND(tasks, nodes)
    tasks ← SortByBackwardRank(tasks)
    for all j ∈ tasks do
        if not scheduleTaskND(j, nodes) then return False
    return True
```

To perform this scheduling, tasks need to be sorted in backward order (starting by the exit tasks), therefore sorted by backward priority (function SortByBackwardRank). The backward priority of a task is computed the same way as the forward priority, but by considering the predecessors instead of the successors. Equation 5.16 shows the calculation of the backward priority of a task $j \in \mathcal{J}$, where $pred_j$ is the list of predecessors tasks of j :

$$bpriority_j^{\mathcal{J}} = \overline{ET_j^{\mathcal{J}}} + \max_{p \in pred_j} \left(\overline{ETT_{j,\to p}^{\mathcal{J}}} + bpriority_p^{\mathcal{J}} \right) \tag{5.16}$$

Algorithm 8 Single task scheduling near its deadline

```
function scheduleTaskND(task, nodes)
    loc ← None
    for all s ∈ nodes do
        loc_s ← scheduleTaskOnNodeND(j, node)
        if isBestND(loc_s, loc) then                    ▷ Eq. (5.11)
            loc ← loc_s
    if loc not None then
        Reserve(loc)
        return True
    else return False
```

The function scheduleTaskND, detailed in Algorithm 8, reserves a location for a task on a node. The function scheduleTaskOnNodeND, presented in Algorithm 9, is launched on each node for a given task. For a node $n \in \mathcal{N}$ and a task $j \in \mathcal{J}$, this function returns a location, on a VM $v \in \mathcal{V}_n$, where $\mathcal{V}_n \subset \mathcal{V}$ is the list of VM hosted on node n. The returned location is the one that optimizes the local objective evaluated by the fitness function isBestND, which will be presented in the subsection on fitness functions. The goal of this fitness function is to choose the location that is closest to the deadline.

$$dead_j^{\mathcal{J}} = \begin{cases} dead_w^{\mathcal{W}} & If\ |succ_j| = 0 \\ \min\limits_{p \in succ_j} \left(dead_p^{\mathcal{J}} - \min\limits_{n,m \in \mathcal{N}} \left(\frac{F_x(\mu_p, \sigma_p)}{speed_m^{\mathcal{N}}} + \frac{d_{j \to p}}{bw_{n \leftrightarrow m}^{\mathcal{N}}} \right) \right) & Otherwise \end{cases} \tag{5.17}$$

In the function scheduleTaskOnNodeND, one can note the called function scheduleTaskOnVMND. This function retrieves the first location where the task can be placed without overcharging the capacity of the VM. This place is searched between the zero instant of the current schedule and the deadline $dead_j^{\mathcal{J}}$ of the task $j \in \mathcal{J}$, of the workflow $w \in \mathcal{W}$ (that is calculated using Equation 5.17). This function, detailed in Al-

gorithm 10, is also in charge of the VM dimensions. One may note that a VM can be resized if it is not currently powered on, and therefore the capacity of the node must be taken into account in this function in order to not overcharge it. The resizing of the VM is multi dimensional, it can be either on capacity, and temporal aspects.

Algorithm 9 Single task scheduling near its deadline on one node

```
function scheduleTaskOnNodeND(j, n)
    EL_j^J ← ComputeLenOfTask(j, n)
    loc ← None
    zero ← currentTimeStamp
    for all v ∈ V_n do
        if user_v = user_j and canRun(v, j) then
            loc_v ← scheduleTaskOnVMND(j, v, zero, EL_j^J)        ▷ Eq. (5.8,5.6,5.7)
            if isBestND(loc_v, loc) then                          ▷ Eq. (5.11)
                loc ← loc_v
    if loc is None then
        new_v ← EmptyVM(os_j, user_j, n)
        loc ← scheduleTaskOnVMND(j, new_v)                        ▷ Eq. (5.8,5.6,5.7)
    return loc
```

Algorithm 10 Single task scheduling near its deadline on one VM

```
function scheduleTaskOnVMND(j, v, zero, EL_j^J)
    LST ← dead_j^J − EL_j^J                                       ▷ Last Start Time, Eq. (5.17)
    bootTime ← ES_v^V − EP_v^V                                    ▷ Eq. (5.8)
    nodeUsage ← NodeUsageWithoutVM(host_v^V, v)
    nodeCapas ← NodeCapacities(host_v^V)
    MES ← ComputeMinStart(j, host_v^V, pred_j)                    ▷ Eq. (5.18)
    zero ← max(zero, MES)
    position ← LST
    while position > zero + bootTime do
        loc ← (position, position + EL_j^J)
        over ← CollisionOnVM(loc, v, nodeUsage, nodeCapas)
        if over is None then
            return loc
        else
            position ← over − EL_j^J
    return None
```

As can be seen, the value MES is computed in scheduleTaskOnVMND. This value is the minimal estimated start time of the task and is computed by Equation 5.18. Its utility will be explained later, as it requires information given in the panic subsection. The value *zero* on the other hand, is the instant of the current execution of the algorithm. CollisionOnVM is the function in charge of checking if the VM, when adding the

new task, does not exceed the capacity of the node. There are two different types of VM collisions. The first one is when a VM cannot be resized in capacity terms, and an example is illustrated in the Figure 5.4. The second case is when the VM cannot be extended in time because of node overcharge. It is illustrated in Figure 5.5. CollisionOnVM handles multi-dimensional capacities (VCPUs, vram, disk space, ...), but for simplicity of explanation, the two figures represent VCPU resources only.

$$MES_j = \max_{p \in pred_j} (EE_p^{\mathcal{J}} + ETT_{p \to j}^{\mathcal{J}}) \tag{5.18}$$

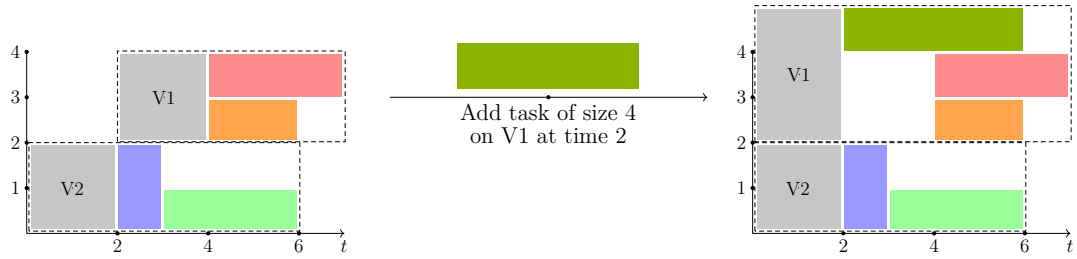


Figure 5.4 – Collision when increasing the capacities of a VM on a server with 4 CPUs

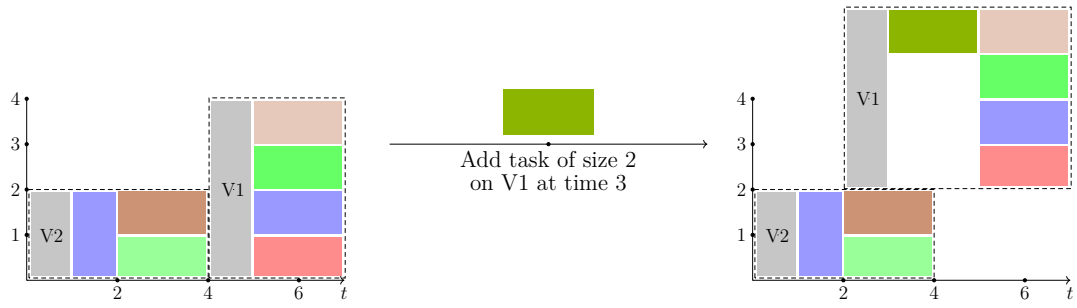


Figure 5.5 – Collision when increasing the length of a VM on a server with 4 CPUs

Let the first case scenario illustrated in the Figure 5.4, be a *capacity collision*, as the collision occurs inside the VM, and changes the VM capacities; and the second case scenario be a *temporal collision*, as the collision is external to the VM and is due to a duration change on the VM. In the case of a capacity collision, the function CollisionOnVM returns the exact location of the collision inside the VM (in the example, the instant 4). In the case of an temporal collision, the whole VM is considered and the function returns the location of the collision at the node level (in the example, the instant 2). It can be noted that the VM does not always provide a possible location as they are constrained. If

none of the currently available VMs hold a valid location, a new empty VM is considered (function EmptyVM, called in Algorithm 9).

5.3.3 Panic mode

When required resources are available on the infrastructure, the previous algorithm scheduleWorkflowND detailed in the previous subsection performs the scheduling of a workflow near its deadline. However, it may happen that scheduling the workflow in time is impossible with the available remaining resources. In that case the considered workflow moves in panic mode, as presented in Algorithm 6.

The function RemoveAllUnder, is an important part of NearDeadline. Since there is no remaining resource to schedule the workflow in time, resources must be released in the current *expected planning* to be able to find a suitable place for the new workflow. The function RemoveAllUnder explores all the nodes and removes all the non-running tasks belonging to workflows that are less urgent than the new *panic* workflow. The tasks that are currently running, though, are not interrupted for energy and efficiency reasons. We are aware that it can cause missed deadlines when dealing with long-term tasks. However, it would not be difficult to add a system to handle this case by computing the ratio between the remaining execution time and the priority of the tasks, and it would be a possible orientation for a future work. When the function RemoveAllUnder ends, it may happen that some running or booting VMs do not contain tasks any longer. In that case, these VMs are killed, even if they have not been useful yet.

Finally, the function RemoveAllUnder returns two lists of workflows, a list P' containing all the removed workflows from the *expected planning* that have already been scheduled in *panic* mode, and the list Q' containing all the other removed partial workflows. It should be noted that those workflows (for both P', and Q') might be incomplete, as the tasks that are finished and the tasks that are currently running are not rescheduled, or only a subset of the workflow is less urgent than the *panic* workflow. That is to say, when the list Q is refilled, some tasks have ancestor that are still in the *expected planning*, and therefore it explains why the value MES must be computed in the function scheduleTaskOnVMND. Each workflow in the list P is then scheduled in best effort mode, using the function ScheduleTaskBestEffort.

The function ScheduleTaskBestEffort is close to the HEFT algorithm of the related work, but with three main differences: (1) it is adapted to take into account virtual resources; (2) it plans workflows one by one instead of mixing all the tasks of all workflows

Algorithm 11 Single task scheduling in best effort on one VM

```
function scheduleTaskOnVMBE(j, v, zero, EL_j^J)
    MES ← ComputeMinStart(j, pred_j)
    bootTime ← ES_v^V − EP_v^V
    nodeUsage ← NodeUsageWithoutVM(host_v^V, v)
    nodeCapas ← NodeCapacities(host_v^V)
    position ← max(zero, MES)
    while True do
        loc ← (position, position + EL_j^J)
        over ← CollisionOnVM(loc, v, nodeUsage, nodeCapas)
        if over is None then
            return loc
        if isTemporalCollision(over) and (position + EL_j^J) >= EE_v^V then
            return None
        else
            position ← over − EL_j^J
```

and then executing the scheduling; (3) it uses the fitness function isBestBE to choose between multiple possible locations. The function scheduleTaskOnNodeBE, is used in ScheduleTaskBestEffort to find a location on a node for a task.

The function scheduleTaskOnNodeBE, is almost equivalent to the function scheduleTaskOnNodeND. There is only two differences. First, instead of isBestND the fitness function isBestBE is used, this function will be detailed in the subsection on fitness function. The second difference is the call to the function scheduleTaskOnVMBE, instead of the function scheduleTaskOnVMND. This function retrieves the first location where the task can be placed without overcharging the capacity of the VM. Unlike the function scheduleTaskOnVMND, for a task $j \in \mathcal{J}$ and a VM $v \in \mathcal{V}$ this function minimizes the makespan (the minimal $EE_j^{\mathcal{J}}$), and hence searches the location starting by the minimal $ES_j^{\mathcal{J}}$ of the task, defined in Equation 5.19. Algorithm 11 presents this function.

$$MES_j = \max_{p \in pred_j} (EE_p^{\mathcal{J}} + ETT_{p \to j}^{\mathcal{J}}) \tag{5.19}$$

The function scheduleTaskOnVMBE, in Algorithm 11, stops its search when the VM is blocked and cannot grew in length.

5.3.4 Fitness functions

In the last subsections, two fitness functions isBestND and isBestBE were introduced. These two functions are used to select one location between two possibilities. These two fitness functions are heuristics that aim at maximizing the fairness, minimizing the energy consumption and minimizing the deadline violation. The function isBestND is used by the algorithm when scheduling a workflow near its deadline, and the function isBestBE is used when the algorithm schedules a workflow in panic mode.

A. isBestND

The first function, isBestND, is used during the first scheduling phase. This function is called by the scheduleTaskND function in order to choose between two locations within two different VMs. These two VMs can be located inside two different nodes, or inside the same node. As the energy consumption of a node is not linear to its CPU load, it is better to execute tasks in parallel on few nodes, and avoid over distribution of the workflows. The function isBestND computes a value that aims at comparing two VMs, and that is denoted $quality_v$. The value $quality_v$, of a VM $v \in \mathcal{V}$, can be computed in two different ways. The first way corresponds to the number of tasks that are located inside the VM. The second way is based on the number of VCPUs reserved by a VM. The first way will be denoted *nbTasks*, and the second will be denoted *nbVcpus*.

The function isBestND, chooses the VM that has the highest *quality* value, and when two VMs have the same *quality*, it chooses the location whose end time ($EE_j^{\mathcal{J}}$) is the closest to the deadline of the task. We assume that, as we are considering the execution time of the task in a pessimist way with the Gaussian distribution, if the scheduling returns a location that is before the deadline, the actual end time of the task will equally be before the deadline.

The idea is that when the first task of the workflow is scheduled (the exit task with the highest backward rank), a new VM will be inevitably considered. Each possible locations, will then be in VMs, with the same *quality*, then the location whose end is the closest to the deadline will be selected in the first place. Then, when the other tasks of the workflow are scheduled, the VM that is used for the first task would have the highest *quality*, so it will be selected, until it will no longer be able to guarantee a valid location, and therefore a new VM has to be considered. Thus new VMs are instantiated only when required, leading to a few number of VMs, and as we will see in Section 6.4, the energy consumption is mainly correlated to the number of VM instantiated. Indeed, The boot time of the VM

consumes a low amount of CPU resources, during a long period of time, even if many CPU resources are reserved by the VM. In addition, the higher the number of VMs, the more time is wasted by booting times, resulting in reduced fairness and deadline violations.

B. isBestBE

The second function isBestBE, is the fitness function used when dealing with best effort scheduling. As for the function isBestND, the function isBestBE relies on the *quality* of the VMs.

This function creates a vector $u = < EE_j^{\mathcal{J}} \times \alpha, \frac{1}{quality_v} >$, for a location of a task $j \in \mathcal{J}$ on the VM $v \in \mathcal{V}$. It then computes the distance of this vector to the vector $i = < 0, 0 >$, and chooses the location whose vector u has the lowest distance to i. The parameter α is a variable used to make vary the priority of the two objectives expressed by the makespan and the *quality* of a VM. We will see in Section 6.4, that if this parameter is too low, the algorithm will instantiate many VMs, leading to both high energy consumption and poor fairness, and that if this parameter is too high, the algorithm will creates few VMs leading to low distribution of the workflows execution and high deadline violation.

As two versions of the *quality* are defined, it is possible to define two versions of the fitness function isBestND and two versions of the function isBestND, and thus two versions of the algorithm NearDeadline. The first version using the *nbTasks quality* will be called NearDeadlineTask, and the second one using the *nbVcpus quality* will be called NearDeadlineVCpu.

5.4 Conclusion

This chapter tackles the problem of scheduling workflows of multiple users with random arrivals and uncertain task execution times, while minimizing the energy consumption of the Cloud infrastructure and maximizing the user fairness. The NearDeadline algorithm has been presented as a solution to this problem. The NearDeadline algorithm adds deadlines to the workflows chosen by users. These deadlines offer an opportunity to make a smart usage of the infrastructure at disposal while respecting the user initial wishes. In contrast to the use made of the deadline in OnlyUsedNodes, the deadline in NearDeadline is not only used for the energy optimization, but also for the user fairness objective. Indeed, by setting a deadline for every user, a fair planning is well defined. An evaluation of the NearDeadline algorithm and of the OnlyUsedNodes algorithm on a real infrastructure is presented in Chapter 6.

Chapter 6

Evaluation of scheduling algorithms

This chapter presents a detailed evaluation of the algorithms OnlyUsedNodes and NearDeadline respectively presented in Chapter 4 and Chapter 5. This evaluation is divided in two sections. The first part presents experimentation conducted via a simulator, that evaluates the algorithm OnlyUsedNodes. The second part presents experimentation conducted on a real infrastructure, with real executions of workflows, that firstly evaluates the algorithm OnlyUsedNodes and secondly the algorithm NearDeadline.

Contents

6.1 Simulation

In this section, we present a detailed evaluation of our OnlyUsedNodes algorithm. All evaluations are performed on realistic workflows generated by the pegasus workflow generator [1] [110]. The experimentations presented in this section have been conducted using a simulator.

6.1.1 Simple workload scheduling

This section presents the experimental results when considering an homogeneous cluster of physical machines, and when no tasks are already running on the cluster (*i.e.*, initial workload). This simplified use case offers the possibility to precisely analyze the behavior of OnlyUsedNodes without too many parameters to take into account. This section shows that introducing a deadline within HEFT is a way to optimize the number of machines used to execute a set of workflows, thereby reducing the energy consumption for the Cloud provider. This section also shows that the execution time of OnlyUsedNodes is very competitive and scalable compared to v-HEFT.

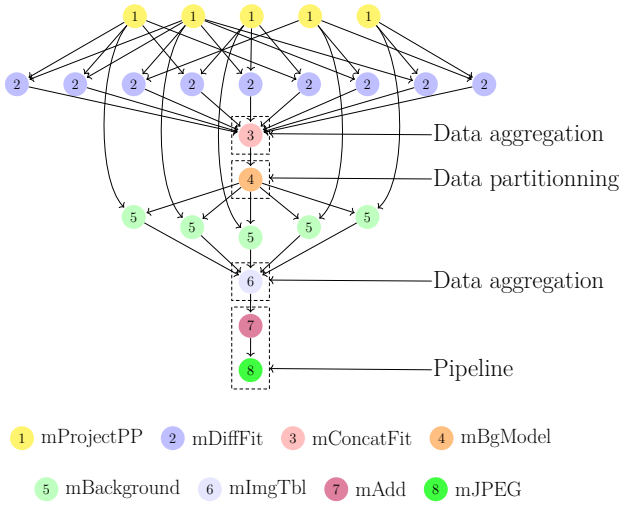


Figure 6.1 – Topology of the *Montage* workflow

1. `https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator`

Experimental setup

The *Montage* workflow depicted in Figure 6.1 is a typical case-study used to evaluate scheduling algorithms [21, 39]. It is a complex workflow that integrates most of the workflow classes characterized by Bharathi et al. in [37], and that assembles astronomical images from mosaic. Figure 6.2 presents an example of result that can be obtained with the *Montage* workflow. The simulated workload is composed of a variable number of *Montage* workflows composed of 25 tasks and the simulated infrastructure is composed of 20 homogeneous nodes with the hardware configuration of the Grid'5000 [13] Econome cluster (see the first row of Table 6.1). To correctly select a set of deadlines for our OnlyUsedNodes algorithm, we ran v-HEFT to get its approximate minimum makespan.

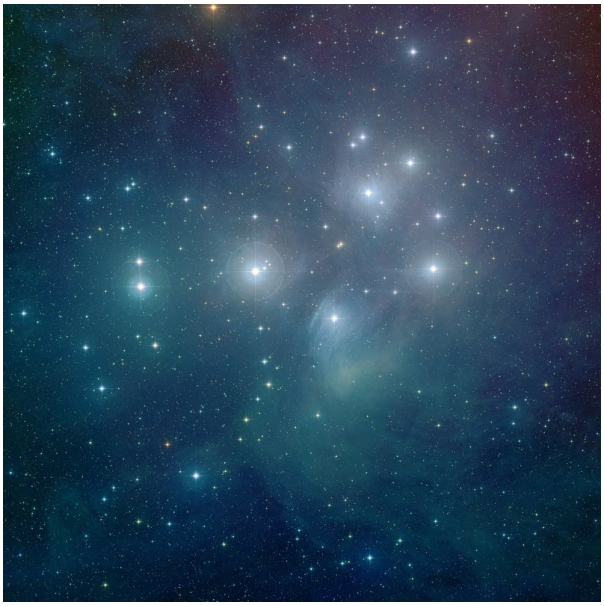


Figure 6.2 – Example of result computed by the *Montage* workflow

Using the Pegasus workflow generator, only one information is provided about a task: its execution time (in seconds). We assume in our experiments that this time was produced by a single CPU core with a computing capacity of 2GFlops (floating operations per second). This assumption is used to transform information on the execution time into a number of instructions (as required by our model). We also assume that each *Montage* task has been executed with a sufficient amount of memory, and we consider in our experiments that the memory requirements are always fulfilled. Furthermore, we consider a single VM template that uses 4 cores. Finally, the boot time of the VM template has been configured to 10 seconds on a 2 GFlops CPU core.

Table 6.1 – Description of the simulated nodes

Location	Name	Number of nodes	CPU	Network
Nantes	econome	22	Intel Xeon E5-2660 (Sandy Bridge, 2.20GHz, 2 CPUs/node, 8 cores/CPU)	10 Gbps
Rennes	parapide	21	Intel Xeon X5570 (Nehalem, 2.93GHz, 2 CPUs/node, 4 cores/CPU)	20 Gbps
Grenoble	yeti	4	Intel Xeon Gold 6130 (Skylake, 2.10GHz, 4 CPUs/node, 16 cores/CPU)	10 Gbps

Quality evaluation

Figure 6.3 represents the number of nodes used, respectively by V-HEFT and ONLYUSEDNODES, to schedule workloads with variable amounts of *Montage* workflows. ONLYUSEDNODES has been run with five different deadlines proportional to the makespan computed by V-HEFT: (1) D=1, (2) D=1.3, (3) D=1.5, (4) D=2, and (5) D=3. This result illustrates that relaxing the deadline offers more possibilities for ONLYUSEDNODES to reduce the number of nodes needed. As a result, when users can extend their deadlines, the Cloud provider can benefits from an energy consumption reduction. Obviously this cost reduction may be passed on the rental price for the users who then have interest in relaxing their deadlines. One can note that even by extending the deadline from 1 (V-HEFT) to 1.5 the number of nodes can be divided by 2. When the deadline is extended from 1 to 3, only five of the twenty available nodes are used to schedule the workloads while V-HEFT would use 20 machines.

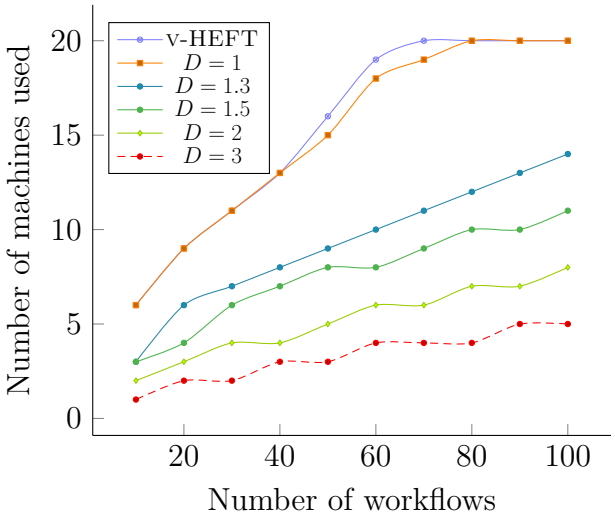


Figure 6.3 – Comparison of the number of nodes used between V-HEFT and ONLYUSEDNODES with four different deadlines D.

Energy consumption evaluation

Figure 6.4 shows estimated immediate power consumption (in Watt) of the nodes for the execution of 100 *Montage* workflows, scheduled by both v-HEFT and OnlyUsedNodes algorithm with different deadlines. This consumption has been estimated according to real power consumption measured on a cluster of Grid'5000 (Ecotype). One can note that, as expected, power consumption is reduced when the deadlines are delayed.

	v-HEFT	OnlyUsedNodes				
Deadline	-	×1	×1.3	×1.5	×2	×3
Nb nodes	20	20 (-0%)	14 (-30%)	11 (-45%)	8 (-60%)	5 (-75%)
Makespan	157	164 (×1.04)	200 (×1.27)	233 (×1.48)	311 (×1.98)	462 (×2.94)
Energy (Joules)	333,777	299,519 (-10.26%)	298,352 (-10.6%)	282,602 (-15.3%)	282,887 (-15.2%)	278,266 (-16.6%)

Table 6.2 – Results of the scheduling of complete workload composed of 100 *Montage* workflows, where the percentages are computed based on the v-HEFT results.

The cost for the Cloud provider is mainly correlated to the energy consumption (Joules). The energy consumption is the sum of power consumption (per second) during the execution of the workload. As already explained, it has been shown that energy consumption cannot be modeled by a linear function [74, 135] defined by the number of machines and the running time (see Equation 4.6). As a result, reducing the number of nodes may induce a reduction of the global energy consumption (*i.e.*, the cost) even if the time required to finish the workload is longer. Table 6.2 illustrates this claim by showing the results obtained for the five different deadlines when scheduling 100 workflows, in terms of the number of nodes used (the maximum number during the overall execution), the total makespan needed to run the workload, as well as the energy consumption. In this experiment, unused machines are considered powered off, and nodes are turned off when they have finished their computations. As expected, the makespan increases with the deadline by using OnlyUsedNodes, thus the overall time spent to execute the complete workload is longer than v-HEFT, but the deadlines of the workflows are respected. Moreover, the total energy consumption is almost reduced of 17% when using OnlyUsedNodes compared to v-HEFT. Therefore, the number of nodes has a direct impact on the energy consumption of the Cloud provider. One can note, though, that the execution with D=2, has a cost a bit higher than the execution with the D=1.5, meaning that the diminution of the number of nodes does not always counterbalance the makespan extension. When using a deadline equivalent to the makespan of v-HEFT, the energy consumption still decreases. This is due to the dispersion of the tasks induced by

the makespan minimization objective of the V-HEFT algorithm, that always chooses the location with the lowest makespan even if a location with an almost equivalent makespan was available but with a lower energy consumption. One can note from Figure 6.4, that at time $t = 100$, the energy consumption of ONLYUSEDNODES with $D = 1$ is far lower than the energy consumption of V-HEFT, indeed at time $t = 100$ only 11 nodes are being used by ONLYUSEDNODES instead of 20 for V-HEFT.

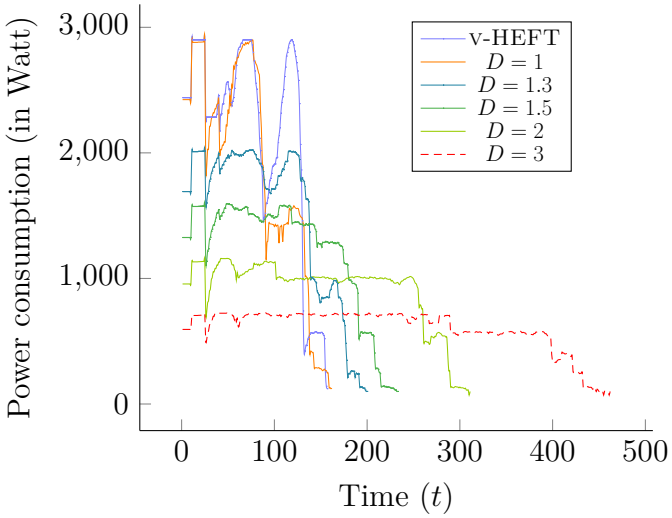


Figure 6.4 – Comparison of immediate power consumption between V-HEFT and ONLYUSEDNODES with five different deadlines D.

Performance evaluation

Figure 6.5 shows the execution time taken by both V-HEFT and ONLYUSEDNODES to compute the scheduling solution of the previous experiment. First, one can note that adding a backtrack system in ONLYUSEDNODES does not deteriorate the efficiency of the algorithm because ONLYUSEDNODES iterates on already used nodes instead of all nodes for V-HEFT.

We can compute the number of scheduling operations performed by each algorithm by using the complexity we introduce in Part 4.3.4. In the case of the scheduling of 100 workflows, for V-HEFT algorithm, the number of operations is $j \times n$, with $j = 100$ and $n = 20$, that is to say 2000 operations - by a factor of 25 which is the number of tasks forming the *Montage* workflow. For ONLYUSEDNODES algorithm, the number of operations depends on the number of nodes under usage when a workflow is scheduled. The

number of operations can be estimated by using the values returned by our experiments when scheduling a variable number of workflows (Figure 6.3). As a result, the number of scheduling operations needed with a deadline $D = 1.3$ is approximately 930, which is 46.5%, of the number of operations of V-HEFT algorithm. This number is validated by experiments where the time taken by OnlyUsedNodes algorithm (1.211 seconds) is half the time taken by V-HEFT algorithm (2.423 seconds). This result validates our claim about the complexity - that is to say that the complexity of the scheduling phase by only considering the already used nodes is in average far better than the worst case complexity - and that the execution speed of the algorithm is competitive in comparison with V-HEFT.

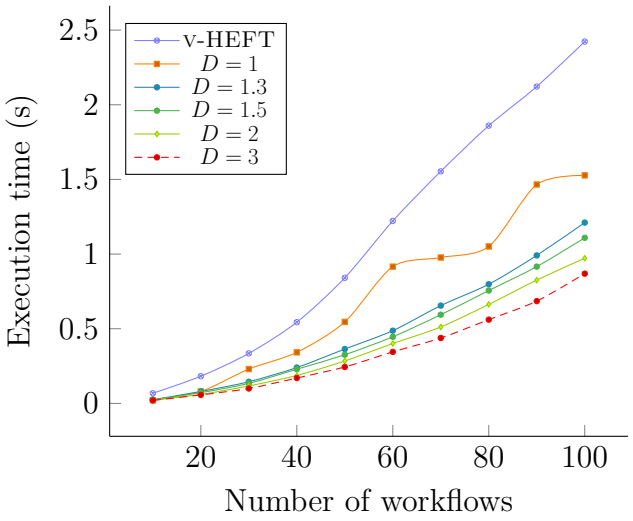


Figure 6.5 – Comparison of execution time between V-HEFT and OnlyUsedNodes with five different deadlines D.

6.1.2 Complex workload scheduling

In this section, we evaluate OnlyUsedNodes on a set of more complex scenarios from both workload and infrastructure perspectives. As for the first evaluation, the experimentations are simulated.

Experimental setup

First, we consider a multi-site infrastructure composed of three different clusters. This infrastructure is a subset of the Grid'5000 testbed that uses the Renater [2] network for communications, which allows inter-cluster communications with a bandwidth of 10 Gbps. Table 6.1 gives a description of the three clusters and their associated nodes.

Second, the considered type of workload is more heterogeneous in this experiment. Table 6.3 details the considered workloads composed of five different kind of workflows (all generated by the Pegasus workflow generator). We assume that, in a real-case, the percentage of complex workflows is less important than simple ones. Finally, benchmarks also use different deadlines for each kind of workflow. In the scenario A the deadlines are approximately the makespan returned by V-HEFT execution.

Name	%	A	B	C	D
Montage	30%	150	150	150	150
CyberShake	10%	1000	1000	1000	1000
Inspiral	6%	1500	2000 (×1.3)	2200 (×1.45)	2200 (×1.45)
Sipht	4%	4000	4000	4000	4000
Pipeline	50%	250	250	250	400 (×1.6)

Table 6.3 – Percentages and deadlines of the workflows composing the complex simulated workloads

	V-HEFT	A	B	C	D
Makespan	5202	5422	5422	5422	5431
Nb nodes	47	35 (-26%)	35 (-26%)	36 (-23%)	34 (-28%)
Time (s)	27.258	9.601 (-64.8%)	9.705 (-64.4%)	10.185 (-62.6%)	9.338 (-65.7%)
Energy (Joules)	9,183,968	8,395,508 (-8.6%)	8,487,680 (-7.6%)	8,555,411 (-6.8%)	8,276,282 (-9.9%)

Table 6.4 – Results of the scheduling of the different simulated scenarios, where the percentages are computed based on the results of the V-HEFT execution.

In the following experiments, each set of workflows are owned by a specific user, *i.e.*, all the pipeline workflows belongs to the first user, *Montage* workflows belongs to the second one, etc. The four workloads presented in the Table 6.3 are submitted twice, meaning that the scheduling algorithm is called two times. Thus, tasks under execution are taken into account in this experiments. The first submission is done at $t = 500$, and the second one at $t = 1500$. Finally, at $t = 0$ half of the nodes are powered on and are half used. This simulated workload ends at $t = 1000$.

2. `https://pasillo.renater.fr/weathermap/weathermap_metropole.html`

Evaluation

Figure 6.6 shows the estimated immediate power consumption (in Watt) of the nodes for the execution of the different scenarios (A to D). As for the first experiment 6.1.1, the unused nodes are considered powered off and nodes are powered off as soon as they have finished their tasks. Table 6.4 shows the results obtained for the four different scenarios in terms of the number of nodes, the total makespan and the energy consumption needed to run each workload twice. Moreover, the execution time of the scheduling algorithm is given.

As expected, the global makespan of the solution returned by our algorithm is less good than the makespan of the V-HEFT solution. However, the makespan minimization is not the optimization objective of ONLYUSEDNODES. One can observe that even in the scenario A, where the deadlines are almost the makespan returned by V-HEFT, the number of nodes used to schedule the workload is reduced. Actually, as V-HEFT minimizes the makespan for each task it favors empty nodes that offers the best makespan even when already used nodes offers almost the same makespan. As a result, the nodes are not used at their full capacity (under used). In contrast, our algorithm will favor nodes that are already used, and will smooth the power consumption of the nodes over the time as we can observe in Figure 6.6.

In the scenarios B and C, one can note that the number of nodes used to schedule the workload is greater than in the scenario A, despite the deadline relaxing of *Inspiral* workflows. This effect is due to the scheduling at two different times. Indeed, as the first schedule is not aware of the second one, the resources usage of the Inspiral workflows will be smoothed and the local makespan of their execution will exceed the arrival of the second scheduling. This involves a lower number of available resources for the scheduling of the second set of workflows at $t = 1500$, and causes an increase on the number of needed nodes. The last scenario D validates this observation. In this scenario the deadlines of the *Pipeline* workflows are relaxed to give them more time to be executed even if the resources are used by *Inspiral* workflows. As expected, the number of used resources is reduced, and the global makespan delayed.

As in the simple evaluation, the total energy consumption observed is directly correlated to the number of nodes used to schedule the workload. In the scenarios B and C, the energy consumption is a bit higher than in A, for the same reasons than explained earlier. In all cases, the energy consumption is reduced compared to the V-HEFT (Table 6.4).

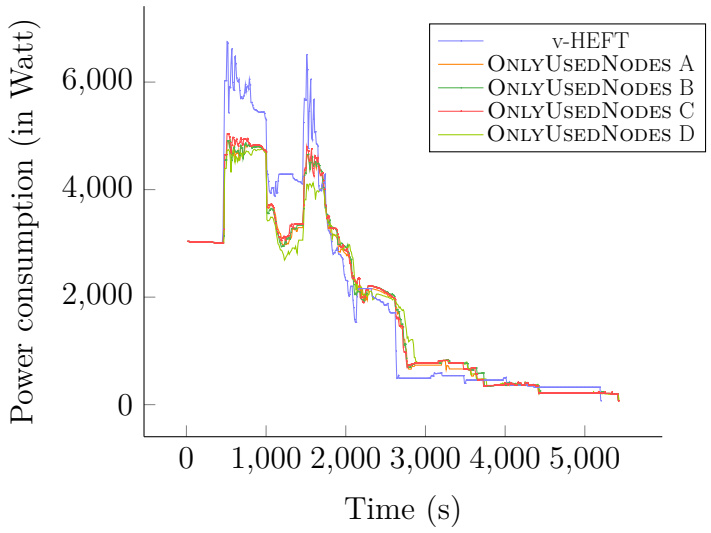


Figure 6.6 – Comparison of the power consumption through time between the different simulated scenarios of Table 6.3.

In the best case, the diminution of the consumption is up to almost 10%, which can be translated to a diminution of the cost for the Cloud provider. We have seen in the context chapter that consumption of PMs represents 35% of the total energy consumption of datacenters, and that in 2014, the energy consumption of US datacenters was estimated to 70 billion kWh (See Section 2.6). By doing a very rough calculation, with a kWh at 0.11€ (average price of a kWh in France in 2020 [3]), 10% of gain in energy consumption can be translated into 270 millions of euros. Even if we are aware that workflow execution represent a small portion of the total usage of datacenters, such diminution could have a significant impact on the cost for the Cloud provider.

6.2 Execution on real infrastructure - Environment description

To evaluate the performances of the OnlyUsedNodes and the NearDeadline algorithms, we have conducted experiments on a real infrastructure. Those experiments were performed on the SeDuCe platform [103], a scientific testbed integrated to Grid'5000 [13], that monitor the electrical consumption of the nodes described in Table 6.5. This section

3. `https://www.edf.fr/sites/default/files/contrib/entreprise/cgv-prix-de-marche/2020/fichecre_tarif_bleu_non_residentiel_1er_aout_2020_.pdf`

presents the execution setup of the experimentation that will be presented in the rest of this chapter.

6.2.1 Infrastructure

The infrastructure used in all our evaluations is the Ecotype cluster of the experimental platform Grid'5000, presented in Table 6.5. All the VMs launched during the experiments use the Ubuntu 18.04 operating system, with the hypervisor KVM. The image is replicated on each compute node. Unlike our simulated experiments, in all following evaluations the nodes are powered on at any time, thus consuming at least the idle consumption.

Location	Name	Number of nodes	CPU	Memory	Storage	Network
Nantes	ecotype	48	Intel Xeon E5-2630L v4 1.80GHz, 2 CPUs/node, 10 cores/CPU	128 GiB	400 GB SSD	2 x 10 Gbps

Table 6.5 – Description of the PMs of the real infrastructure used in the evaluation

6.2.2 Execution platform

For all evaluations we implemented a platform by using the library Scala Akka. This platform of execution is a simplified version of the WaaS (a centralized version) that is presented in Chapter 7. The architecture of the platform is modular and allows to indifferently choose one scheduling algorithm or another. Therefore algorithms comparisons can be performed in fair conditions.

The platform is based on a Master Worker architecture with three modules: MASTER, SCHEDULER and WORKER. Each node runs an instance of the WORKER daemon that is responsible for the provisioning of VMs on the node, and that answers simple orders such as starting a task on a given VM, downloading a file from another node, etc. The MASTER is in charge of applying the *configurations* that are transmitted to it by the SCHEDULER. A configuration is the set of tasks, VMs and files that are currently running or need to be launched or sent immediately. A configuration unlike a planning does not contain information about the future. In order to apply a configuration, the MASTER sends orders to the WORKERS. The MASTER is the entry point of the platform, meaning that the users submit new workflows to it. The file transfers are performed at the node level instead of the VM level, as it as been explained in Section 4.2.4 of Chapter 4. The source code of

the platform, the different algorithms, and all the experimental results can be accessed on a public git repository [4].

6.2.3 Workflows

We evaluate the algorithms by running the realistic workflow *Montage* presented in [78] and depicted in Figure 6.1. There are two different versions of the *Montage* workflow used in these evaluations. A version composed of 31 tasks, and a version composed of 619 tasks. Table 6.6 and Table 6.7 lists the different tasks of respectively *Montage* 31 and *Montage* 619, and presents for each task its execution time (on an Ecotype node), its amount of input data and the amount of data it creates. All executions presented in the results have been correctly computed, *i.e.*, producing the correct result of the *Montage* workflow (an example of result is presented in Figure 6.2).

Name	Quantity	μ (s)	σ (s)	size (MB)	input (MB)	output (MB)
mProject	6	23	3	4.5	1.5	8
mDiffFit	3	2	1	18	16	1
mConcatFit	3	2	1	0.5	1	1
mBgModel	3	2	1	0.1	1	1
mBackground	6	2	1	4.4	8	8
mImgtbl	3	2	1	4.5	8	1
mAdd	3	2	1	8.8	16	1
mJPEG	4	2	1	5.1	1	1
Sum	31	37	10	169.5	187	114

Table 6.6 – Description of the 31 tasks composing *Montage* 31 - namely the average and the deviation of the length, the size of the executable and the sizes of the input and output data

6.3 Execution on real infrastructure - OnlyUsedNodes evaluation

This section presents an experimentation that compares the OnlyUsedNodes and the v-HEFT algorithm. In this experimentation, unlike the first simulated experimentation, v-HEFT and OnlyUsedNodes uses the resources selection algorithm presented in Section 5.3 of Chapter 5, and that is used by the NearDeadline algorithm. Therefore, instead of selecting the size of a VM in a set of predefined VM templates (cf. Section 4.3.3),

4. `https://gitlab.inria.fr/ecadorel/workflowplatform`

Name	Quantity	μ (s)	σ (s)	size (MB)	input (MB)	output (MB)
mProject	90	23	3	4.5	1.5	8
mDiffFit	423	2	1	18	16	1
mConcatFit	3	2	1	0.5	1	1
mBgModel	3	8	1	0.1	1	1
mBackground	90	2	1	4.4	8	8
mImgtbl	3	2	1	4.5	8	1
mAdd	3	2	1	8.8	16	1
mJPEG	4	2	1	5.1	1	1
Sum	619	43	10	8,477	7,705	1,879

Table 6.7 – Description of the 619 tasks composing *Montage* 619 - namely the average and the deviation of the length, the size of the executable and the sizes of the input and output data

the algorithm computes the size of the VM based on the amount of resources used by the tasks it will execute (cf. Algorithm 10 and Algorithm 11).

This evaluation is divided in two parts. The first part presents the scenario and the evaluation metrics. The second part presents the results of the experimentation.

6.3.1 Scenario and performance metrics

Parameters of the execution environment

The execution environment used in this evaluation is the environment presented in Section 6.2. The boot time of the ubuntu VMs is assumed to take 40 seconds, this estimation has been acquired by sampling. In this evaluation, the uncertainty is not taken into account, thus the value σ presented in Table 6.6 and Table 6.7 are not used, and the time taken by the tasks is assumed to be the μ value. The execution is performed on 25 nodes of the Ecotype cluster, with one node dedicated to both the MASTER and the SCHEDULER modules, and the others dedicated to instances of the WORKER module.

Scenario composition

The workload of this evaluation is composed of 10 *Montage* 619 workflows submitted at time $t = 0$. Each workflow belongs to one user, meaning that the VMs used to execute the tasks of one workflow cannot be used to execute the tasks of another one. The deadline of the workflows is made variable and vary from 100 to 600 seconds, by increment of 100 seconds. A deadline of 200 seconds is a tight deadline, considering that by using almost all the nodes (22 of the 24), one *Montage* 619 workflow needs around 150 seconds to be

executed, and takes approximately 300 seconds using only one node (when scheduled by the v-HEFT algorithm).

Performance metrics

There are two performance metrics in this evaluation, the number of nodes being used at some point in the experimentation, named *nb-used* and the energy consumption *power-usage*. The last metric, named *power-usage*, is the power consumption of the physical machines during a period of T seconds, divided by the maximal possible power consumption during the same period of time. The period T seconds is the maximal duration of execution of a scenario in these evaluations. The real power consumption of the nodes is retrieved thanks to the SeDuCe platform [103], which monitors the electrical consumption of the nodes using Power Distribution Units (PDUs).

6.3.2 Evaluation results

Table 6.8 presents the *power-usage* of the different execution, as well as the number of used nodes, and the makespan.

	v-HEFT	OnlyUsedNodes					
Deadline	-	100	200	300	400	500	600
power-usage (with $T = 539$ (s))	70.1%	70.2%	62.5%	60.4%	60.4%	59.9%	59.5%
Energy (Joules)	1,267,849	1,270,559	1,131,243	1,093,689	1,092,629	1,084,747	1,076,537
Gain	0%	+0.1%	-10.8%	-13.8%	-13.8%	-14.6%	-15.1%
nb-used	24	24	20	10	9	7	5
Makespan	427	473	244	315	416	457	539

Table 6.8 – *Power-usage* and *nb-used* (cf. 6.3.1), with the gain computed in comparison to v-HEFT results

The v-HEFT algorithm does not take into consideration that the workflows belongs to multiple users (except for VM privacy, where a task can be executed only by a VM of its user), and merge all of them into one big workflow. In addition, it spreads the tasks on a maximum of nodes, as the makespan minimization is the objective of the algorithm. For this reason, when the algorithm schedules the *mDiffFit* tasks, almost all the infrastructure is reserved for one user, leading to the need to start new VMs for the *mDiffFit* tasks of the second user, and so on. In fact, the v-HEFT algorithm almost instantiates new VMs for each level of tasks of the *Montage* workflow, creating big time losses due to VM booting. An example of this phenomenon is illustrated in Figure 6.7. For

this reason the makespan of the v-HEFT execution is not the best possible makespan, and OnlyUsedNodes is able to have a better makespan than v-HEFT when the deadline is tight but still feasible (*e.g.*, the makespan of v-HEFT is 423 seconds, when the makespan of OnlyUsedNodes with a deadline of 300 seconds is 315). However, when the deadline is too tight, OnlyUsedNodes has to use all the nodes, and results the same behavior as v-HEFT, with the exception that the synchronization instead of being at the granularity of the tasks (*e.g.*, *mConcatFit* waiting for all *mDiffFit* tasks of all users), is at the granularity of the workflows (*e.g.*, the workflow of second user waits the end of the workflow of the first user), thus resulting with a makespan far from the deadline (*i.e.*, 473 instead of 100 seconds, even if a deadline of 100 seconds is in principle impossible to guarantee), and a high energy consumption, as it can be seen in Table 6.8.

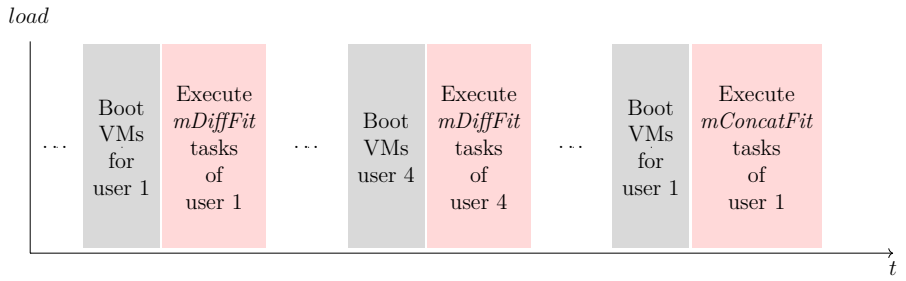


Figure 6.7 – Representation of delay introduced by synchronization with HEFT algorithm

One can note that the makespan is sometimes over the deadline, when using the OnlyUsedNodes algorithm, this is due to the fact that no uncertainty is taken into account, and thus sometimes tasks takes more time to be executed than the time estimated by the algorithm. To validate this hypothesis, we have executed the OnlyUsedNodes algorithm with the uncertainty system presented in Equation 5.2 of Section 5.2, using the x parameter with a value of 0.7, and a deadline of 300 seconds. This execution, instead of using 10 nodes, used 15 nodes, and had a makespan of 297 seconds, instead of 315 seconds. This execution by using 5 more nodes, consumed more energy as its *power-usage* was 61.7% instead of 60.4%.

From Table 6.8, we can claim that the gain of energy consumption observed in the simulation results of Section 6.1, is also observed on a real infrastructure execution. One can note a correlation between the number of used nodes, and the energy consumption, and that the OnlyUsedNodes algorithm has the expected behavior even when scheduling big workflows composed of hundreds of tasks.

6.4 Execution on real infrastructure - NearDeadline evaluation

This evaluation section shows comparisons between NearDeadlineTask, NearDeadlineVCpu, v-HEFT, and OnlyUsedNodes. Unfortunately, as the algorithms proposing online workflow scheduling with uncertainty in the state of the art consider a different problem (*e.g.*, different infrastructure) they cannot be adapted in our context without strong modifications. Consequently our evaluation does not present experimental comparison with them.

This evaluation is divided in four parts. The first part presents the scenario composition and the metrics of evaluation. A first evaluation is performed on 10 nodes, with small workflows composed of 31 tasks, this evaluation aims at evaluating the quality of the algorithms with an easily controllable workload. The second evaluation is performed on 25 nodes with bigger workflows composed of 619 tasks, and is intended to evaluate the scalability of the contribution. A third evaluation is meant to evaluate the impact of the parameter α - of the fitness function isBestBE presented in Section 5.3.4 - on the quality of the execution, and the different metrics.

6.4.1 Scenarios and performance metrics

Parameters of the execution environment

The execution environment used in the three following evaluations is the environment presented in Section 6.2. The properties $\mu_{v,n}$ and $\sigma_{v,n}$ used for the estimation of the booting time of the ubuntu VMs, acquired by sampling are set to respectively 31 seconds, and 20 seconds. The certainty parameter x presented in Equation 5.2 of Section 5.2, has been set to 0.7 for both VM and task execution times. This is the value that gives us the best results, being a good trade-off between a too optimistic and a too pessimistic accounting of the uncertainty. In the following evaluations, the uncertainty parameters are used for every algorithm (NearDeadline, v-HEFT, and OnlyUsedNodes), meaning that every algorithm will have the same estimation of the required time for each task.

Scenarios composition

Our evaluations are conducted on a multi-user scenarios where each user submits one workflow at a given instant, with a given deadline. We have divided the workload into two

subsets, the workload that arrives at time 0, named *initial workload*, and the workload composed of workflows with unknown arrivals (for the algorithm), named the *interfering workload*. A scenario consists in five different variables, that are defined as follows:

— *nb_init*: the number of users submitting a workflow at time 0;

— *nb_inter*: the number of users submitting a workflow at a different arrival;

— *dead_init*: the deadline of the workflows in the initial workload;

— *dead_inter*: the deadline of the workflows in the interfering workload;

— *arrival*: the arrival time of the workflows in the interfering workload.

Performance metrics

The performance metrics are based on the objectives presented in the Section 5.2. The first metric, named *time-violation*, is the sum of $violation_w$ of Equation 5.10 for each $w \in \mathcal{W}$. The second metric, named *nb-succeed*, is the number of workflows that have successfully been executed before their deadline. The last metric, named *power-usage*, is the power consumption of the physical machines during a period of T seconds, divided by the maximal possible power consumption during the same period of time. The period T seconds is the maximal duration of execution of a scenario in these evaluations. The real power consumption of the nodes is retrieved thanks to the SeDuCe platform, which monitors the electrical consumption of the nodes using Power Distribution Units (PDUs).

6.4.2 Small workflows scheduling

This section aims at evaluating the performance of the NearDeadline algorithm with a small infrastructure and small workflows, in order to have an easily controllable workload. We have used 10 nodes of the Ecotype cluster, with one node dedicated to both the master and the scheduler modules, and the others dedicated to instances of the worker module. The parameters used in this evaluation are those presented in Section 6.4.1. The α parameter of the isBestND fitness function of the NearDeadline algorithm, is set to $\frac{1}{250}$. This choice will be explained in the third evaluation in Section 6.4.4.

Scenarios

The scenarios of this evaluation are presented in Table 6.9, that lists the values of the five parameters presented in Section 6.4.1: *nb_init*, *nb_inter*, *dead_init*, *dead_inter* and

scenario	*nb_init*	*nb_inter*	*dead_init* (s)	*dead_inter* (s)	*arrival* (s)
A	50	15	300	{250, 200, 150, 100}	each 10
B	50	15	300	200	{at 0, at 10, at 50, at 150}
C	50	{5, 10, 15, 20}	300	200	each 10
D	50	15	{300, 250, 200, 150}	200	each 10

Table 6.9 – Description of the different scenarios of the evaluation of NearDeadline with small workflows

arrival. The scenario (A), where each interfering workflow arrives each 10 seconds from the instant 0, aims at showing the impact of the variation of the deadline of the workflows in the interfering workload. In the second scenario (B) all the workflows of the interfering workload arrive at the same time, but this arrival time is made varying. The scenario (C) makes the number of interfering workflows vary when they arrive each 10 seconds. The last scenario (D) aimed at showing the impact when the deadline of the initial workload varies.

Evaluation results

Figure 6.8 represents the *power-usage* metric for the execution of the different scenarios (cf. Table 6.9) with the different algorithms. The first observation that can be made, is that our algorithm NearDeadline is always able to minimize the energy consumption compared to v-HEFT. This is due to the dispersion, *i.e.*, distribution, of the tasks among all the nodes induced by the makespan minimization. To prevent task dispersion, OnlyUsedNodes, places the tasks on already used nodes and therefore tries to consolidate before using a new node. One can note that OnlyUsedNodes is often better than NearDeadlineTask and NearDeadlineVCpu for minimizing energy consumption, as for the scenario (A) for instance. In this scenario, both NearDeadlineTask and NearDeadlineVCpu reconsider the *expected planning*, kill VMs and launch new ones, and therefore consume more energy, when OnlyUsedNodes will not change anything. This is illustrated by the number of VMs required by NearDeadlineTask and NearDeadlineVCpu, as it can be seen in Figure 6.11 that shows the number of VMs instantiated during the execution. However, NearDeadlineTask and NearDeadlineVCpu are far better than OnlyUsedNodes in optimizing the user fairness, as it can be seen in the Figure 6.9 and 6.10, that respectively shows the *time violation* and *nb succeed* metrics. Figure 6.9 illustrates that NearDeadlineTask and NearDeadlineVCpu are able to adapt their *expected planning* to the submission of new

workflows at different arrivals.

In the scenario (A), both v-HEFT and OnlyUsedNodes place the new workflows at the end of their planning, and therefore, the more the deadlines of the interfering workflows are tight, the more the *time violation* will be high. For NearDeadlineTask and NearDeadlineVCpu, however, this does not have such a significant impact. The time violation for v-HEFT and OnlyUsedNodes are always correlated to the fact that they do not reconsider the previous planning in every scenarios.

The scenario (D) aimed at showing the impact of the variation of *dead_init*. In the fourth variation, for the value 150, no algorithm is able to guarantee the deadline for the initial workload, but there is probably no solution to successfully perform this execution in time. In addition, NearDeadlineTask and NearDeadlineVCpu are still able to overcome v-HEFT and OnlyUsedNodes in both energy and user fairness objectives. In this (D) scenario, as the deadline gets shorter for the *initial workload*, the *interfering workload* has a lower priority, to the point that in the two last executions (with a deadline of 200 and 150 for the *initial workload*), the NearDeadlineTask and NearDeadlineVCpu algorithm does not make any reconsideration of the initial *expected planning*, and thus the energy consumption succeed to be even lower than the one of OnlyUsedNodes. One can observe some correlation between the number of instantiated VMs and the energy consumption for both NearDeadlineTask and NearDeadlineVCpu algorithms.

The scheduling algorithm is executed simultaneously to the workflow execution. Hence, knowing the duration for each decision taken is difficult to evaluate. But one can note that the solving time of NearDeadlineTask and NearDeadlineVCpu is necessarily small enough for the algorithm to give an *expected planning* that maximizes the user fairness in most case.

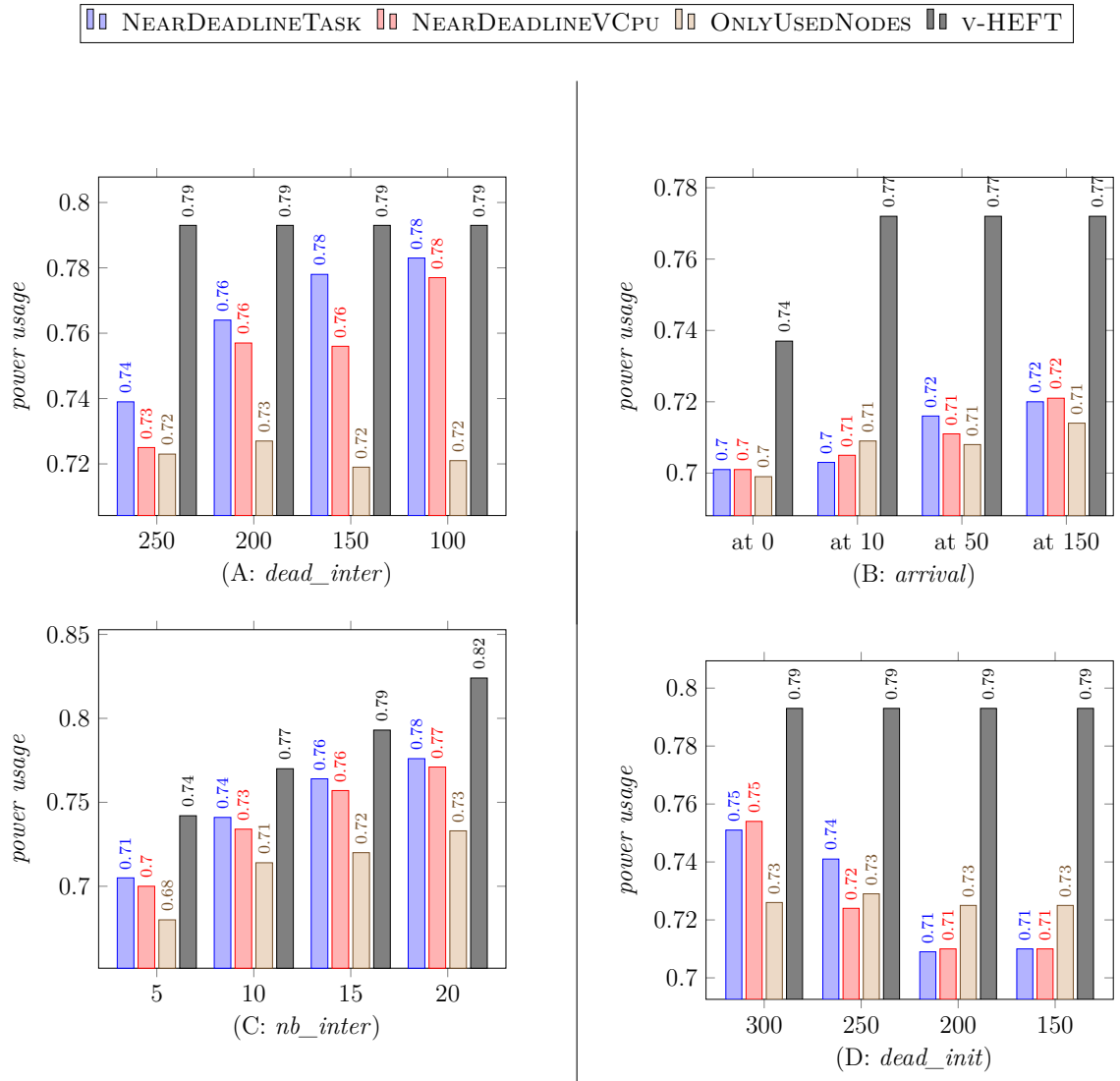


Figure 6.8 – Power usage of the different scenario executions

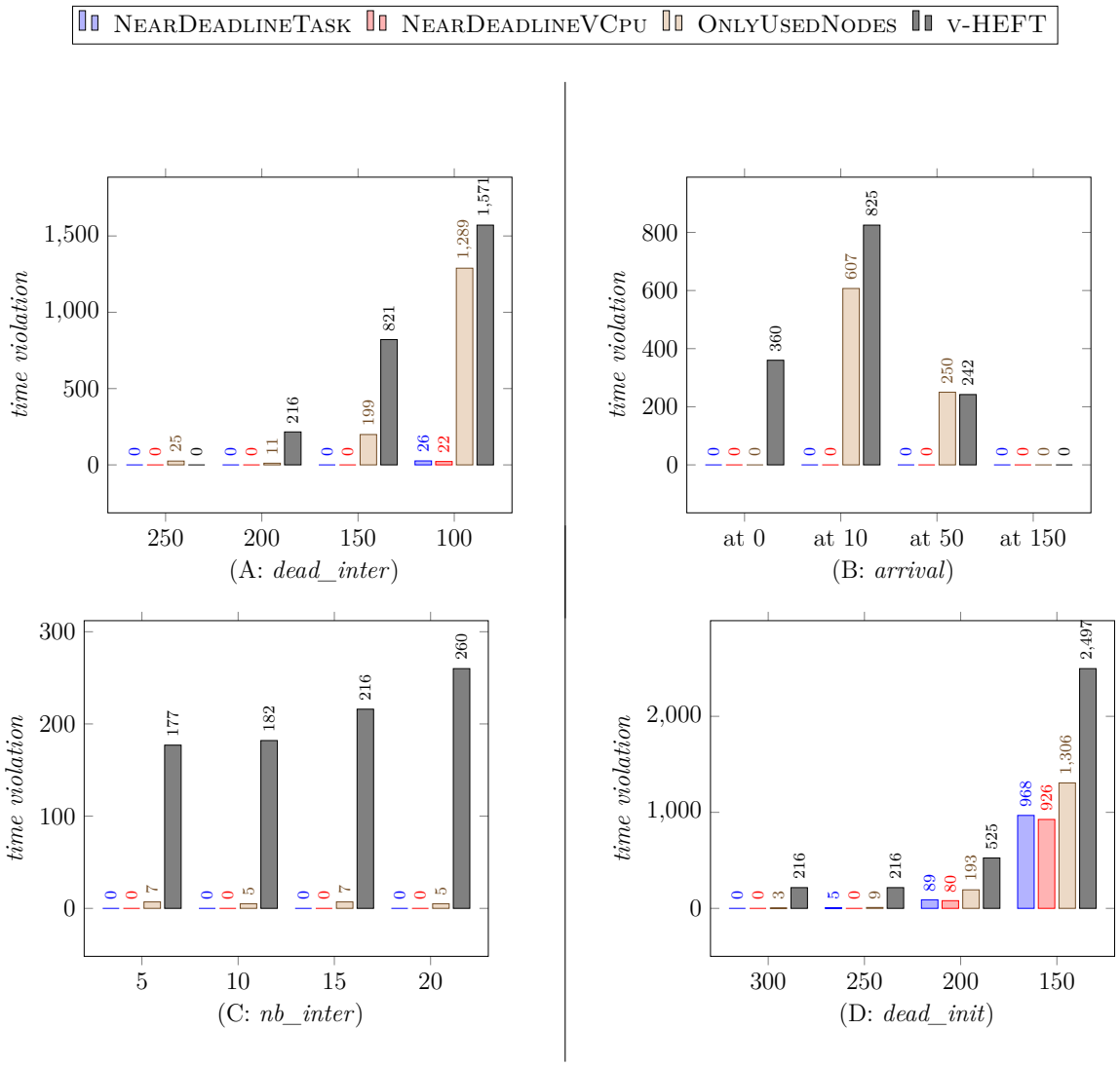


Figure 6.9 – Time violation of the different scenario executions

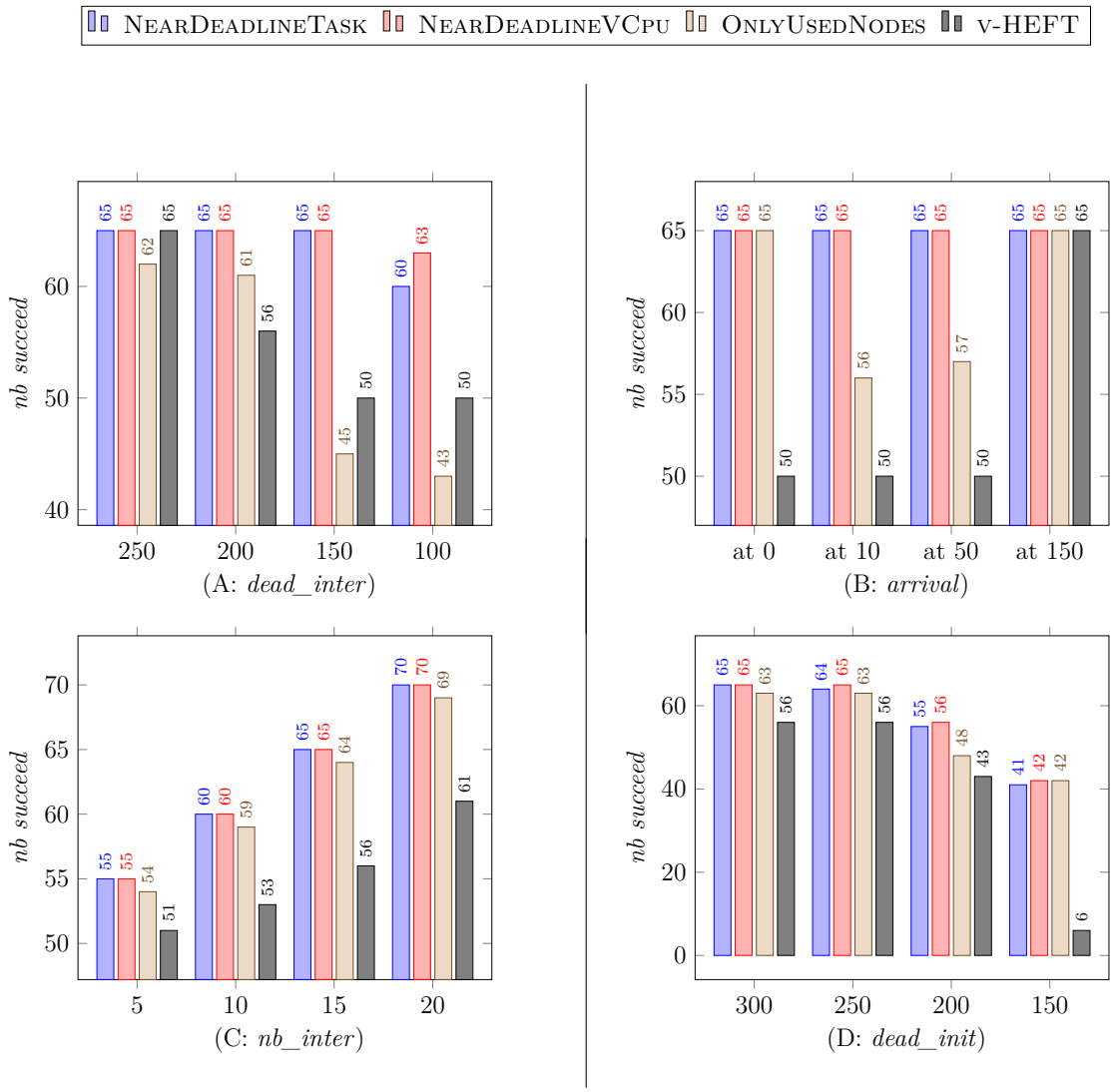


Figure 6.10 – The number of successful execution under deadline

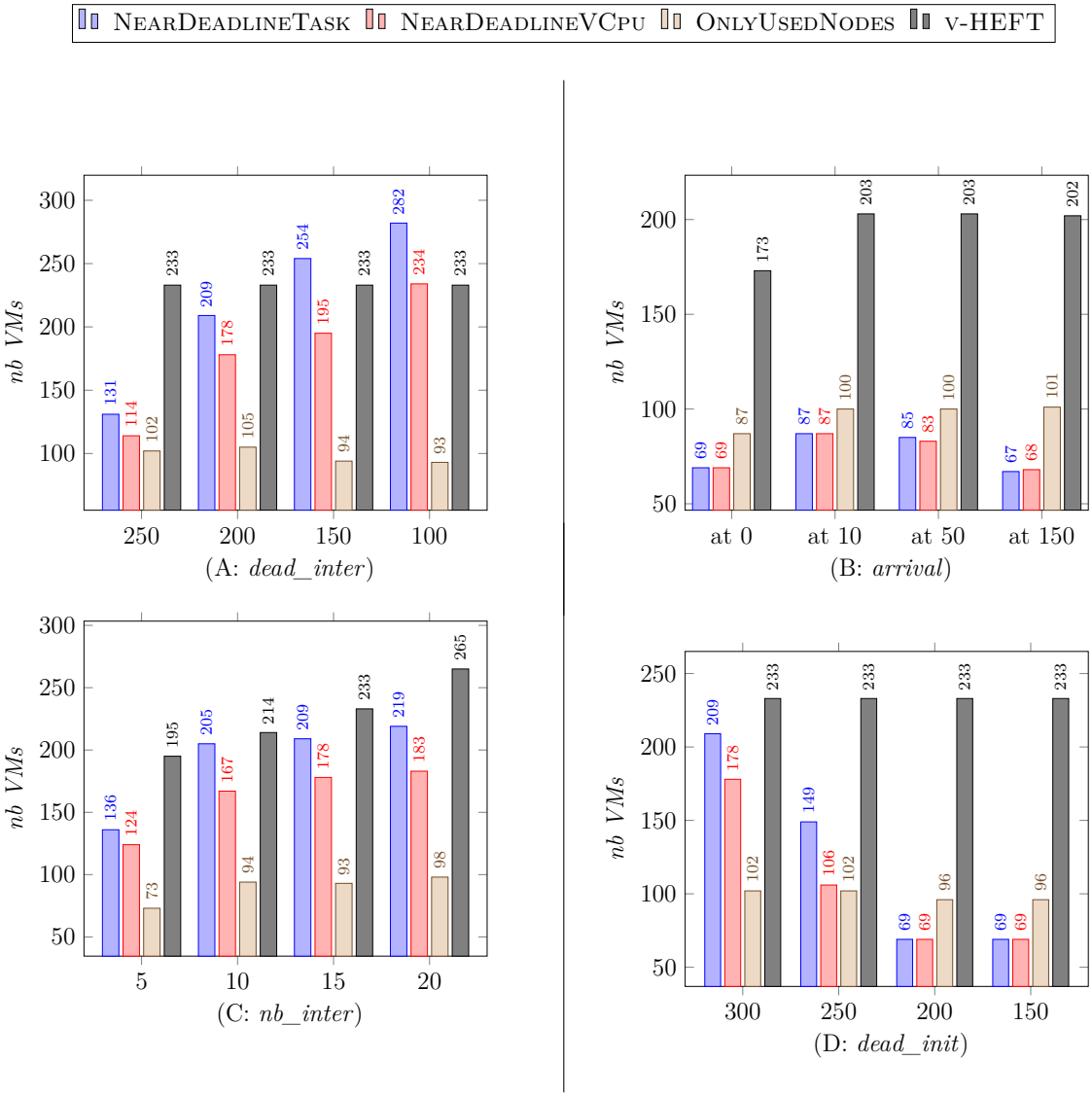


Figure 6.11 – Number of instanciated VMs during the different scenario executions

6.4.3 Scalability evaluation

This second evaluation aims at evaluating the performance of the NearDeadline algorithm in a bigger infrastructure with bigger workflows, in order to evaluate the scalability of the solution. In this evaluation, the workflow *Montage* composed of 619 tasks is used. We have used 25 nodes of the Ecotype cluster, presented in Table 6.5, with one node dedicated to both the master and the scheduler modules, and the others dedicated to instances of the worker module. The parameters used in this second evaluation are the same as those presented in Section 6.4.1. As for the first evaluation of NearDeadline algorithm, the parameter α has been set to $\frac{1}{250}$. This chosen value is argumented in the third evaluation of Section 6.4.4.

Scenario

In this evaluation, the workflow *Montage* 619 is used, and the value *nb_init* is set to 20 workflows, and *nb_inter* to 10. All the workflows of the *interfering workload* arrives at the same time $t = 50$. In this evaluation, the number of tasks is 9.2 times higher than the number of tasks in the second evaluation, for a number of compute nodes around 2.6 times higher. The value of *dead_init* is set to 1000, and the value of *dead_inter* is set to 300. As already mentioned, the *Montage* workflow, composed of 619 tasks, takes around 150 seconds to be executed when scheduled by the v-HEFT algorithm, and this execution uses 22 of the 24 compute nodes of the infrastructure. Thus this scenario represents a workload that highly loads the infrastructure.

Evaluation results

Figure 6.12 presents the distance between the makespan and the deadline (see Equation 5.9). The x-axis represents the deviation of the workflow executions to the deadline in seconds, with the represented values being the minimum value, the lower quartile, the average, the upper quartile and the maximum value.

	v-HEFT	OnlyUsedNodes	NearDeadlineTask	NearDeadlineVCpu
power-usage (with $T = 1312$ (s))	74.2%	64.3%	62.8%	62.7%
Energy (Joules)	4,030,664	3,496,220	3,411,950	3,408,759
Gain	0%	-13.3%	-15.4%	-15.5%

Table 6.10 – *Power-usage* (cf. 6.4.1) during the second evaluation, with gain computed in comparison with v-HEFT results

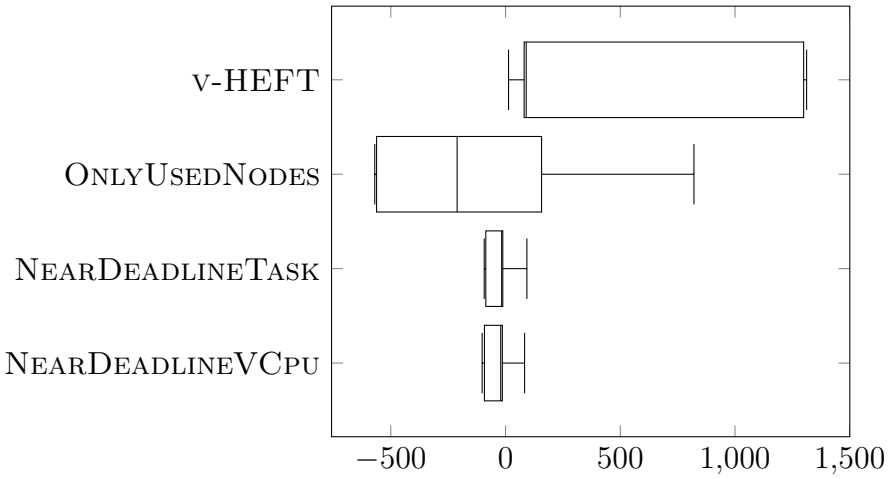


Figure 6.12 – Distance to deadline of the workflow executions

As already discussed in Section 6.3.2, The v-HEFT algorithms does not take into consideration that the workflows belong to multiple users, and instantiates new VMs for each level of tasks of the *Montage* workflow, creating big time loss due to VM booting. When the *interfering* workload is submitted at time $t = 50$, all the resources are yet reserved for the *initial* workload, and as no reconsideration is made by v-HEFT algorithm, the workflows are scheduled after the end of the *initial* workload. For all this reasons, not a single deadline is respected.

The algorithm OnlyUsedNodes, succeeds in creating a small number of VMs when the deadlines can be met with a reduced number of nodes (*initial* workload). But when deadlines are tights, due to the arrival of the *interfering* workload, all the nodes in the infrastructure are needed, and the algorithm start to take the same decisions as the v-HEFT algorithm, and schedule the new submitted workflows after the already scheduled ones. However, the problem raised by the v-HEFT algorithm (global synchronization) is not present in the algorithm OnlyUsedNodes because the workflows are scheduled one by one instead of being merged together. Nevertheless, in term of fairness, the first submitted workflow will have a much better makespan than the last one submitted, that is why a significant variance can be observed in Figure 6.12.

One can note that both NearDeadlineTask and NearDeadlineVCpu algorithms are far better than v-HEFT and OnlyUsedNodes in user fairness, as the deadlines are more respected. To this observation, we can also add that the workflow executions never finish very far from their deadlines, unlike the OnlyUsedNodes algorithm, which sometimes schedules workflow to be finished 500 seconds before their deadline. Consequently,

when using NearDeadlineTask and NearDeadlineVCpu, free resources are available for the execution of the workflows with tight deadline, leading to an enhancement of user fairness.

Table 6.10 shows the power usage of each execution. NearDeadlineTask and NearDeadlineVCpu have approximately similar results, and shown to be more effective than v-HEFT and OnlyUsedNodes in energy consumption minimization when the workload is high. Indeed, because OnlyUsedNodes uses all the available nodes, once the *interfering* workload is submitted, it then has the same behavior as v-HEFT, and spread the tasks across all the nodes, and consume much energy. On the other hand, NearDeadlineTask and NearDeadlineVCpu algorithms, by giving priority to VMs that are already containing many tasks or with many VCPUs, are able to minimize the energy consumption.

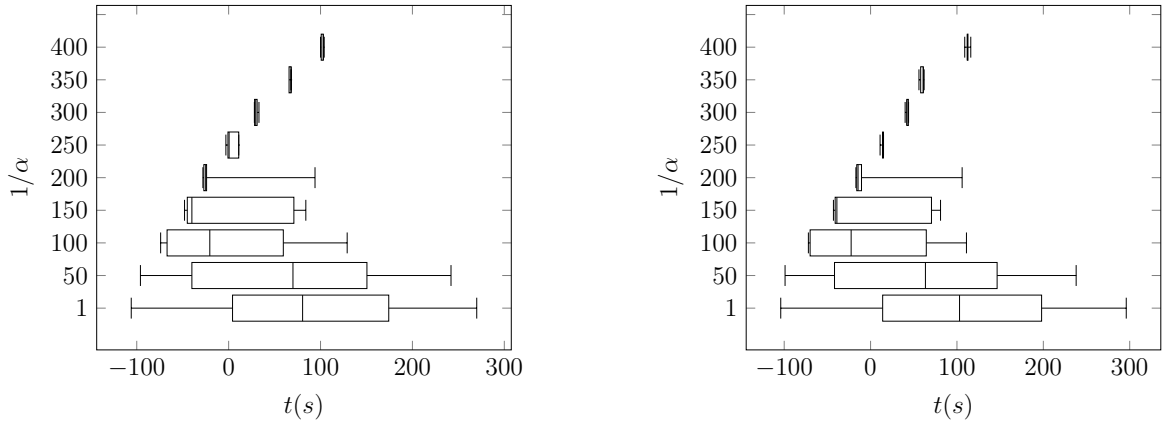
6.4.4 Analysis of the α parameter

This third evaluation aims at showing the impact of the α parameter used in the objective function isBestBE, during the panic mode. The α parameter is used in the fitness function in order to make vary the priority of the two following objectives: the minimization of the makespan of a task, and the maximization of the *quality* of the VM that will execute the task. Recall that the *quality* of a VM presented in Section 5.3.4 is greater for a VM that have more VCPU when using NearDeadlineVCpu or more tasks when using NearDeadlineTask. The greater the value of the α parameter, the greater the priority of the makespan over the priority of the *quality* and by extension of the consolidation.

In this evaluation, the workload is composed only of *initial workload* with a tight deadline that is almost impossible to meet, such that the panic modes of both the NearDeadlineTask and NearDeadlineVCpu algorithms are evaluated. We have used 25 nodes of the Ecotype cluster, with one node dedicated to both the master and the scheduler modules, and the others dedicated to instances of the worker module. The initial workload is composed of 10 *Montage* workflows composed of 619 tasks. The *init deadline* is set to 250 seconds, which is a tight deadline, considering that by using almost all the nodes (22 of the 24), one *Montage* 619 workflow needs around 150 seconds to be executed, and takes approximately 300 seconds using only one node (when scheduled by the v-HEFT algorithm). The parameter α is made varying between $\frac{1}{1}$ and $\frac{1}{400}$. For readability reasons, in the following figures the inverse of the parameter α, namely $\frac{1}{\alpha}$ is used between 1 to 400.

Evaluation results

The x-axis of Figure 6.13 represents the deviation of the workflow executions to the deadline in seconds (cf. Equation 5.9). The represented values being the minimum value, the lower quartile, the average, the upper quartile and the maximum value. The y-axis of Figure 6.13 is the variation of the α parameter. The parameter α has an impact on the distribution of the execution across the nodes. The smaller the α parameter, the lower the distribution of the tasks across the nodes, and as a result the makespan of a workflow becomes also higher. Indeed, as α decreases, the priority of the makespan decreases relative to the consolidation. However, if the distribution of the tasks is low, the number of required nodes for one workflow is also low, and thus the fairness between the users is enhanced.

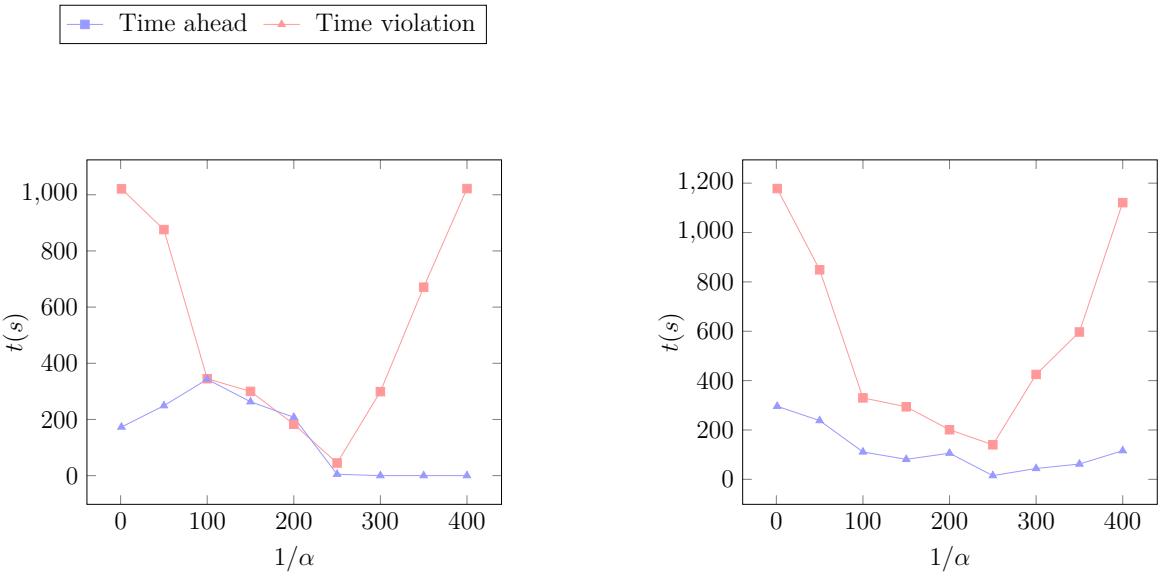


(a) NearDeadlineTask (b) NearDeadlineVCpu

Figure 6.13 – Distance to deadline of each workflow with variable α parameter

From Figure 6.13, one can note that with $\alpha = \frac{1}{1}$, the best makespan (minimal value of the distance to the deadline) is 100 seconds before its deadline, or about 150 seconds, as it would be with the v-HEFT algorithm, due to a high distribution of the tasks across the nodes. It can be observed that with $\alpha \leq \frac{1}{250}$, the fairness between the users is almost optimal, the variance of the distance to deadline being very low, meaning that all the users have almost the same deadline violation. With $\alpha = \frac{1}{250}$, not only the variance is low, but the deadline violations are the lowest as it can be observed in Figure 6.14. In this figure are represented two values: the sum of violation of the deadlines of all the executed workflows (cf. Equation 5.10), denoted Time violation; and the sum of the ahead time

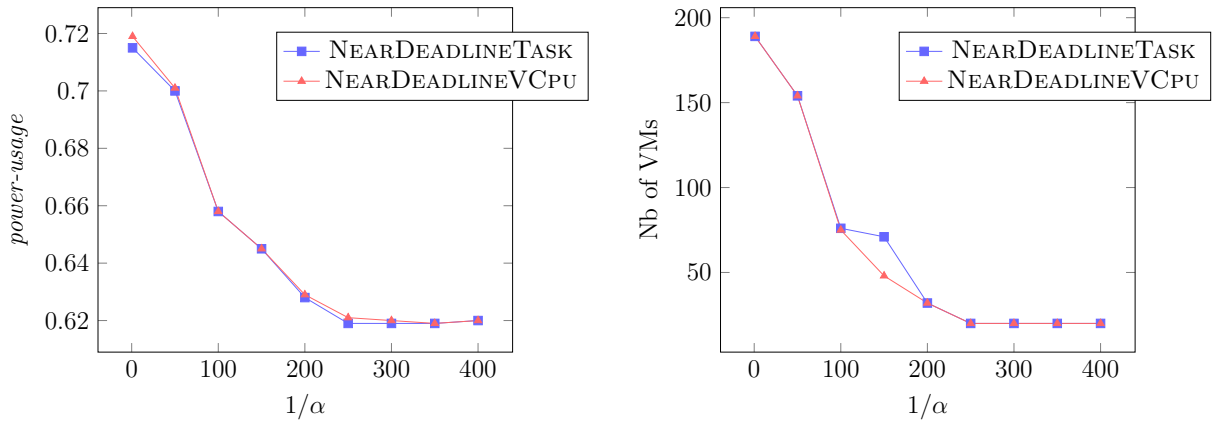
(cf. Equation 5.12), denoted Time ahead. The maximization of the time ahead is not an objective of the NearDeadline algorithm, but is a great indicator of the fairness between user. Indeed, if the time ahead is high, the resources are used by users who could have used less resources and still guarantee the deadline of their workflow, and thus deprive the other users from resources, leading them to have high deadline violation. In Figure 6.14, as the time ahead decreases the time violation also decreases, until a peak, where the makespan objective has too low priority, and thus faster execution could have been performed, without deteriorating the fairness.



(a) NearDeadlineTask (b) NearDeadlineVCpu

Figure 6.14 – Sum of the time violation (Eq. 5.10) and sum of the time ahead (Eq. 5.12) with variable α parameter

Figure 6.15a presents the energy consumption recorded during the executions of the workflows with a varying α parameter. Figure 6.15b presents the number of instantiated VMs during the same executions. One can observe a correlation between the α parameter and the number of instantiated VMs, which is expected (less distribution meaning fewer VMs). This correlation can also be observed between the α parameter and the energy consumption, the lower the α parameter the lower the energy consumption. This correlation can be explained by the energy lost during the VM booting processes, which consume energy but produce no results, thus if there are fewer VM, the sum of consumption of the boot processes is also lower.



(a) Percentage of energy consumption with variable α parameter

(b) Number of instanciated VMs with variable α parameter

Figure 6.15 – *Power-usage* and number of VMs for NearDeadlineTask and NearDeadlineVCpu execution with a variable α parameter

6.5 Conclusion

In this chapter, multiple experimentations have been presented. Simulations on realistic workloads have shown a reduction in the number of nodes by up to 28% for an estimated 10% energy consumption reduction for the Cloud provider, when using the OnlyUsedNodes algorithm in comparison to the v-HEFT algorithm. Further experimentation on a real infrastructure, have shown that this reduction is still observed, when executing real workflows. The evaluation on real infrastructure have shown a reduction in the energy consumption up to 15% for a reduction of 79% in term of number of used nodes.

In a second part, the NearDeadline algorithm has been compared to v-HEFT and the OnlyUsedNodes algorithms, on real experiments on a real infrastructure. Experiments have shown real benefits in the reduction of both deadline violation, and thus in user fairness as well as in energy optimization. Indeed, as v-HEFT and OnlyUsedNodes does not reconsider the planning they have computed, when new submissions arrive with higher priorities, they are placed at the end of the planning, resulting in a poor fairness optimization. Conversely, by reconsidering the *expected planning* NearDeadline is able to enhance the fairness. NearDeadline as the OnlyUsedNodes algorithm is able of

minimizing the energy consumption, and experimentation have shown a reduction of the energy consumption up to 15% in comparison with v-HEFT.

In this evaluation, the parameter α of the NearDeadline algorithm has been investigated. Experimentations that aim at finding the best possible values for the α parameter, in different infrastructures and for different workflows, could be the object of interesting future works.

Part II

Automatic execution of scientific workflows

Chapter 7

WaaS : Workflow as a Service A Cloud service for scientific workflow execution

This chapter presents the definition of a new Cloud service designed specifically for the execution of scientific workflows, with a *specific-service-vision*. We show that this conceptual choice enhances separation of concerns between end-users and Cloud providers, and enhances resource management from the Cloud provider viewpoint. Furthermore, to facilitate the integration of the new WaaS service by Cloud providers, it is designed as a turnkey solution that handles modularity of virtualization mechanisms and scheduling policies, as well as scalability issues.

Contents

7.1 Introduction

Scientific workflows often become very complex and difficult to manage. Thus, in order to execute a workflow, an end-user (*i.e.*, a scientist) will generally use a workflow engine or a workflow management system, such as Pegasus [56], DEWE [86], or Hyperflow [27] for instance, that will automatically orchestrate the different execution steps. These systems are designed in a way to be used and deployed by the end-users, and not by a Cloud provider. Indeed, for instance, Pegasus and Hyperflow uses existing resource provisioning mechanisms like those of AWS, and DEWE uses existing Function-as-a-Service (FaaS).

This conceptual vision will be called *existing-service-vision* in the rest of this chapter. As existing services of the Cloud are not specific to scientific workflows, this conceptual choice requires additional work for the scientist: resource provisioning, configuration etc. This can be a challenging task, especially when considering that end-users are generally not experts in resource management and system configuration.

The remainder of this chapter is organized as follows. Section 7.2 details the contribution, *i.e.*, the submission language, the architecture and the modular aspect of the new WaaS service, and Section 7.3 evaluates this solution. Finally, Section 7.4 concludes this work.

7.2 WaaS : Workflow as a Service

In this section the WaaS Cloud service is presented. This new Cloud service aims to be specific to scientific workflows and aims to be offered by Cloud providers on their distributed infrastructures. The WaaS addresses the limitations of the related work in terms of virtualization, modularity and separation of concerns. Indeed, the WaaS service has been designed with the separation of the concerns between two actors in mind, namely the *end-user*, who is the scientist writing tasks and composing them, and the *Cloud provider*, responsible for virtual and physical resource management.

This section is divided into two parts. The first part presents the operations performed by an end-user wishing to run a scientific workflow, while the second part presents how the service can be deployed and customized by a Cloud provider. This second part also presents a detailed view of the service architecture and its different modules. In this section, the presentation focuses on the objectives presented in Table 3.5 of Section 3.2, on the related work of the automatic execution of scientific workflows.

7.2.1 End-user concerns

As presented in the Chapter 2, a scientific workflow is a succession of tasks with file dependencies, which can be described by a DAG. In our context, the end-user's job is to develop the different tasks of the workflow and to describe the workflow topology. As explained in a Section 2.1, the tasks that make up the workflows can be very heterogeneous. In the WaaS, the workflow topology must be given by the end-user in a Yaml [124] description file. Figure 7.1 presents the meta-grammar of the workflow description language,

and Figure 7.2 presents an example of a workflow. In the WaaS the meta-grammar of the workflow description has to be implemented and customized by the Cloud provider to get its specific grammar. This will be detailed later.

```
files: # list of files
    - id: value # file ID
      name: value # file name
      type: output # Optional
      *additional optional attributes
tasks: # list of tasks
    - app: value # Name of executable
      hardware: # list of Hardware requirements
        key1: value1
        key2: value2
        ...
      os:
        name : value # Name of operating system
        software: # list of software dependencies
          - value1
          - ...
      output: # list of file references
        - value1
        - ...
      input: # list of file references
        - value1
        - ...
      params : value # execution parameters
      *additional optional attributes
```

Figure 7.1 – Meta-grammar of the workflow description file

In the YAML description file, two main elements are required, the specifications of *files* and *tasks*. The *files* section is a list of all the files that are created during the workflow execution and transmitted from tasks to tasks. Each file is specified by an ID and a name. A file whose type is *output* (optional), is a file that the end-user wants to retrieve at the end of the execution. Similarly, The *tasks* section is a list of all the tasks of a workflow. For each task is associated the path and name of the executable file to run, as well as the associated execution parameters, the hardware requirements of the task, the needed operating system with attached software dependencies, its input files as well as its output files (as references to the *files* list). The end-user has the ability to specify a list of

software dependencies which must be installed before launching the task. All provisioning and installation operations are automated by the WaaS, and the user does not have to manage any of them (close to *Docker File* mechanism).

One can note that for each file, or task, optional attributes can be added. Those optional attributes depend on the type of scheduler that the Cloud provider chooses to deploy, and on the type of hardware and operating systems available in their specific infrastructure. Thus, it is up to the Cloud provider to define the specific grammar of the description file accepted in its service, by defining the list of attributes required in the Yaml description file. Figure 7.2 presents an example of a workflow. The example file corresponds to the workflow depicted on the left-side of the figure. In the example, a few optional parameters have been added by the Cloud provider: the size of the files to transfer between tasks; the duration of each task as a number of instructions; the demand in terms of CPUs and memory.

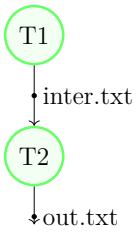
Once the end user has described her/his workflow, developed the tasks and created the input files, he/she submits the workflow to the service. This submission can be made on a simple web interface, or via a single command line. Once the workflow has been correctly executed by the Cloud provider the result files (output files) of the workflow will be available to the end-user via a web link. The end user does not have any other work to do, so it can be said that WaaS **objective 8** - *Hidden management for the end-user* - of Table 3.5 regarding separation of concerns is achieved.

7.2.2 Cloud provider concerns and Waas architecture

The entire execution of the workflow, *i.e.*, the scheduling, the resource management, the execution of the tasks as well as the file management, is made by the WaaS service on the Cloud provider side. The service is composed of two main modules, the Master and the Worker modules. The Master module is responsible of a cluster of Workers, where each worker is attached to one physical machine (*i.e.*, node). A cluster is composed of multiple nodes. The bandwidth within a cluster is supposed homogeneous.

Master module

The Master module is hosted by one of the nodes of the infrastructure, and is accessible from the outside of the cluster in order to receive submission requests from end-users. It contains a scheduler submodule. This scheduler is responsible of the following decisions:




```
files:
    - id: File1
      name: inter.txt
      size: 1000 # Kbytes
    - id: File2
      name: out.txt
      size: 2096 # Kbytes
      type: output
tasks:
    - app: T1
      len: 22000
      needs:
          cpu : 1
          memory : 1024
      os: ubuntu18.04
      output :
          - File1
    - app: T2
      len: 34000
      needs:
          cpu: 2
          memory: 2048
      os: ubuntu18.04
      input:
          - File1
      output:
          - File2
      params: -i inter.txt -o out.txt
```

Figure 7.2 – Example of a workflow with two tasks by using a cloud-specific-grammar built from the meta-grammar.

where (on which resource), and when to execute a task. In the WaaS the role of the scheduler is abstracted as a set of interfaces as follows. The scheduler provides for each task a slot $s \in \mathcal{S}$ such as $s =< t, b, e, r >$, where $t \in V$ is a task of a workflow $G = (V, A)$, $b, e \in \mathbb{N}$ are the starting and ending times of the task, and $r \in \mathcal{R}$ is the virtual resource that will execute the task. A virtual resource is defined as follows $r =< b, a, e, k, n >$, where $b, a, e \in \mathbb{N}$ are respectively the starting instant, the availability instant of the resource (after the boot process), and the ending instant (killing of the resource) of the virtual resource. The attribute $k \in \mathcal{K}$ represents the capacities of the VM, such as vCPU, or memory. The attribute n is the node that will host the virtual resource. Depending on available information, the scheduler can determine the ending time of a task, or may set an infinite lifetime for the task slot, and its associated virtual resource. As long as the scheduler is designed with the correct communication interface, any kind of scheduler

can be used in this submodule, thus achieving the **objective 7** - *Modular scheduler* - of Table 3.5.

The set of virtual resources and the tasks affectations to resources is called a planning. The Master module is triggered when new events occur (*e.g.*, new submission or new monitoring information from workers). In case of a new submission, the scheduler submodule is called, otherwise the planning continues to be applied by the Master module.

```
1   name: value
2   hardware:
3     # hardware capacities
4     key1: value1
5     ...
6   os:
7     # OS and boot times
8     key1: value1
9     ...
10  *optional attributes :
```

(a) Meta-grammar of a worker description

```
1   name: node_1
2   speed: 2200
3   hardware:
4       cpus: 16
5       memory: 16384
6   os:
7       ubuntu: 30
```

(b) Example of description of a worker by using a cloud-specific-grammar built from the meta-grammar of Figure 7.3a.

Figure 7.3 – Worker description file

Worker module

One Worker module is deployed on each node of the Cloud infrastructure dedicated to scientific workflows. When deployed a Worker module registers itself to the Master module responsible of the cluster. To the registration is attached information, that can be customized by the Cloud provider, and that are described by a YAML file. The meta-grammar of this description file is presented in Figure 7.3a, and an example is given in Figure 7.3b by using one possible specification of this meta-grammar for a given Cloud provider. The *hardware* attributes defines the hardware capabilities of a node, and one can note the relation with the hardware requirements of a task presented in Figure 7.1. The *os* section lists all the virtual resources that can be provisioned on the nodes and the number of seconds it will require in average. One can note again the relation with the *os* requirements of a task presented in Figure 7.1. The attributes of a node are typically used by the scheduler of the Master module.

A Worker module is responsible of all the virtual resources, all the tasks that will be executed, and all the files that will be transferred on the node. The Worker module is not capable of making decisions and simply answers to the orders sent by the Master module.

There are four orders that can be treated by a Worker:

— launch a virtual resource;

— kill a virtual resource;

— execute a task on the virtual resource;

— download a file for the task from a remote node.

It can be noted that the number of orders is small, and that only the three first depends on the type of virtualization. Therefore, defining new types of Worker to manage different types of virtual resources is not difficult. In addition, unlike the public FaaS, the virtual resources provisioned by the Worker module may be unstacked and may run directly on the physical node, thus, better performance can be provided. One may also note, that a version of Worker module providing bare metal execution can also be easily specified. In the evaluation section 7.3, an implementation of different worker module using the library Scala AKKA is presented. An example of the orders launch a VM and kill a VM are presented in Figure 7.4. In this example, the parameters *os*, is the name of the image used to produce the VM, the parameters *capas* is a dictionary containing the hardware information of the VM, the parameter *user* is the name of the user that will have access to the VM provisioned - this parameter is used to mount the local directory (on the node) containing all the files belonging to that user -, and finally the parameter *script* is the list of commands to run once the VM is booted before launching any task on it - this script is used to install the software dependencies.

```
1       case DaemonProto.LaunchVM (mid, id, os, capas, user, script) =>
2           log.info (s"Launch vm : $id, with $os of $capas \
3                       for $user with $script")
4           KVM.launchVM (mid, id, os, capas, user, script, masterModule)
5
6       case DaemonProto.KillVM (mid, id) =>
7           log.info (s"Kill vm : $id")
8           KVM.killVM (mid, id, masterModule)
```

Figure 7.4 – Example of orders in Scala AKKA implementation of a KVM Worker

A cluster can contain heterogeneous Worker modules, *e.g.*, Workers providing container virtualization, along with workers providing VM virtualization. The **objectives 5 and 6** - *Dedicated resource management* as no stacking of the virtualization is made, and *Modular virtual resources* - of Table 3.5 are achieved thanks to the Worker module.

The last order is used to transfer files created by a task of the workflow during the execution, and ensures the dependencies between the tasks. To enable the transfer of the different files, each module instance (master and workers) is running along with a file server, that is hosting all the files produced during the execution of the workflows. When a Worker module requires access to files, it will simply get them by a file download request. In the workflow definition, multiple tasks could depend on the same file. For this reason, when multiple tasks are to be executed by the same Worker, and have the same file dependency, the file transfers are merged in order to prevent multiple copies of the files that would be useless. All the file transfers are managed by the Worker module automatically, thus achieving the **objective 3** - *Automatic file management* - of Table 3.5. In addition, all the task executions are launched by the Worker, achieving the **objective 2** - *Automatic task execution.*

As specified previously, the operating system that can be provisioned may be customized by deriving from a standard OS. All the dependencies are automatically installed by the Worker module. This achieves the **objective 1** - *Software dependencies management* - of Table 3.5.

Finally, virtual resources are provisioned only when required by the workers, and resources can be released when becoming useless, therefore, resource provisioning is elastic. This achieves the **objective 4** - *Elastic resources* - of Table 3.5.

Multiple cluster infrastructure - Federation of services

Until now we have described the WaaS that manages the resources of a single cluster of nodes. Unlike the solution presented in the state of the art, the WaaS is designed to be deployed by Cloud providers on their infrastructures as a new specific service. Therefore, the WaaS is designed to handle multiple workflow submissions from many users simultaneously. Most of the time the infrastructures offered by Cloud providers are composed of multiple clusters (assuming that a cluster refers to a single local area network). In the case of multi-cluster, scalability issues are raised, *e.g.*, a single Master managing many Workers, heterogeneous bandwidth, bottlenecks, etc. The problem of scalability addressed in this chapter is due to the number of events that need to be handled by the Master module. By dividing the planning execution between multiple Master modules, the number of events that are managed by a single Master are reduced (less Worker, tasks, files and virtual resources to manage), thus enhancing the scalability of the solution.

One solution would have been to deploy as many WaaS services as the number of clusters, and let the end-users choose on which cluster to execute their workflows. However, in case of big workflows or when clusters are highly loaded, a workflow can be poorly executed on a single cluster, while using multiple clusters could improve the execution quality. For this reason, we have opted for a federated service where each cluster is handled by one Master and where a Leader is deployed to federate the Master modules. In this federated version, the Leader is the only one making decisions and possessing a scheduler submodule.

Thus, instead of calling the scheduler, each Master module receives a planning to apply from the Leader module. The planning received by a Master module only involves the nodes of the cluster under its supervision, as well as only the tasks that will be executed on this cluster. Workers from different clusters are not accessible from two different clusters. However, the different Master modules can communicate to handle file transfers from one cluster to another.

In this section, a new Cloud service for the execution of scientific workflows has been described. This service is usable in a context of multiple cluster infrastructure providing a good turnkey solution for Cloud provider. This achieves the **objective 9** - *Turnkey solution* for the Cloud provider - of Table 3.5.

To conclude one can note that since a distributed infrastructure with several clusters is targeted, decentralized scheduling algorithms would be more suitable. However, there is a plethora of scheduling algorithms in the literature that have been designed for centralized decision making. The WaaS service has been designed so that these algorithms can be used without any modification. Considering distributed scheduling algorithms could be the subject of future work, in order to solve scalability issues in terms of decision making.

7.3 Evaluation

In this section, a detailed evaluation of the WaaS Cloud service is presented. Each section refers to the achievements of the objectives of Table 3.5. Furthermore the experimental protocol is detailed hereafter. All codes and results are available on a public *git* repository [1].

1. `https://gitlab.inria.fr/ecadorel/waas/`

Experimental protocol

In this section is presented the experimental protocol used to validate the WaaS cloud service. Indeed, this section presents the experimental infrastructure as well as the application executed within experiments.

WaaS service implementation

The WaaS service has been implemented using the library Scala Akka, the source code of this implementation is available on the git repository. The architecture of the platform has been presented in Section 7.2. The Scala Akka library is an actor oriented library. In the WaaS service implementation, each module is an actor answering and sending messages to the other modules. Four different scheduling algorithms of the literature have been implemented, HEFT [122], Min-Min, Max-Min [89] and OnlyUsedNodes. HEFT, Min-Min and Max-Min have originally been designed for Grid scheduling, and have been adapted to be able to schedule virtual resources. This adaptation uses the resource selection algorithms presented in Section 5.3 (cf. Algorithm 10 and Algorithm 11). Furthermore, two different Workers have been implemented, one providing KVM virtual machines, and one providing Docker containers. Unfortunately, even if the utilization of dynamic algorithm is in theory possible, the implementation does not provide a possibility to update the planning of the Master modules, when the scheduler has modified them. Thus, the execution of NearDeadline is not possible with this implementation, even if it is not a conceptual problem of the WaaS.

Execution infrastructure

The experimentation has been run on a distributed infrastructure composed of two different clusters. Those two clusters are part of the Grid'5000 [13] experimental platform, and are presented in Table 7.1. The WaaS Cloud service has been deployed on this infrastructure with five Master modules. One of them handles the cluster *Econome*, and each of the four others handles a sub cluster composed of 11 nodes of *Ecotype*. In other words, *Ecotype* is divided in four sub clusters for the experiments. On the one hand, the Worker providing Docker virtualization has been instantiated and deployed both on the nodes of two of the *Ecotype* sub clusters, and on the nodes of the *Econome* cluster. On the other hand, the Worker providing KVM virtualization has been instantiated and deployed on the nodes of the two remaining sub clusters of *Ecotype*. The bandwidth between the *Ecotype* and *Econome* clusters is 10 Gbps.

Name	Nodes	CPU	Memory	Storage	Network
Econome	22	Intel Xeon E5-2660 2.20GHz, 2 CPUs/node 8 cores/CPU	64 GB	2.0 TB HDD SATA	10Gbps
Ecotype	44	Intel Xeon E5-2630L v4 1.80GHz, 2 CPUs/node 10 cores/CPU	128 GB	400 GB SSD	2 x 10Gbps

Table 7.1 – Description of the used infrastructure

Workload

The *Montage* workflow is a typical case-study used to evaluate workflow engines and scheduling algorithms [21, 39, 77, 81]. It is a complex workflow that integrates most of the workflow classes characterized by Bharathi et al. in [37]. The simulated workload is composed of 100 *Montage* workflows, each of them composed of 619 tasks. Table 6.7 lists the different tasks of *Montage* 619, and presents for each task its execution time (on an Ecotype node), its amount of input data and the amount of data it creates. The execution of these workflows has been performed four times using the four implemented scheduling algorithms (HEFT, Min-Min, Max-Min and OnlyUsedNodes). Each workflow is submitted by a new client when the Leader is ready to receive a new submission (*i.e.*, has treated the last one and updated the planning, so approximately every 3 to 4 seconds). The average duration of the workload execution for the four execution is 24 minutes.

Results

End-user concerns

The submission of the Montage workflow composed of 619 tasks has been made 400 times in the experiment. The end-users did not have to make any modification, even when different scheduling algorithms were deployed. The end-users also did not have to worry about the virtualization technologies that have been used, and chosen by the Cloud provider, and no resource management has been asked to the end-user. Therefore, the objective 8 is correctly achieved (*Hidden management for the end-user* - See Table 3.5). One can note that the description file of the Montage workflow in our experiments has been generated automatically by the workflow generator provided by Pegasus, and was easily adapted for the WaaS with a transformation script. This file [2] is 11977 lines long.

2. `https://gitlab.inria.fr/ecadorel/waas/tree/master/apps/Montage_619/flow.yaml`

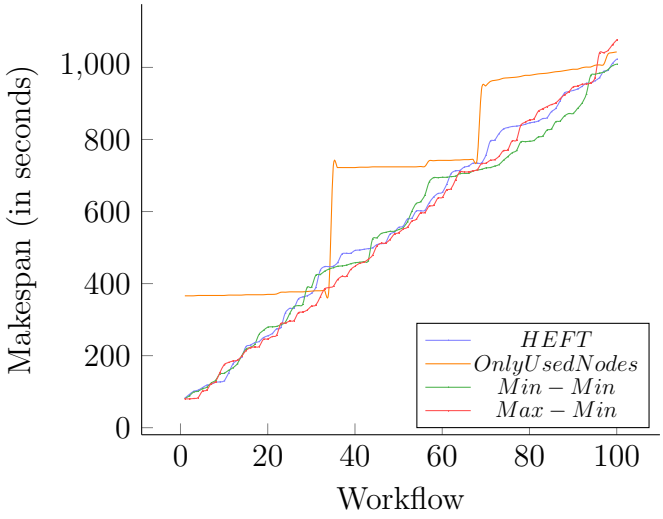


Figure 7.5 – Comparison of the makespan of each submitted workflow.

Modular scheduler

In this experimentation, we aimed at showing that different scheduling algorithms could be used or added by Cloud providers with minimal modifications. The code of these scheduling algorithms can be found on the git repository [3], where only one file of about 100 lines is required for each scheduler. The objective 7 (*Modular scheduler* - See Table 3.5) is therefore correctly achieved. Each scheduler has a different behavior and uses the infrastructure in a different way. We do not intend to compare the performances and results of the different algorithms, however, in order to validate the scheduler modularity Figure 7.5 presents the makespan of the Montage workflow (*i.e.*, execution time of the workflow) with the four different algorithms.

Modular Worker

Two different Workers have also been deployed on the infrastructure, providing different types of virtualization. We do not intend to compare the different virtualization techniques but to illustrate the ability of the WaaS to integrate and handle different Workers. The code of the Workers can be found on the git repository [4], where only one file differs from KVM to Docker. This file is 200 lines long for Docker and 500 for KVM. Table 7.2

3. `https://gitlab.inria.fr/ecadorel/waas/tree/master/source/src/main/scala/com/orch/leader/scheduling`
4. `https://gitlab.inria.fr/ecadorel/waas/tree/master/source/src/main/scala/com/orch/daemon/`

Virtualization	HEFT	OnlyUsedNodes	Min-Min	Max-Min
Docker	4728	1197	4923	3981
KVM	1749	0	1539	1658

Table 7.2 – Number of virtual resources provisioned during the experiment

Cluster	HEFT	OnlyUsedNodes	Min-Min	Max-Min
Ecotype-1	17095	21636	16752	16922
Ecotype-2	10815	21777	11248	11244
Ecotype-3	6051	0	5927	6196
Ecotype-4	6434	0	6144	6342
Econome-1	21505	18487	21829	21196

(a) Number of tasks executed

Cluster	HEFT	OnlyUsedNodes	Min-Min	Max-Min
Ecotype-1	287402	211825	281331	287402
Ecotype-2	104329	175991	112066	115575
Ecotype-3	65892	18	64917	65692
Ecotype-4	68613	17	65363	74851
Econome-1	174047	143578	177889	169672
estimation all	570891	498031	571523	570716

(b) Number of messages processed

Table 7.3 – Distribution of the workload across clusters

lists the number of virtual resources provisioned by KVM Workers, and Docker Workers in our experiment. VMs provisioned with KVM where Ubuntu 18.04 virtual machines, and each provisioning took 59 seconds, when the provisioning of the Docker container (also Ubuntu 18.04 container) took in average 3 seconds. Econome nodes have HDD instead of SSD harddrives, thus booting time is way slower on Econome than on Ecotype, explaining the difference between the boot times observed in this experimentation and the boot time observed in the experimentation presented in Chapter 6. The heterogeneity of virtual resources is not possible in other workflow engines. Furthermore, the solution provided by the WaaS service is elastic. For instance, with the HEFT algorithm, the execution of the first submitted workflow used 22 virtual resources with duration ranging from 11 seconds to 114 seconds with an average of 62 seconds. These virtual resources have executed 20 to 60 tasks each, with an average of 28 tasks. The makespan of the workflow in the experiment was 118 seconds. This shows that resources are created and deleted in an elastic manner, *i.e.*, with shorter lifetime than the overall makespan. The objectives 4, 5 and 6 are validated (*Elastic resources*, *Dedicated resource management*, *Modular virtual resources (Heterogeneity)* - See Table 3.5).

Multiple cluster infrastructure

This experimentation also shows the capability of the WaaS service to use a distributed infrastructure composed of multiple clusters. Table 7.3a lists the number of tasks that have been executed in each cluster. Every scheduler took approximately the same time to schedule one workflow, with an average of 2.5 seconds. The bottleneck of the execution is not located in the scheduling algorithm, but on the number of messages that

are transmitted, as illustrated in Table 7.3b by the number of messages treated by each Master module during the execution. Furthermore, the last line of this table shows an approximation of the number of messages that would have been treated if only one Master module had been used. The distribution of the work between the different clusters is correlated to the scheduler that is used, however, it may be noted that in all cases the workload is dispatched between multiple clusters.

7.4 Conclusion

In this chapter has been presented a new Cloud service for the execution of scientific workflows, namely the Workflow-as-a-Service. Unlike the related work, the WaaS adopts a *specific-service-vision* and is therefore presented as a turnkey solution for Cloud providers. In this chapter, we have shown that this conceptual approach of the workflow engine improves the separation of concerns between the end user, who is only responsible for the specification of the workflow, and the Cloud provider, who is responsible for the resource management and configuration. In addition, this approach, by leveraging its specificity for scientific workflows, improves the elasticity and optimality of resource scheduling and provisioning. To facilitate the adoption by the Cloud provider and improves the flexibility of the service, the WaaS has been made modular, thus facilitating the definition of new types of virtualization for Worker modules, and the integration of new schedulers into Master modules. Finally, the WaaS has been designed to be scalable, even when considering a complex distributed infrastructure with multiple clusters. The service has been tested on a real distributed infrastructure divided in five clusters, with four different scheduling algorithms, and two types of virtualization systems (KVM and Docker), and has shown consistency during the execution of hundreds of workflows.

Chapter 8

Conclusion

In this chapter, we first concludes the work and the contributions of this thesis on the scheduling and the execution of scientific workflows . Then perspectives for potential future works are discussed.

Contents

8.1 Achievements

In this thesis, we investigated the problem of the scheduling and the execution of scientific workflows. Effective execution of scientific workflows requires the use of complex multi-cluster infrastructure, but scientists developing these applications are not experts in infrastructure management. For this reason, most users (scientists) have opted for the use of workflow engines that aim to bring the execution of the workflow within a Cloud computing environment. However, these workflow engines assume that the physical infrastructure under the Cloud computing environment is managed by the Cloud provider and thus these engines have no impact on the placement of the virtual resources used to execute workflow tasks. In addition, Cloud providers have no information about the application that is running within their infrastructure, and therefore are not able to perform strong optimization regarding resource usage, such as energy consumption or resource sharing. This could lead to under-used physical machines, and under-optimized energy consumption. Nevertheless, energy consumption in datacenters is a major issue as it has a direct significant environmental impact, and has to be taken into account.

In this thesis, we shifted the problems of workflow tasks scheduling, and workflow execution from the user side to the Cloud provider side. To this end, new information about the topology of the application that are to be executed are made available to

the Cloud provider. By doing this we provided better optimization in terms of physical infrastructure usage and thus a better energy optimization as well as a good fairness among the users submitting workflows for execution.

This new approach to the management of scientific workflows has introduced a new problem. Indeed, by locating the scheduling decisions on the Cloud provider side, the scheduling problem to resolve is different from the problems that are resolved in the state of the art. Instead of virtual machines, that can provide infinite resources only limited by the end-user budget, limited number of physical machines providing limited number of resources had to be considered. In addition, the scheduling algorithm had to take into account that multiple users are sharing the same physical infrastructure. Once the scheduling problem has been resolved, the planning, that has been computed, has to be efficiently applied.

The problem resolved in this thesis can be separated into two sub problems: the resolution of the scheduling problem and the execution of the computed planning.

To resolve the scheduling problem we have presented two scheduling algorithms:

— **Static scientific workflow scheduling algorithm for energy optimization in a Cloud computing environment -** The objective of this algorithm - named OnlyUsedNodes and presented in Chapter 4 - is to minimize the number of physical machines required for the execution of a set of scientific workflows submitted by different users. In the problem resolved by this algorithm, a deadline is attached to every workflow. This deadline is used to transform the problem from a multi-objective problem (makespan and energy optimization) to a single objective problem. Thanks to a partial backtracking heuristic, the algorithm is able to reduce the number of required PMs, and by extension the energy consumption. This partial backtracking heuristic considers only already used PMs when scheduling the tasks of a workflow, and backtrack when the deadline of the workflow can no longer be met. As the deadline cannot be met, a new PM is considered for the scheduling of the workflow.

The OnlyUsedNodes algorithm considers virtual resources (Virtual Machines or containers) to resolve the problem of software dependencies of the applications, and also to give access only to a sub part of the resources of a PM to a user, and therefore enhance the resource usage of each PM. Thanks to that mechanism multiple users could use the same PM at the same time, while the isolation of the computation is still guaranteed.

Simulations (see Section 6.1) have shown a reduction of the number of required nodes as well as a reduction of the energy consumption of the infrastructure. Further experimentation (see Section 6.4) have shown that energy reduction is also observed on a real infrastructure and when executing real case scientific workflows.

— **Dynamic scientific workflow scheduling algorithm for user fairness and energy optimization -** The objective of this algorithm - named NearDeadline and presented in Chapter 5 - is to reduce the energy consumption of the physical infrastructure of a Cloud provider, answering to the submissions of multiple users. To every submitted workflow is attached a deadline that has to be met. This deadline allows to transform the problem from a multi-objective problem (energy and makespan minimization) to a single objective problem, with energy optimization objective. The deadline is also used to define the concept of user fairness, where a computed planning is consider fair when the deadlines of all the workflows (belonging to different users) are met, or when the deadline violation is distributed equally among the users. Unlike, OnlyUsedNodes, this algorithm considers that new submissions with higher priority may arrive at uncertain time, and therefore that the planning may have to be reconsidered in order to guarantee the fairness between the users. This algorithm is based on a two phase scheduling. The first phase tries to schedules a submitted workflow near its deadline - with the purpose of using as few resources as possible. When the first phase fails, the second phase schedule the workflows in best effort mode, after the workload with lower priority has been removed from the planning.

  As for OnlyUsedNodes, this algorithm relies on virtualization mechanisms in order to provide better resource usage of each PM and therefore optimize the energy consumption. This virtualization mechanisms are also used to guarantee the software requirements of the tasks. Experiments on a real infrastructure (see Section 6.4) have shown real benefits in term of equity and energy consumption, in comparison with state of the art algorithm.

Once the planning has been computed, the Cloud provider has to be able to apply it and perform the operations on the infrastructure. To this end, we have proposed the following contribution:

— **WaaS: Workflow as a Service, a Cloud computing service for scientific workflow execution -** The WaaS - presented in Chapter 7 - is a new Cloud service for the execution of scientific workflows. All the decisions are performed on

the Cloud provider side (scheduling, resource provisioning, file transfers, etc.), so as to reinforce the separation of concerns between the end-users (scientist) and the Cloud provider, in addition of improving the resource management from the Cloud provider's perspective. The planning computed by a scheduling algorithm lists the resources that have to be provisioned and the tasks that are to be executed on this resources. Depending on the size of the infrastructure and the size of the workload, a planning may contain many resources and many task assignments, and therefore the planning execution must be efficient in order to respect the decisions made by the scheduling algorithm.

The WaaS service has been designed to facilitate the integration by a Cloud provider that may want to use different virtualization mechanisms, or different scheduling algorithms. To this purpose, the WaaS is designed as a turnkey solution composed of customizable modules giving the opportunity to the Cloud provider to extend the solution with minimal modifications. This solution also allows the usage of a complex multi-cluster infrastructure, by providing a federation of sub-services, thus enhancing the resolution of scalability issues.

As the separation of concern is a major objective of the WaaS, the ease of use from the end-user point of view has also been investigated. Indeed, the end-users by using the service, only need to define the topology, the applications (executable tasks of the workflow), and the input files of their workflows. The topology of a workflow, is described by a single file, that contains all the required information for the Cloud provider, and that can be differently treated depending on the Cloud provider information requirements. As the end-users never have to manage anything but the description of their workflows, we claim that the separation of concern is correctly achieved.

The service has been tested on a real distributed infrastructure (see Section 7.3) divided in five clusters, with four different scheduling algorithms, and two types of virtualization systems (KVM and Docker), and has shown consistency during the execution of hundreds of workflows.

8.2 Perspectives

The contributions of this thesis could be extended in different ways. In this section is discussed research direction that could lead to promising results.

8.2.1 Prospects related to energy optimization

Powering off unused PMs

In our work in Chapter 4, the algorithm OnlyUsedNodes computes a static planning with objective of energy minimization by using as few machine as possible. It could be interesting to consider the shutdown technique in order to remove the idle energy consumption of the unused nodes, and minimize even more the energy consumption. In that case, the energy consumption of the power on and power off operations would have to be taken into account in the model, in order to define whether or not it is interesting to power off a machine, considering that it may have to be powered on at the next submission interval. This technique could also be investigated for the algorithm NearDeadline. Such technique has already been investigated in general cases [32, 100], but could be adapted to the specific case of scientific workflows.

Using DVFS when deadlines are flexible

In Chapter 4 and 5 a deadline is associated to each workflow in order to reduce the scheduling problem to a single objective problem, and help in minimizing the energy consumption. This deadline is used in the proposed algorithms to minimize the number of resources needed, and consolidate the nodes under usage. In addition to the consolidation technique, the DVFS technique could be investigated, by reducing the frequency of the PMs while the deadline of the workflow can be easily met. This technique could be used in both OnlyUsedNodes and NearDeadline, by computing the lowest possible frequency that still guarantee the respect of the deadlines. DVFS technique has already been investigated for the scheduling of scientific workflow in the state of the art, but in the context of Grid environment [84, 93]. An adaptation in the context of virtualized environment with heterogeneous applications could be an interesting future work.

8.2.2 Prospects related to service oriented execution

Enhancing the scalability of the scheduling phase

The WaaS could gain in scalability by decentralizing scheduling decisions. This could be done in different ways. First, by using decentralized algorithms, which is for the moment impossible, since all the scheduling decisions are made inside the Leader module that is unique and thus centralized. Second, by selecting before each scheduling decision the clusters that are the most promising for the execution. To this end, the size of the submitted workload would have to be computed in order to define the upper bound in term of number of required resources. This bound could be used to create a virtual cluster (cluster composed of machines that may belong to different clusters) on which the workload could be scheduled. That way, virtual clusters would be created on the fly, and the scheduling of high workload could be done in parallel as long as there is no intersection between the virtual clusters.

Cohabitation with other kind of services

We have seen in our work that creating a service oriented execution of scientific workflows is a good way of achieving separation of concerns between the end users and the Cloud provider. However, it can be understandable that Cloud providers may be reluctant to reserve a part of their infrastructure only for the execution of scientific workflows, provided that this part of the infrastructure may sometime be underused. It would be interesting to investigate the possibility to make cohabit an existent Cloud service, such as IaaS or FaaS with the WaaS, and use the resources that are not used by those services in order to execute workflows. That way, higher infrastructure usage would be possible.

BIBLIOGRAPHY

[1] *Amazon Lambda*. url: https://aws.amazon.com/fr/lambda/.

[2] *Amazon Web Services*. url: https://aws.amazon.com.

[3] *Ansible*. url: https://www.ansible.com/.

[4] *Apache Airflow*. url: https://airflow.apache.org/.

[5] *Apache Cassandra* url: http://cassandra.apache.org/.

[6] *Apache Hadoop* url: http://hadoop.apache.org/.

[7] *Argo Workflows*. url: https://argoproj.github.io/argo/.

[8] *Docker*. url: https://www.docker.com/.

[9] *gLite*. url: http://grid-deployment.web.cern.ch/grid-deployment/glite-web/.

[10] *Google App Engine*. url: https://cloud.google.com/appengine.

[11] *Google Cloud functions*. url: https://cloud.google.com/functions.

[12] *Google Cloud*. url: https://cloud.google.com.

[13] *Grid'5000*. url: https://www.grid5000.fr.

[14] *IBM Cloud*. url: https://cloud.ibm.com/login.

[15] *Linux Containers*. url: https://linuxcontainers.org/.

[16] *Microsoft Azure*. url: https://azure.microsoft.com/.

[17] *Microsoft HDInsight*. url: https://azure.microsoft.com/en-us/services/hdinsight/.

[18] *Puppet*. url: https://puppet.com/.

[19] *Singularity*. url: https://singularity.lbl.gov/.

[20] *Terraform*. url: https://www.terraform.io/.

[21] S. Abrishami, M. Naghibzadeh, and D. H. Epema. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds. *Future Generation Computer Systems*, 29(1):158 – 169, 2013. Including Special section: AIRCC-NetCoM 2009 and Special section: Clouds and Service-Oriented Architectures.

[22] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema. Cost-driven scheduling of grid workflows using partial critical paths. *IEEE Transactions on Parallel and Distributed Systems*, 23(8), 2012.

[23] E. N. Alkhanak, S. P. Lee, R. Rezaei, and R. M. Parizi. Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues. *Journal of Systems and Software*, 113:1 – 26, 2016.

[24] H. Arabnejad and J. Barbosa. Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, 2012.

[25] H. Arabnejad and J. G. Barbosa. A budget constrained scheduling algorithm for workflow applications. *Journal of Grid Computing*, 12(4):665–679, Dec 2014.

[26] H. Aziza and S. Krichen. A hybrid genetic algorithm for scientific workflow scheduling in cloud environment. *Neural Computing and Applications*, May 2020.

[27] B. Balis. Hyperflow: A model of computation, programming approach and enactment engine for complex distributed workflows. *Future Generation Computer Systems*, 55:147 – 162, 2016.

[28] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5), Oct. 2003.

[29] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

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

[31] M. Bencivenni, D. Bortolotti, A. Carbone, A. Cavalli, A. Chierici, S. Dal Pra, D. Girolamo, L. dell'Agnello, M. Donatelli, A. Fella, D. Galli, A. Ghiselli, D. Gregori, A. Italiano, R. Kumar, U. Marconi, B. Martelli, M. Mazzucato, M. Onofri, and S. Zani. Performance of 10 gigabit ethernet using commodity hardware. *IEEE Transactions on Nuclear Science*, 57:630–641, 05 2010.

[32] A. Benoit, L. Lefèvre, A.-C. Orgerie, and I. Raïs. Reducing the energy consumption of large-scale computing systems through combined shutdown policies with multiple constraints. *The International Journal of High Performance Computing Applications*, 32(1):176–188, 2018.

[33] F. Berman, H. Casanova, A. Chien, K. Cooper, H. Dail, A. Dasgupta, W. Deng, J. Dongarra, L. Johnsson, K. Kennedy, C. Koelbel, B. Liu, X. Liu, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, C. Mendes, A. Olugbile, M. Patel, D. Reed, Z. Shi, O. Sievert, H. Xia, and A. YarKhan. New grid scheduling and rescheduling methods in the grads project. *International Journal of Parallel Programming*, 33(2):209–229, Jun 2005.

[34] D. Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014.

[35] G. B. Berriman, J. C. Good, A. C. Laity, A. Bergou, J. Jacob, D. S. Katz, E. Deelman, C. Kesselman, G. Singh, M.-H. Su, and R. Williams. Montage: A grid enabled image mosaic service for the national virtual observatory. *Astronomical Data Analysis Software and Systems (ADASS) XIII*, 2003.

[36] L. Bertram, A. Ilkay, B. Chad, H. Dan, J. Efrat, J. Matthew, L. E. A., T. Jing, and Z. Yang. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[37] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. Su, and K. Vahi. Characterization of scientific workflows. In *2008 Third Workshop on Workflows in Support of Large-Scale Science*, pages 1–10, 2008.

[38] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, Jun 2009.

[39] L. F. Bittencourt and E. R. M. Madeira. Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds. *Journal of Internet Services and Applications*, 2(3):207–227, Dec 2011.

[40] M. Blasgen, J. Gray, M. Mitoma, and T. Price. The convoy phenomenon. *SIGOPS Oper. Syst. Rev.*, 13(2):20–25, Apr. 1979.

[41] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. volume 2, pages 759 – 767 Vol. 2, 06 2005.

[42] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous

distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810 – 837, 2001.

[43] R. Buyya, A. Beloglazov, and J. H. Abawajy. Energy-efficient management of data center resources for cloud computing: A vision, architectural elements, and open challenges. *CoRR*, abs/1006.0308, 2010.

[44] J. Błażewicz, E. Pesch, and M. Sterna. The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research*, 127(2):317 – 331, 2000.

[45] E. Cadorel, H. Coullon, and J.-M. Menaud. Ordonnancement multi-objectifs de workflows dans un Cloud privé. In *ComPAS 2018 - Conférence d'informatique en Parallélisme, Architecture et Système*, pages 1–8, Toulouse, France, July 2018.

[46] E. Cadorel, H. Coullon, and J.-M. Menaud. A workflow scheduling deadline-based heuristic for energy optimization in Cloud. In *GreenCom 2019 - 15th IEEE International Conference on Green Computing and Communications*, Atlanta, United States, 2019. IEEE.

[47] E. Cadorel, H. Coullon, and J.-M. Menaud. Online Multi-User Workflow Scheduling Algorithm for Fairness and Energy Optimization. In *CCGrid2020 - 20th International Symposium on Cluster, Cloud and Internet Computing*, Melbourne, Australia, Nov. 2020.

[48] Y. Caniou, E. Caron, A. K. W. Chang, and Y. Robert. Budget-aware scheduling algorithms for scientific workflows with stochastic task weights on heterogeneous iaas cloud platforms. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 15–26, May 2018.

[49] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski. Serverless programming (function as a service). In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017.

[50] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, page 103–116, New York, NY, USA, 2001. Association for Computing Machinery.

[51] E. Coffman, M. Garey, and D. Johnson. Bin packing with divisible item sizes. *Journal of Complexity*, 3(4):406 – 428, 1987.

[52] G. Cook, J. Lee, T. Tsai, A. Kong, J. Deans, B. Johnson, and E. Jardim. Clicking clean:who is winning the race to build a green internet? *Greenpeace Inc.*, 2017.

[53] H. Coullon, G. Le Louët, and J.-M. Menaud. Virtual Machine Placement for Hybrid Cloud using Constraint Programming. In *ICPADS 2017*, Shenzhen, China, Dec. 2017.

[54] S. B. Davidson and J. Freire. Provenance and scientific workflows: Challenges and opportunities. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, page 1345–1350, New York, NY, USA, 2008. Association for Computing Machinery.

[55] M. Dayarathna, Y. Wen, and R. Fan. Data center energy consumption modeling: A survey. *IEEE Communications Surveys Tutorials*, 18(1):732–794, 2016.

[56] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. In M. D. Dikaiakos, editor, *Grid Computing*. Springer Berlin Heidelberg, 2004.

[57] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528 – 540, 2009.

[58] F. Deng, M. Lai, and J. Geng. Multi-workflow scheduling based on genetic algorithm. In *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 2019.

[59] R. Duan, T. Fahringer, R. Prodan, J. Qin, A. Villazón, and M. Wieczorek. Real world workflow applications in the askalon grid environment. In P. M. A. Sloot, A. G. Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, editors, *Advances in Grid Computing - EGC 2005*, pages 454–463, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[60] J. J. Durillo, V. Nae, and R. Prodan. Multi-objective energy-efficient workflow scheduling using list-based heuristics. *Future Generation Computer Systems*, 36:221 – 236, 2014. Special Section: Intelligent Big Data Processing Special Section: Behavior Data Security Issues in Network Information Propagation Special Section: Energy-efficiency in Large Distributed Computing Architectures Special Section: eScience Infrastructure and Applications.

[61] R. Ferreira da Silva, T. Glatard, and F. Desprez. Controlling fairness and task granularity in distributed, online, non-clairvoyant workflow executions. *Concurrency and Computation: Practice and Experience*, 26(14), 2014.

[62] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[63] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. 01 2003.

[64] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *The International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[65] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop*, pages 1–10, 2008.

[66] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, 79(8):1230 – 1242, 2013.

[67] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, pages 238–239. W.H. Freeman and Co., 1979.

[68] M. Ghose, P. Verma, S. Karmakar, and A. Sahu. Energy efficient scheduling of scientific workflows in cloud environment. In *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 170–177, 2017.

[69] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *Computer*, 40(12):24–32, 2007.

[70] A. Gupta, L. Kalé, D. Milojicic, P. Faraboschi, R. Kaufmann, V. March, F. Gioachin, C. Suen, and B. Lee. The who, what, why and how of high performance computing applications in the cloud. volume 1, 12 2013.

[71] R. A. Haidri, C. P. Katti, and P. C. Saxena. Cost effective deadline aware scheduling strategy for workflow applications on virtual machines in cloud computing. *Journal of King Saud University - Computer and Information Sciences*, 2017.

[72] H. A. Hassan, S. A. Salem, and E. M. Saad. A smart energy and reliability aware scheduling algorithm for workflow execution in dvfs-enabled cloud environment. *Future Generation Computer Systems*, 112:431 – 448, 2020.

[73] E. S. H. Hou, N. Ansari, and Hong Ren. A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2):113–120, 1994.

[74] C. Hsu and S. W. Poole. Power signature analysis of the specpower_ssj2008 benchmark. In *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*, pages 227–236, April 2011.

[75] W. Huber, V. J. Carey, R. Gentleman, S. Anders, M. Carlson, B. S. Carvalho, H. C. Bravo, S. Davis, L. Gatto, T. Girke, R. Gottardo, F. Hahne, K. D. Hansen, R. A. Irizarry, M. Lawrence, M. I. Love, J. MacDonald, V. Obenchain, A. K. Oleś, H. Pagès, A. Reyes, P. Shannon, G. K. Smyth, D. Tenenbaum, L. Waldron, and M. Morgan. Orchestrating high-throughput genomic analysis with bioconductor. *Nature Methods*, 2015.

[76] Jia Yu, R. Buyya, and Chen Khong Tham. Cost-based scheduling of scientific workflow applications on utility grids. In *First International Conference on e-Science and Grid Computing (e-Science'05)*, 2005.

[77] Q. Jiang, Y. Lee, and A. Zomaya. Serverless execution of scientific workflows. 2017.

[78] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3), 2013. Special Section: Recent Developments in High Performance Computing and Security.

[79] M. Kaplan, W. Forrest, and N. Kindler. Revolutionizing data center energy efficiency. Technical report, 2008.

[80] C. Kenyon et al. Best-fit bin-packing with random order. In *SODA*, volume 96, pages 359–364, 1996.

[81] J. Kijak, P. Martyna, M. Pawlik, B. Balis, and M. Malawski. Challenges for scheduling scientific workflows on cloud functions. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018.

[82] L. Kleinrock. Analysis of a time-shared processor. *Naval Research Logistics Quarterly*, 11(1):59–73, 1964.

[83] D. Kliazovich, P. Bouvry, and S. U. Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012.

[84] J. Kołodziej, S. U. Khan, L. Wang, and A. Y. Zomaya. Energy efficient genetic-based schedulers in computational grids. *Concurrency and Computation: Practice and Experience*, 27(4):809–829, 2015.

[85] G. Le Louët and J.-M. Menaud. OptiPlace: designing cloud management with flexible power models through constraint programing. In *International Conference on Utility and Cloud Computing*, Dresden, Germany, Dec. 2013.

[86] L. Leslie, C. Sato, Y. Lee, Q. Jiang, and A. Zomaya. Dewe: A framework for distributed elastic scientific workflow execution. *Conferences in Research and Practice in Information Technology Series*, 163:3–10, 01 2015.

[87] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, and L. B. D. Lea. Nist cloud computing reference architecture. Technical report, Gaithersburg, MD, United States, 2011.

[88] J. Liu, J. Ren, W. Dai, D. Zhang, P. Zhou, Y. Zhang, G. Min, and N. Najjari. Online multi-workflow scheduling under uncertain task execution time in iaas clouds. *IEEE Transactions on Cloud Computing*, 2019.

[89] M. Maheswaran, S. Ali, H. Siegal, D. Hensgen, and R. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. pages 30–44, 02 1999.

[90] G. Mateescu, W. Gentzsch, and C. J. Ribbens. Hybrid computing—where hpc meets grid and cloud computing. *Future Generation Computer Systems*, 27(5):440 – 453, 2011.

[91] V. Mathew, R. Sitaraman, and P. Shenoy. Energy-aware load balancing in content delivery networks. *Proceedings - IEEE INFOCOM*, 09 2011.

[92] P. M. Mell and T. Grance. Sp 800-145. the nist definition of cloud computing. Technical report, Gaithersburg, MD, United States, 2011.

[93] M.-S. Mezmaz, N. Melab, Y. Kessaci, Y.-C. Lee, E.-G. Talbi, A. Zomaya, and D. Tuyttens. A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *Journal of Parallel and Distributed Computing*, 71:1497–1508, 11 2011.

[94] K. Miettinen. *Nonlinear Multiobjective Optimization*. 1999.

[95] A. Muller and S. Wilson. *Virtualization with Vmware Esx Server*. Syngress Publishing, first edition, 2005.

[96] T. L. Nguyen and A. Lebre. Virtual Machine Boot Time Model. In *PDP 2017 - 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 430 – 437, St Peterbourg, Russia, Mar. 2017.

[97] T. L. Nguyen and A. Lebre. Conducting Thousands of Experiments to Analyze VMs, Dockers and Nested Dockers Boot Time. Research Report RR-9221, INRIA, 2018.

[98] J. Ni and X. Bai. A review of air conditioning energy performance in data centers. *Renewable and Sustainable Energy Reviews*, 67:625 – 640, 2017.

[99] V. Nitu, B. Teabe, L. Fopa, A. Tchana, and D. Hagimont. Stopgap: elastic vms to enhance server consolidation. *Software: Practice and Experience*, 47(11):1501–1519, 2017.

[100] A. Orgerie, L. Lefèvre, and J. Gelas. Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems. In *2008 14th IEEE International Conference on Parallel and Distributed Systems*, pages 171–178, 2008.

[101] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin. Performance evaluation of virtualization technologies for server consolidation. 2007.

[102] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder. Heuristics for vector bin packing. January 2011.

[103] J. Pastor and J. M. Menaud. Seduce: a testbed for research on thermal and power management in datacenters. In *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2018.

[104] J. pastor and J. M. Menaud. Seduce: Toward a testbed for research on thermal and power management in datacenters. In *Proceedings of the Ninth International Conference on Future Energy Systems*, e-Energy '18, page 513–518, New York, NY, USA, 2018. Association for Computing Machinery.

[105] M. Poess and R. Nambiar. Energy cost, the key challenge of today's data centers: A power consumption analysis of tpc-c results. *PVLDB*, 1:1229–1240, 08 2008.

[106] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, July 1974.

[107] A. Qumranet, Y. Qumranet, D. Qumranet, U. Qumranet, and A. Liguori. Kvm: The linux virtual machine monitor. *Proceedings Linux Symposium*, 15, 01 2007.

[108] R. Rajak. A comparative study: Taxonomy of high performance computing (hpc). *International Journal of Electrical and Computer Engineering*, 8:3386–3391, 10 2018.

[109] L. Reiter, O. Rinner, P. Picotti, R. Hüttenhain, M. Beck, M.-Y. Brusniak, M. O. Hengartner, and R. Aebersold. mprophet: automated data processing and statistical validation for large-scale srm experiments. *Nature Methods*, 8:430 EP –, 03 2011.

[110] M. A. Rodriguez and R. Buyya. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. *IEEE Transactions on Cloud Computing*, 2(2):222–235, April 2014.

[111] H. L. Röst, Y. Liu, G. D'Agostino, M. Zanella, P. Navarro, G. Rosenberger, B. C. Collins, L. Gillet, G. Testa, L. Malmström, and R. Aebersold. Tric: an automated alignment strategy for reproducible protein quantification in targeted proteomics. *Nature Methods*, 13:777 EP –, 08 2016.

[112] H. L. Röst, G. Rosenberger, P. Navarro, L. Gillet, S. M. Miladinovic, O. T. Schubert, W. Wolski, B. C. Collins, J. Malmström, L. Malmström, and R. Aebersold. Openswath enables automated, targeted analysis of data-independent acquisition ms data. *Nature Biotechnology*, 32(3):219–223, 2014.

[113] K. S and M. Nair. Bin packing algorithms for virtual machine placement in cloud computing: a review. *International Journal of Electrical and Computer Engineering (IJECE)*, 9:512, 02 2019.

[114] D. Sahu, K. Singh, M. Manju, D. Taniar, L. Tuan, L. Son, M. Abdel-Basset, and H. Viet Long. Heuristic search based localization in mobile computational grid. *IEEE Access*, PP:1–1, 06 2019.

[115] A. Shehabi, S. Smith, D. Sartor, M. Herrlin, R. Brown, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner. United states data center energy usage report, 06 2016.

[116] J. Shi, J. Luo, F. Dong, and J. Zhang. A budget and deadline aware scientific workflow resource provisioning and scheduling mechanism for cloud. In *Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 672–677, 2014.

[117] D. Sirisha and G. Vijayakumari. Exploring the efficacy of branch and bound strategy for scheduling workflows on heterogeneous computing systems. *Procedia Computer*

Science, 93:315 – 323, 2016. Proceedings of the 6th International Conference on Advances in Computing and Communications.

[118] Y. Sotskov and N. Shakhlevich. Np-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics*, 59(3):237 – 266, 1995.

[119] M. Srinivas and L. M. Patnaik. Genetic algorithms: a survey. *Computer*, 27(6):17–26, 1994.

[120] A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. 2006.

[121] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. *Condor: A Distributed Job Scheduler*, page 307–350. MIT Press, Cambridge, MA, USA, 2001.

[122] H. Topcuouglu, S. Hariri, and M.-y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(3):260–274, Mar. 2002.

[123] K. Vahi, M. Rynge, G. Papadimitriou, D. Brown, R. Mayani, R. Ferreira da Silva, E. Deelman, A. Mandal, E. Lyons, and M. Zink. Custom execution environments with containers in pegasus-enabled scientific workflows. In *2019 15th International Conference on eScience (eScience)*, pages 281–290, 2019.

[124] W. [van der Aalst] and A. [ter Hofstede]. Yawl: yet another workflow language. *Information Systems*, 30(4):245 – 275, 2005.

[125] L. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: Towards a cloud definition. *Computer Communication Review*, 39:50–55, 01 2009.

[126] C. Vecchiola, S. Pandey, and R. Buyya. High-performance cloud computing: A view of scientific applications. *I-SPAN 2009 - The 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, 10 2009.

[127] J. Wainer, M. Weske, G. Vossen, and C. B. Medeiros. Scientific workflow systems, 1997.

[128] J. Watson. Virtualbox: bits and bytes masquerading as machines. *Linux Journal*, 2008(166):1, 2008.

[129] L. M. Weber, M. Nowicka, C. Soneson, and M. D. Robinson. diffcyt: Differential discovery in high-dimensional cytometry via high-resolution clustering. *bioRxiv*, 2019.

[130] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the askalon grid environment. *SIGMOD Rec.*, 34(3):56–62, Sept. 2005.

[131] A. S. Wu, H. Yu, S. Jin, K. . Lin, and G. Schiavone. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 15(9):824–834, 2004.

[132] C. Q. Wu, X. Lin, D. Yu, W. Xu, and L. Li. End-to-end delay minimization for scientific workflows in clouds under budget constraint. *IEEE Transactions on Cloud Computing*, 3(2):169–181, 2015.

[133] H. Wu, X. Chen, X. Song, C. Zhang, and H. Guo. Scheduling large-scale scientific workflow on virtual machines with different numbers of vcpus. *The Journal of Supercomputing*, Apr 2020.

[134] H. Wu, S. Ren, G. Garzoglio, S. Timm, G. Bernabeu, K. Chadwick, and S. Noh. A reference model for virtual machine launching overhead. *IEEE Transactions on Cloud Computing*, 4(3):250–264, 2016.

[135] W. Wu, W. Lin, and Z. Peng. An intelligent power consumption model for virtual machines under cpu-intensive workload in cloud environment. *Soft Computing*, 21(19):5755–5764, Oct 2017.

[136] B. Xavier, T. Ferreto, and L. Jersak. Time provisioning evaluation of kvm, docker and unikernels in a cloud platform. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2016.

[137] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, 34(3):44–49, Sept. 2005.

[138] J. Yu, R. Buyya, and K. Ramamohanarao. *Workflow Scheduling Algorithms for Grid Computing*, pages 173–214. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[139] M. Yue. A simple proof of the inequality ffd (l) $\leq$ 11/9 opt (l) + 1, $\forall$l for the ffd bin-packing algorithm. *Acta Mathematicae Applicatae Sinica*, 7:321–331, 1991.

[140] H. Zhao and R. Sakellariou. An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm. In H. Kosch, L. Böszörményi, and H. Hellwagner, editors, *Euro-Par 2003 Parallel Processing*, pages 189–194, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Prise en compte de l'énergie dans la gestion des workflows scientifiques dans le Cloud : Une vision centrée sur le fournisseur de service

Mot clés : Workflows scientifiques ; fournisseur de services de Cloud ; ordonnancement ; exécution ; optimisation energétique ; systèmes distribués ; infrastructures distribuées

Résumé : Les simulations scientifiques par ordinateur sont généralement très complexes et se caractérisent par de nombreux processus parallèles. Afin de mettre en évidence les parties parallèlisables, et de permettre une exécution efficace, de nombreux scientifiques ont choisi de définir leurs applications sous forme de workflows. Un workflow scientifique représente une application comme un ensemble de tâches de traitement unitaires, liées par des dépendances. De nos jours, grâce à leur faible coût, leur élasticité et leur aspect à la demande, les services de cloud computing sont largement utilisés pour l'exécution de workflows. Les utilisateurs utilisant ce type d'environnement gèrent l'exécution de leur workflow, ainsi que les ressources nécessaires, à l'aide de service standards tel que le IaaS (Infrastructure-as-a-Service). Néanmoins, comme les services de cloud ne sont pas spécifiques à la nature de l'application à exécuter, l'utilisation des ressources physiques n'est pas aussi optimisée qu'elle pourrait l'être. Dans cette thèse, nous proposons de déplacer la gestion et l'exécution des workflows du côté du fournisseur de Cloud afin d'offrir un nouveau type de service dédié aux workflows. Cette nouvelle approche rend possible une amélioration de la gestion des ressources et une réduction de la consommation d'énergie et de ce fait l'impact environnemental de l'infrastructure utilisée.

Title: Energy-aware management of scientific workflows in the Cloud: A Cloud provider-centric vision

Keywords: Scientific workflows; cloud service provider; scheduling: execution; energy efficiency; distributed systems; distributed infrastructures

Abstract: Scientific computer simulations are generally very complex and are characterized by many parallel processes. In order to highlight the parts that can be parallelized, and to enable efficient execution, many scientists have chosen to define their applications as workflows. A scientific workflow represents an application as a set of unitary processing tasks, linked by dependencies. Today, because of their low cost, elasticity, and on-demand nature, cloud computing services are widely used for workflow execution. Users using this type of environment manage the execution of their workflow, as well as the necessary resources, using standard services such as IaaS (Infrastructure-as-a-Service). However, because cloud services are not specific to the nature of the application to be executed, the use of physical resources is not as optimized as it could be. In this thesis, we propose to move the management and execution of workflows to the cloud provider's side in order to offer a new type of service dedicated to workflows. This new approach makes it possible to improve resource management and reduce energy consumption and thus the environmental impact of the infrastructure used.