N° d'ordre :

REPUBLIQUE ALGERIENNE DEMOCRATIQUE & POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR & DE LA RECHERCHE SCIENTIFIQUE


UNIVERSITE DJILLALI LIABES - SIDI BEL ABBES
FACULTE DES SCIENCES EXACTES
DEPARTEMENT D'INFORMATIQUE

# THESE DE DOCTORAT EN SCIENCES

Présentée par

Mohammed BENHAMMOUDA

Spécialité : Informatique
Option : Systèmes d'information et de connaissances (SIC)

Intitulée

# Contribution à l'optimisation d'ordonnancement de workflows dans un environnement cloud

**Soutenue le 03/01/2021**

**Devant le jury composé de :**

**Examinateurs :**

| | | |
|---|---|---|
| Mr. BOUKLI-HACENE Sofiane | Professeur à l'université de Sidi Bel Abbès | Président |
| Mr. TOUMOUH Adil | M.C.A à l'université de Sidi Bel Abbès | Examinateur |
| Mr. AMAR BENSABER Djamel | M.C.A à l'ESI de Sidi Bel Abbès | Examinateur |
| Mr. BERRABAH Djamal | M.C.A à l'université de Sidi Bel Abbès | Examinateur |
| Mr. BOUCHIHA Djelloul | Professeur au centre universitaire de Naama | Examinateur |

**Directeur de thèse :** Mr MALKI Mimoun, Professeur à l'ESI de Sidi Bel Abbès

Année universitaire 2020/2021

# Dédicaces

Je dédie ce modeste travail

à ma femme

à mes deux enfants

et à ma grande famille

# Remerciements

Je tiens tout d'abord à remercier mon directeur de thèse Monsieur MALKI Mimoun, Professeur à l'école supérieure en Informatique de Sidi Bel Abbes pour sa disponibilité, ses conseils judicieux et son orientation fruits de sa longue expérience, ses critiques fondées, son encouragement et sa grande patience.

Je tiens aussi à remercier vivement les membres du jury Monsieur BOUKLI-HACENE Sofiane, Professeur à l'université de Sidi Bel Abbès, Monsieur TOUMOUH Adil, Maître de conférences A à l'université de Sidi Bel Abbès, Monsieur AMAR BENSABER Djamel, Maître de conférences A à l'ESI de Sidi Bel Abbès, Monsieur BERRABAH Djamal, Maître de conférences A à l'université de Sidi Bel Abbès, Monsieur BOUCHIHA Djelloul, Professeur au centre universitaire de Nâama, pour avoir accepté d'examiner mon modeste travail.

Enfin, je remercie toutes les personnes qui m'ont aidé de près ou de loin durant mon travail de thèse.

# Résumé

Le cloud computing, initialement développée à partir de l'informatique en grille (grid computing), de l'informatique distribuée et de l'informatique parallèle, consiste en un ensemble d'ordinateurs virtuels interconnectés et utilisés dynamiquement, présentés sous forme de ressources informatiques unifiées basées sur des accords de niveau de service (SLA) établis par négociation entre le fournisseur de services, et les consommateurs, il est de plus en plus considéré comme un nouveau moyen d'utiliser les services informatiques, de stockage et de réseau à la demande de manière transparente et efficace.

Plusieurs applications scientifiques dans de nombreux domaines, tels que la bioinformatique et l'astronomie, impliquent généralement de nombreuses tâches contraintes par des relations de priorité. Ces applications sont souvent exprimées sous forme de flux de travail, appelé aussi workflow, une série d'étapes de traitement connexes (tâches), où chaque étape prend des données en entrée, effectue un traitement et produit des données en sortie qui seront transmises aux étapes de traitement suivantes. Malheureusement, les services de cloud computing ne sont pas gratuits, un contrat d'accord de niveau de service (Service Level Agreement) signé entre le fournisseur et l'utilisateur fixera le prix à payer pour les ressources louées dans un modèle de paiement à l'utilisation. La planification des tâches est une étape importante lors de l'exécution d'un workflow dans un environnement de cloud computing. L'algorithme de planification doit trouver une répartition efficace des tâches du workflow de l'utilisateur sur le pool de ressources du fournisseur, en minimisant le temps nécessaire à l'exécution des workflow de l'utilisateur et en optimisant la disponibilité des ressources du cloud. En outre, la planification des tâches dans un environnement de cloud computing est un problème NP-difficile en général, et les algorithmes d'optimisation heuristiques sont largement utilisés pour la résolution de ce genre de problèmes. Un algorithme heuristique est utilisé pour trouver une solution sous-optimale du problème.

Dans cette thèse, nous abordons le problème de la planification des workflows sur une infrastructure hétérogène de cloud computing, et une approche basée sur la parallélisation de l'algorithme de recuit simulé appelé PSA (Parallel

Simulated Annealing) implémenté sur une architecture GPU est proposée. Sa mission est de planifier les tâches de chaque workflow destinée à utiliser les ressources de l'environnement cloud. La planification consiste à affecter chaque tâche du workflow à un processeur donné pour son exécution. Le processus doit être effectué en respectant la contrainte de précédence qui consiste à respecter l'ordre des tâches dans le workflow en fonction de leurs interdépendances, tout en minimisant le temps de calcul (makespan) qui a un impact sur la consommation d'énergie et le coût dans le cas où l'utilisateur dispose d'un budget limité.

**Mots clés :** Cloud computing, Planification des tâches, Workflow, Flux de travail, Gpu, Recuit simulée.

# ABSTRACT

Cloud computing, initially developed from grid computing, distributed computing, and parallel computing, consists of a set of interconnected and dynamically used virtual machines presented as computing resources. service-based level agreements (SLAs) established by negotiation between the service provider and consumers, it is increasingly seen as a new way to use IT, storage, and network services for request in a transparent and efficient way.

Many scientific applications in many fields, such as bioinformatics and astronomy, usually involve many tasks constrained by priority relationships. These applications are often expressed as a workflow, also called a workflow, a series of related processing steps (tasks), where each step takes input data, performs processing, and outputs data that will be passed to users. Unfortunately, cloud services are not free, a service level agreement signed between the provider and the user will set the price to pay for the leased resources in a payment model use. Scheduling tasks is an important step when running a workflow in a cloud computing environment. The scheduling algorithm must find an efficient allocation of the user's workflow tasks to the provider's resource pool, minimizing the time required to execute the user's workflow and maximizing the availability of cloud resources. In addition, scheduling tasks in a cloud computing environment is an NP-hard problem in general, and heuristic optimization algorithms are widely used for solving such problems. A heuristic algorithm is used to find a sub-optimal solution of the problem.

In this thesis, we address the problem of planning workflows on a heterogeneous cloud computing infrastructure, and an approach based on the parallelization of the simulated annealing algorithm called PSA (Parallel Simulated Annealing) implemented on a GPU architecture is proposed. Its mission is to plan the tasks of each workflow intended to use the resources of the cloud environment. Planning involves assigning each task in the workflow to a given processor for execution. The process must be carried out respecting the precedence constraint of respecting the order of tasks in the workflow according to their interdependencies, while minimizing the computing time (makespan) that has an impact on the cost in the case where the user has a limited budget.

**Keywords: Cloud Computing, Task Scheduling, Workflow, Gpu, Simulated Annealing**

# مُلَخَّص

تتكون الحوسبة السحابية ، التي تم تطويرها مبدئيًا من حوسبة الشبكة ، والحوسبة الموزعة ، والحوسبة المتوازية ، من مجموعة من الأجهزة الافتراضية المترابطة والمستخدمة ديناميكيًا والتي تم تقديمها كموارد للحوسبة. اتفاقيات مستوى الخدمة المستندة إلى الخدمة (SLAs) التي تم التوصل إليها عن طريق التفاوض بين مزود الخدمة والمستهلكين ، يُنظر إليها بشكل متزايد على أنها طريقة جديدة لاستخدام خدمات تكنولوجيا المعلومات والتخزين والشبكات عند الطلب بطريقة شفافة وفعالة.

تتضمن العديد من التطبيقات العلمية في العديد من المجالات ، مثل المعلوماتية الحيوية وعلم الفلك ، العديد من المهام التي تقيدها علاقات الأولوية. يتم التعبير عن هذه التطبيقات غالبًا على أنها workflow "سير عمل" أو "تدفق عمل" ، وهي سلسلة من خطوات المعالجة (المهام) ذات الصلة ، حيث تأخذ كل خطوة بيانات الإدخال ، وتؤدي المعالجة ، وتخرج البيانات التي سيتم تمريرها إلى المستخدمين. بعد خطوات المعالجة. لسوء الحظ ، الخدمات السحابية ليست مجانية ، فاتفاقية مستوى الخدمة الموقعة بين المزود والمستخدم ستحدد السعر لدفع الموارد المستأجرة في نموذج الدفع عند الإستخدام. تعد مهام الجدولة خطوة مهمة عند تشغيل workflow في بيئة الحوسبة السحابية. يجب أن تجد خوارزمية الجدولة تخصيصًا فعالًا لمهام سير عمل المستخدم ، مما يقلل من الوقت اللازم لتنفيذ سير عمل المستخدم ويزيد من توفر الموارد السحابية إلى الحد الأقصى . بالإضافة إلى ذلك ، تعد جدولة المهام في بيئة الحوسبة السحابية مشكلة NP صعبة بشكل عام ، وتستخدم خوارزميات التحسين الاسترشادي على نطاق واسع لحل مثل هذه المشكلات

في هذه الأطروحة ، نعالج مشكلة تخطيط سير العمل workflow على بنية تحتية للحوسبة السحابية غير المتجانسة ، ويقترح نهج قائم على موازاة خوارزمية محاكاة الصُلب تسمى (Parallel Simulated Annealing) PSA المنفذة على بنية GPU . تتمثل مهمتها في تخطيط مهام كل سير عمل يهدف إلى استخدام موارد بيئة السحابة. يتضمن التخطيط تعيين كل مهمة في سير العمل لمعالج معين للتنفيذ. يجب أن يتم تنفيذ العملية مع مراعاة قيود الأسبقية المتمثلة في احترام ترتيب المهام في سير العمل وفقًا لترابطها ، مع تقليل وقت الحوسبة (المكوّن) الذي يؤثر على استهلاك الطاقة إلى الحد الأدنى والتكلفة في حالة ما إذا كان المستخدم لديه ميزانية محدودة.

**الكلمات المفتاحية :** الحوسبة السحابية ، جدولة المهام ، سير العمل ، وحدة معالجة الرسومات.

# Sommaire

# Liste des figures

# Liste des tableaux

# Chapitre 1 : Introduction générale

## 1.1 Contexte

Le cloud computing, initialement développée à partir de l'informatique en grille (grid computing), de l'informatique distribuée et de l'informatique parallèle, consiste en un ensemble d'ordinateurs virtuels interconnectés et utilisés dynamiquement, présentés sous forme de ressources informatiques unifiées basées sur des accords de niveau de service (SLA) établis par négociation entre le fournisseur de services, et les consommateurs, il est de plus en plus considéré comme un nouveau moyen d'utiliser les services informatiques, de stockage et de réseau à la demande de manière transparente et efficace.

Plusieurs applications scientifiques dans de nombreux domaines, tels que la bioinformatique (Oinn et al., 2004) et l'astronomie (Deelman et al., 2003), impliquent généralement de nombreuses tâches contraintes par des relations de priorité. Ces applications sont souvent exprimées sous forme de flux de travail, appelé aussi workflow, une série d'étapes de traitement connexes (tâches), où chaque étape prend des données en entrée, effectue un traitement et produit des données en sortie qui seront transmises aux étapes de traitement suivantes. Ces applications peuvent être modélisées par DAG (Direct Acyclic Graph), où les sommets du DAG représentent des tâches et les arcs représentent des dépendances de flux de données (Buyya et al., 2009).

## 1.2 Problématique

Malheureusement, les services de cloud computing ne sont pas gratuits, un contrat d'accord de niveau de service (Service Level Agreement) signé entre le fournisseur et l'utilisateur fixera le prix à payer pour les ressources louées dans un modèle de paiement à l'utilisation.

Les workflows scientifiques sont de plus en plus utilisés pour exécuter des applications scientifiques dans divers domaines, ces applications sont composés de tâches liés entre elles par une relation de précédence qui exprime la contrainte qu'une tâche prend en entrée le résultat de l'exécution d'une autre tâche. Les workflows modélisent ces applications scientifiques sous forme de graphes orientés acycliques ou DAG (Direct Acyclic Graph). Par exemple dans le domaine de l'astronomie, Montage est un workflow scientifique (Deelman et al., 2008) qui assemble des images du ciel pour former une seule image réaliste.

Un autre exemple du domaine de la bioinformatique est SciPPGx (Ocaña, de Oliveira, Dias, et al., 2012) et SciPhy (Ocaña et al., 2011) qui sont des workflows scientifiques utilisés par l'industrie pharmaceutique pour trouver de nouveaux médicaments.

Le cloud offre une plateforme intéressante pour l'exécution de ces workflows et la planification des tâches est une étape importante lors de l'exécution d'un workflow dans un environnement de cloud computing. L'algorithme de planification doit trouver une répartition efficace des tâches du workflow de l'utilisateur sur le pool de ressources du fournisseur, en minimisant le temps nécessaire à l'exécution des workflow de l'utilisateur et en optimisant la disponibilité des ressources du cloud. En outre, la planification des tâches dans un environnement de cloud computing est un problème NP-difficile en général (Brucker, 2007), et les algorithmes

d'optimisation heuristiques sont largement utilisés pour la résolution de ce genre de problèmes. Un algorithme heuristique est utilisé pour trouver une solution sous-optimale du problème (Dahal et al., 2007).

## 1.3 Objectifs

Dans ce travail, nous abordons le problème de la planification des workflows sur une infrastructure hétérogène de cloud computing, et une approche basée sur la parallélisation de l'algorithme de recuit simulé appelé PSA (Parallel Simulated Annealing) implémenté sur une architecture GPU est proposée. Sa mission est de planifier les tâches de chaque workflow destinée à utiliser les ressources de l'environnement cloud. La planification consiste à affecter chaque tâche du workflow à un processeur donné pour son exécution. Le processus doit être effectué en respectant la contrainte de précédence qui consiste à respecter l'ordre des tâches dans le workflow en fonction de leurs interdépendances, tout en minimisant le temps de calcul (makespan) qui a un impact sur la consommation d'énergie et le coût dans le cas où l'utilisateur dispose d'un budget limité.

## 1.4 Contribution

L'une des principales problématiques de recherche lorsqu'on aborde le domaine du workflow et du cloud computing, est sans doute le problème d'ordonnancement de workflows. Ce problème, bien connu, est considéré comme NP-complet, ce qui ouvre la voie à l'utilisation des métaheuristiques pour résoudre ce problème. Parmi les modèles de service du cloud computing, notre contribution dans cette thèse se situe au niveau IAAS (Infrastructure As a Service), il s'agit du développement d'un algorithme pour l'ordonnancement de

tâches de workflow, cet algorithme est basé sur la parallélisation de l'algorithme du recuit simulé avec implémentation sur une architecture GPU. Notre algorithme optimise le makespan, c'est-à-dire le temps d'exécution du workflow qui est la métrique la plus importante lorsqu'on s'intéresse à l'ordonnancement de tâches. Ce travail a fait l'objet d'une publication dans un journal international (Benhammouda & Malki, 2019)

## 1.5 Organisation de la thèse

Notre thèse est organisée comme ceci. Tout d'abord, dans le chapitre 1 nous présentons le contexte et la problématique de notre thèse ainsi que les objectifs à atteindre et notre contribution.

Dans le chapitre 2 nous définissons le concept de cloud computing et nous donnons ces caractéristiques, ses modèles de service et ses modèles de déploiement.

Dans le chapitre 3 nous définissons le concept de workflow en décrivant le système de gestion de workflows et son rôle puis on donne des exemples de workflows scientifiques.

Dans le chapitre 4 nous donnons une définition formelle du problème d'ordonnancement de tâches ainsi que sa modélisation et on introduit la notion de métaheuristique et son utilisation pour résoudre les problèmes d'optimisation.

Dans le chapitre 5, nous présentons un état de l'art des algorithmes d'ordonnancement de Workflow.

Dans le chapitre 6 nous présentons notre approche basée sur la méthode du recuit simulé et les processeurs graphiques GPU.

Les résultats expérimentaux de l'étude comparative de notre approche et de l'algorithme standard de l'industrie sont présentés dans le chapitre 7, ainsi que les métriques et le benchmark utilisés.

Nous terminons par une conclusion générale dans le chapitre 8 et présentation de quelques perspectives de recherche.

# Partie I :

# Background et état de l’art

# Chapitre 2 :
# Cloud computing

## 2.1 Introduction

Ce chapitre commence par introduire les concepts de base du cloud computing (ou l'informatique en nuages), suivie d'une rétrospective de l'évolution du cloud et d'une comparaison avec plusieurs technologies connexes. Grâce à l'analyse de l'architecture du système, des modèles de déploiement et des types de services, les caractéristiques du cloud computing sont définies sur les plans technique, fonctionnel et économique. Ensuite, les efforts actuels, tant du point de vue commercial que du point de vue de la recherche, sont présentés afin de saisir les défis et les opportunités dans ce domaine.

## 2.2 Définitions des clouds

Depuis 2007, le terme cloud est devenu l'un des mots les plus à la mode de l'industrie informatique. De nombreux chercheurs tentent de définir le cloud computing à partir de différents aspects applicatifs, mais il n'existe pas de définition consensuelle à ce sujet.

La société de conseil en informatique Gartner fournit la définition suivante: « C'est Un style d'informatique où les capacités informatiques évolutives et élastiques sont fournies en tant que service à de multiples clients externes en utilisant les technologies Internet » (Plummer et al., 2008).

Une autre définition plus objective est donnée par le U.S. National Institute of Standards and Technology NIST (Mell & Grance, 2011) :

« Le cloud computing est un modèle permettant un accès pratique et à la demande à un réseau partagé de ressources informatiques configurables (p. ex. réseaux, serveurs, stockage, applications et services) qui peuvent être rapidement provisionnées et diffusées avec un effort de gestion ou une interaction minimale avec les fournisseurs de services ».

En tant qu'universitaire, Foster donne la définition suivante du cloud computing : « C'est un paradigme de calcul distribué à grande échelle qui repose sur des économies d'échelle, dans lequel un pool de puissance de calcul, de stockage, de plates-formes et de services virtualisés, dynamiquement évolutifs et gérés est fourni à la demande aux clients externes via Internet » (Foster et al., 2008). La définition se concentre sur plusieurs caractéristiques techniques qui différencient le cloud computing des autres paradigmes de l'informatique distribuée. Par exemple, les entités informatiques sont virtualisées et fournies en tant que services, et ces services sont dynamiquement déterminés par les économies d'échelle.

## 2.3 Caractéristiques du cloud

Les cinq caractéristiques du Cloud sont définit de la façon suivante (Mell & Grance, 2011) :

- Mise en commun des ressources ou partage des ressources (pooling) : les ressources mises à la disposition des utilisateurs en utilisant une plate-forme de type Cloud sont accessibles via Internet quelque soit le périphérique utilisé (PC, Mac, tablette, Smartphone...). Ces ressources peuvent être partagées par plusieurs utilisateurs. Les fournisseurs s'appuient sur le concept de virtualisation, c'es-à-dire les ressources sont

allouées et libérées selon les besoins des clients. Ceci a pour conséquence de masquer la localisation physique des ressources pour les présenter sous forme de machines virtuelles.

– Élasticité : il est possible, grâce au Cloud, de disposer de plus de ressources instantanément pour soutenir une forte demande. Inversement, par exemple en cas de diminution du nombre d'utilisateurs d'une plate-forme, il est possible de libérer des ressources. Autrement dit, il est possible avec Cloud d'acquérir de nouvelles ressources en fonction de l'évolution des besoins métiers, et le client peut soit augmenter ou diminuer par exemple le nombre de machines et/ou leurs puissances. Généralement, cette tâche est automatisée et c'est l'opérateur de service qui ajuste les ressources en fonction des besoins. Du point de vue du client (utilisateur), les ressources du Cloud doivent paraître illimitées, accessibles en n'importe quelle quantité et de n'importe quel endroit.

– Service à la demande : le Cloud permet d'utiliser des ressources sans pour autant avoir besoin de faire une demande d'intervention des fournisseurs. En pratique, les utilisateurs accèdent via une interface Web à des services informatiques tels que la capacité supplémentaire de stockage et/ou de calcul sans intervention humaine (cette tâche étant automatisée) de la part de chaque fournisseur de services.

– Service mesuré : dans un environnement de type Cloud, le fournisseur de service est capable de mesurer de façon précise la consommation des utilisateurs des différentes ressources mises à sa disposition (par exemple CPU, stockage, bande passante, etc.), ce qui lui permet de facturer à l'usage ses clients. Parmi les services facturés, on peut citer : (i) la quantité

d'espace de stockage utilisé, (ii) le nombre de transactions, (iii) la bande passante, (iv) le nombre d'entrées et de sorties et (v) la puissance de calcul utilisée.

– Bande passante élevée et accès réseau universel : les ressources mises à disposition des utilisateurs sont accessibles via Internet de n'importe quel endroit et en utilisant des plateformes hétérogènes (PC, téléphone, mobile, PDA, ordinateur portable, etc.) grâce à la capacité actuelle des réseaux.

## 2.4 Modèles du cloud computing

Dans cette section nous présentons les modèles du cloud, à savoir les modèles de service et les modèles de déploiement.

### 2.4.1 Modèles de service du cloud computing

XaaS (X as a Service) représente la base du paradigme du cloud computing, où X représente un service tel qu'un logiciel, une plateforme, une infrastructure, un Business Process, etc. Nous présentons, dans cette section, quatre modèles de services selon le NIST (Mell & Grance, 2011) , la Figure 2-1 montre le modèle classique et les différents modèles de service de cloud :

- Logiciel entant que services SaaS (Software as a Service), où le matériel, l'hébergement, le framework d'application et le logiciel sont dématérialisés ;
- Plateforme en tant que service PaaS (Platform as a Service), où le matériel, l'hébergement et le framework d'application sont dématérialisés ;

- Infrastructure en tant que service IaaS (Infrastructure as a Service) ;
- Matériel en tant que service HaaS (Hardware as a Service), où seul le matériel (serveurs) est dématérialisé dans ces deux derniers cas.

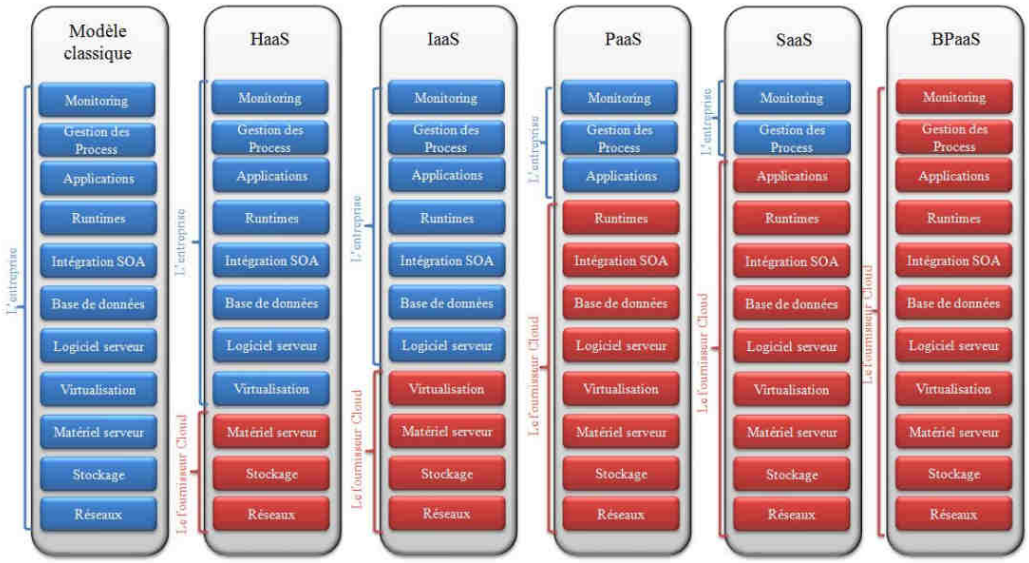



*Figure 2-1 Les services XaaS du cloud computing (Yassa, 2014)*

### 2.4.1.1 Software as a Service (SaaS)

Des applications sous forme de services en ligne déployés dans le cloud sont offertes aux utilisateurs. Cette couche est gérée par le fournisseur de SaaS de façon transparente pour les utilisateurs. Les derniers ne sont pas impliqués par le déploiement de ces applications, ils ne possèdent pas ces applications mais ils vont plutôt payer pour les utiliser. L'accès à ces applications se fait à tout moment via internet. Les applications sont utilisées soit directement via l'interface disponible, soit via des API fournies (souvent réalisées grâce aux Web Services ou à l'architecture REST (REpresentational State Transfer) (Mell & Grance, 2011).

Comme principales applications de ce modèle on peut citer les applications de gestion de la relation client (CRM : Customer Relationship Management), la vidéo conférence, les outils collaboratifs et la messagerie électronique.

#### 2.4.1.2 Business Process as a Service (BPaaS)

Le BPaaS cible plus précisément les processus métiers où les ressources sont mutualisées entre les différents utilisateurs. Il s'intègre au dessus des applications. Ce service permet aux entreprises l'automatisation et l'orchestration des flux d'actions et la supervision de ces flux (Mell & Grance, 2011).

Comme exemples de BPaaS on peut citer la gestion des voyages, le paiement en ligne, la gestion financière, etc. Les principaux fournisseurs de ce service sont : IBM, Atos, Wipro et autres.

#### 2.4.1.3 Platform as a Service (PaaS)

PaaS est un type de cloud orienté au développement d'applications. Il offre une plateforme entièrement configurée et gérée, sur laquelle l'utilisateur peut développer, tester et exécuter ses applications. Le déploiement des solutions PaaS est automatisé et évite à l'utilisateur d'avoir à acheter des logiciels ou à réaliser des installations supplémentaires. Le PaaS offre une grande flexibilité, permettant notamment de tester rapidement un prototype ou encore d'assurer un service informatique sur une période de courte durée (Mell & Grance, 2011).

Les principaux fournisseurs du PaaS sont SalesForce.com (Force.com), Google (Google App Engine), Microsoft (Windows Azure), Facebook (Facebook Platform).

#### 2.4.1.4 Hardware as a Service (HaaS)

Dans le cas du HaaS, le fournisseur de cloud loue essentiellement du matériel physique (par exemple, serveur, stockage). Les utilisateurs accèdent au HaaS via Internet pour installer et configurer les serveurs loués. Ils ont le contrôle sur le système d'exploitation et la pile logiciel/matériel. Les chercheurs scientifiques louent les services du HaaS pour leurs besoins en calculs-intensifs et/ou traitement intensifs de données applications et les configurent selon leurs besoins (Mell & Grance, 2011). Les principaux fournisseurs qui offrent du HaaS sont IBM Cloud (*IBMC*, 2020) et Amazon AWS (*AWS*, 2020).

#### 2.4.1.5 Infrastructure as a Service (IaaS)

L'IaaS permet la mise à disposition des ressources d'infrastructure telles que des capacités de calcul, des moyens de stockage, du réseau sous forme de services publics, où les ressources offertes sont virtualisées à la différence du HaaS. Les utilisateurs d'IaaS sont libérés des charges de gestion ou le contrôle du matériel et n'ont pas accès directement aux machines physiques, mais indirectement à travers les machines virtuelles (VMs). Les utilisateurs ont la plupart du temps un contrôle presque complet sur les VMs qu'ils louent: ils peuvent choisir des images de systèmes d'exploitation préconfigurées par le fournisseur, ou bien des images de machines personnalisées contenant leurs propres applications, bibliothèques et paramètres de configuration. Les utilisateurs peuvent également choisir les différents ports d'Internet à travers lesquels les machines virtuelles seront accessibles, etc. Les utilisateurs peuvent héberger et exécuter des logiciels de leurs choix, et stocker des données sur l'infrastructure en ne payant que les ressources qu'ils consomment (Mell & Grance, 2011).

La particularité de l'IaaS est de fournir un approvisionnement en ressources informatiques, qui est dynamique, flexible, extensible et qui possède de bonnes propriétés de passage à l'échelle pour répondre aux besoins spécifiques des utilisateurs. Le cloud est vu ainsi comme une source de ressources de calcul, de stockage et de réseau accessibles en un modèle de paiement à l'usage.

Il existe de nombreux fournisseurs commerciaux et open-source de l'IaaS. Parmi les plateformes commerciales, nous pouvons citer: Amazon EC2 (*Amazon EC2*, 2020), Microsoft Azure (*MicAz*, 2020) et d'autres. Pour les communautés open-sources, nous pouvons citer : Eucalyptus (*Eucal*, 2020), Nimbus (*Nimb*, 2020), OpenNebula (*OpenN*, 2020) et d'autres.

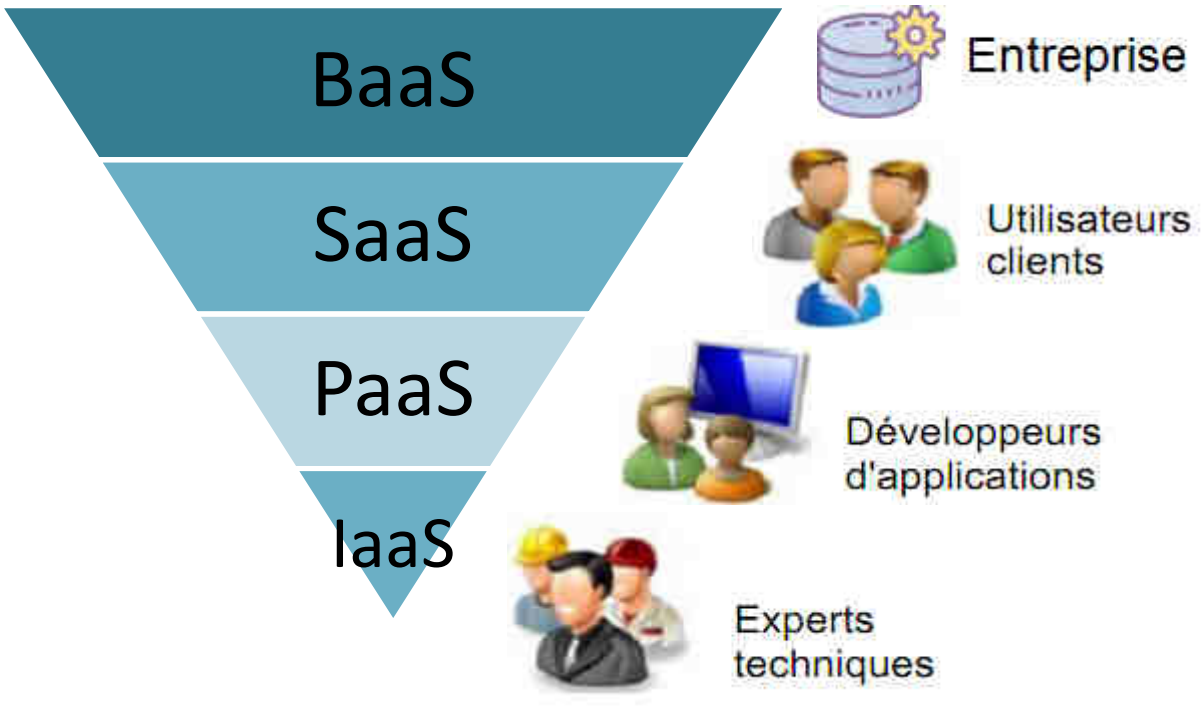



*Figure 2-2 Les services XaaS du cloud computing et leurs utilisateurs*

### 2.4.2 Modèles de déploiement

Selon la définition du cloud computing  donnée part le NIST(Mell & Grance, 2011), il existe quatre modèles de déploiement des services de cloud, à savoir : cloud privé, cloud communautaire, cloud public et cloud hybride, comme illustré dans la Figure 2-3.

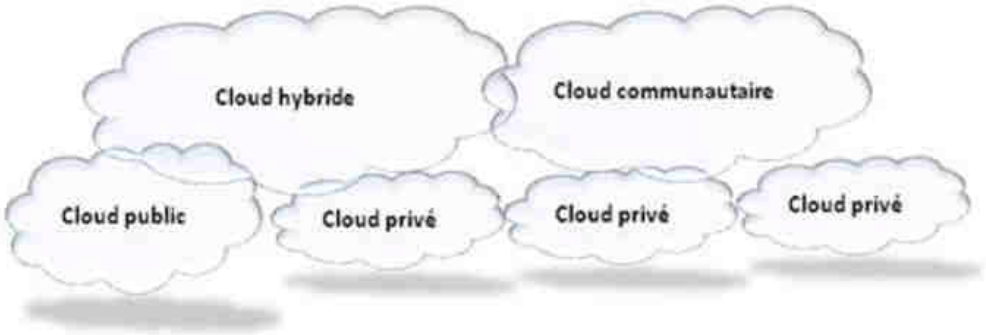



*Figure 2-3 Modèles de déploiement du cloud computing*

**Cloud privé**. L'ensemble des ressources d'un cloud privé est exclusivement mis à disposition d'une entreprise ou organisation unique. Le cloud privé peut être géré par l'entreprise elle-même (cloud privé interne) ou par une tierce partie (cloud privé externe). Les ressources d'un cloud privé se trouvent généralement dans les locaux de l'entreprise ou bien chez un fournisseur de services. Dans ce dernier cas, l'infrastructure est entièrement dédiée à l'entreprise et y est accessible via un réseau sécurisé (de type VPN). L'utilisation d'un cloud privé permet de garantir, par exemple, que les ressources matérielles allouées ne seront jamais partagées par deux clients différents (Mell & Grance, 2011).

**Cloud communautaires**. L'infrastructure d'un cloud communautaire est partagée par plusieurs organisations indépendantes ayant des intérêts communs.

L'infrastructure peut être gérée par les organisations membres ou par un tiers. L'infrastructure peut être située, soit au sein des dites organisations, soit chez un fournisseur de services (Mell & Grance, 2011).

**Cloud public**. L'infrastructure d'un cloud public est accessible à un large public et appartient à un fournisseur de services. Ce dernier facture les utilisateurs selon la consommation et garantit la disponibilité des services via des contrats SLA (Mell & Grance, 2011).

**Cloud hybride**. L'infrastructure d'un cloud hybride est une composition de plusieurs clouds (privé, communautaire ou public). Les différents clouds composant l'infrastructure restent des entités uniques, mais sont reliés par une technologie standard ou propriétaire permettant ainsi la portabilité des données ou des applications déployées sur les différents clouds (Mell & Grance, 2011).

## 2.5 Conclusion

Nous avons présenté dans ce chapitre le paradigme du Cloud computing. Un paradigme de plus en plus adopté par de nombreuses entreprises pour exécuter leurs applications.

Les services PaaS et SaaS peuvent être aussi utilisés pour les workflows. Le PaaS peut exposer un environnement de développement et d'exécution de haut niveau permettant de construire et de déployer les workflows sur l'infrastructure cloud.

Les fournisseurs SaaS offrent aux utilisateurs finaux des solutions logicielles standardisées qui pourraient être intégrées dans leurs workflows existants.

Cependant, le Cloud Computing engendre plusieurs problèmes tels que les problèmes de sécurité et de confidentialité des données. Un autre problème et non le moindre et qui constitue la principale motivation des travaux présentés dans cette thèse, c'est celui du problème d'ordonnancement des tâches des workflow en utilisant des ressources déployées dans le Cloud. Nous continuons dans le chapitre suivant par présenter avec plus de détails le concept de Workflow

# Chapitre 3 : Workflow

## 3.1 Introduction

Le concept de workflow trouve ses racines dans les entreprises commerciales en tant qu'outil de modélisation des processus métier. Ces workflows métier visent à automatiser et optimiser les processus d'une organisation, vus comme une séquence ordonnée d'activités, et constituent un domaine de recherche mature dirigé par la Workflow Management Coalition (WfMC), fondée en 1993.

Cette notion de workflow s'est étendue à la communauté scientifique où les flux de travail scientifiques sont utilisés pour soutenir des processus scientifiques complexes à grande échelle. Ils sont conçus pour mener des expériences et prouver des hypothèses scientifiques en gérant, analysant, simulant et virtualisant des données scientifiques. En science, il est courant que ces applications soient composées d'un ensemble de tâches de calcul et d'un ensemble de données ou de dépendances de contrôle entre les tâches.

Dans ce chapitre, dans un premier temps nous présentons le concept de workflow scientifique, ainsi que la notion de système de gestion de workflow (Workflow Management System). Ensuite, nous passons en revue les fonctionnalités communes des WMS conçues pour les environnements cloud ainsi qu'une architecture générale et ses composants pour soutenir l'exécution des workflows.

## 3.2 Définition de workflow

Une définition du concept de workflow générale et indépendante des domaines spécifiques a été donné par la WfMC : " Un workflow est l'automatisation d'un processus métier, en tout ou en partie, au cours de laquelle des documents, des informations ou des tâches sont passées d'un participant à un autre pour l'action, selon un ensemble de règles procédurales" (WfMC, 1999).

Dans le contexte des workflows scientifiques, la définition suivante est largement acceptée : Un workflow est composé d'un ensemble de tâches (traitements) organisées selon un ordre bien déterminé, afin de réaliser un traitement global, complexe et pertinent sur un ensemble de données sources volumineuses (Lin et al., 2009).

Les workflows scientifiques de grande taille nécessite souvent un calcul intensif, il est donc souhaitable de répartir les tâches entre plusieurs ressources, afin d'optimiser les temps d'exécution. En tant que tel, les workflows impliquent souvent des calculs répartis sur des clusters, des grilles, et d'autres infrastructures informatiques. Récemment, les clouds computing sont évalués comme une plateforme d'exécution de workflows (Hoffa et al., 2008). L'exécution d'un workflow est gérée par un Système de Gestion de Workflow (SGWf).

## 3.3 Système de gestion de workflows

Les workflows scientifiques sont souvent des applications gourmandes en données et en ressources et nécessitent une plate-forme distribuée pour obtenir des résultats significatifs dans un délai raisonnable. Leur déploiement est géré par un système de gestion de workflow (Workflow Management Systems ) ou WMS qui

est chargé d'orchestrer en toute transparence l'exécution des tâches de workflow dans un ensemble de ressources de calcul réparties tout en garantissant la préservation des dépendances. Un aperçu de haut niveau de ce processus est illustré à la Figure 3-1. En général, les WMS fournissent des fonctionnalités essentielles pour permettre l'exécution de workflows tels que la gestion et la provenance des données, la planification des tâches, l'approvisionnement des ressources et la tolérance aux pannes, entre autres.

Un WMS représente un système qui définit, implémente et gère l'exécution de workflows à l'aide d'un environnement logiciel fonctionnant avec un ou plusieurs moteurs de workflows et capable d'interpréter la définition d'un processus, de gérer la coordination des participants et d'invoquer des applications externes. L'objectif de la gestion de workflow est de s'assurer que des activités appropriées sont exécutées par les bons services, au bon moment (Rodriguez & Buyya, 2017).

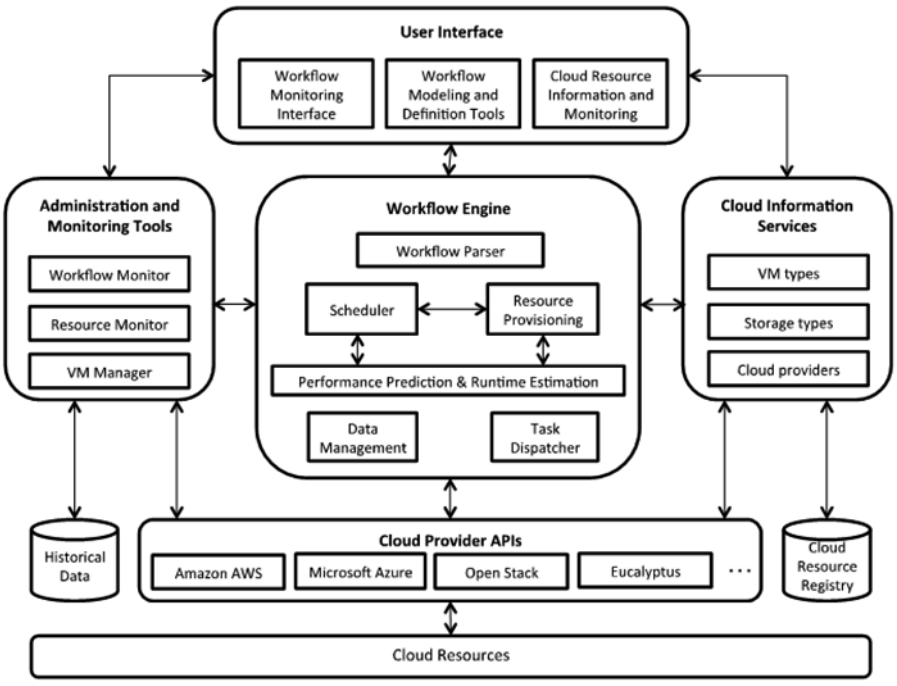



*Figure 3-1 Architecture de référence d'un système de gestion de workflows dans le cloud (Rodriguez & Buyya, 2017).*

Nous commencerons par introduire un modèle architectural général pour les WMS. En général, un WMS permet la création, la surveillance et l'exécution de workflows scientifiques et a la capacité de gérer de manière transparente les tâches et les données en cachant les détails d'orchestration et d'intégration parmi les ressources distribuées (Yu & Buyya, 2004). Une architecture de référence est représentée sur la Figure 3-1. Les composants représentés sont communs à la plupart des implémentations WMS cloud, cependant, tous ne sont pas tenus d'avoir un système entièrement fonctionnel.

**Interface utilisateur (User interface)**. L'interface utilisateur permet aux utilisateurs de créer, modifier, soumettre et surveiller leurs applications.

**Moteur de workflow (Workflow engine)**. Le moteur de workflow est le cœur du système et est responsable de la gestion de l'exécution réelle du workflow. Le module d'analyseur dans le moteur interprète un workflow représenté dans un langage de haut niveau tel que XML et crée la représentation de workflow interne correspondante telle que des objets de tâche et de données. L'ordonnanceur et les modules de provisionnement des ressources travaillent ensemble pour planifier l'exécution de workflow. Le module de provisioning des ressources est responsable de la sélection et du provisioning des ressources cloud et le composant de planification applique des politiques spécifiques qui mappent les tâches aux ressources disponibles, les deux processus sont basés sur les exigences de QoS et les objectifs de planification. Le module de prédiction de performances et d'estimation d'exécution utilise des données historiques, la provenance des données ou des modèles de prédiction de séries chronologiques, entre autres méthodes, pour estimer les performances des ressources cloud et le temps nécessaire aux tâches à exécuter sur différentes machines virtuelles. Ces

données sont utilisées par les modules d'approvisionnement et de planification des ressources pour prendre des décisions précises et efficaces concernant l'allocation des tâches. Le composant de gestion des données du moteur de workflow gère le mouvement, le placement et le stockage des données comme requis pour l'exécution du workflow. Enfin, le répartiteur de tâches a la responsabilité d'interagir avec les APis cloud pour répartir les tâches prêtes à être exécutées sur les machines virtuelles disponibles.

**Outils d'administration et de suivi (Administration and Monitoring Tools)**. Les outils d'administration et de surveillance de l'architecture WMS comprennent des modules qui permettent la surveillance dynamique et continue des tâches de workflow et des performances des ressources ainsi que la gestion des ressources louées, telles que les machines virtuelles. Les données collectées par ces outils peuvent être utilisées par des mécanismes de tolérance aux pannes ou peuvent être stockées dans une base de données historique et utilisées par des méthodes de prédiction de performances, par exemple.

**Services d'information cloud (Cloud information services).** Un autre composant de l'architecture est le service d'information cloud. Ce composant fournit au moteur de workflow des informations sur les différents fournisseurs de cloud, les ressources qu'ils offrent, notamment leurs caractéristiques et leurs prix, leur emplacement et toute autre information requise par le moteur pour prendre les décisions de sélection et de mapping des ressources.

**Fournisseur de cloud Apis (Cloud provider APis)**. Ces APis permettent l'intégration d'applications avec des services cloud. Pour le problème de planification décrit dans ce chapitre, ils permettent le provisionnement et le déprovisionnement à la demande des machines virtuelles, la surveillance de

l'utilisation des ressources au sein d'une machine virtuelle spécifique, l'accès aux services de stockage pour enregistrer et récupérer des données, le transfert de données dans ou hors de leurs installations, et la configuration des paramètres de sécurité et de travail réseau, entre autres. La majorité des IaaS APis sont exposées en tant que services REST (Representational State Transfer) et SOAP (Simple Object Access Protocol), mais des protocoles tels que XML-RPC sont également utilisés. Par exemple, CloudSigrna, Rackspace, Windows Azure et Amazon EC2 proposent tous des APis basés sur REST. Au lieu de fournir des services pour une plate-forme spécifique, d'autres solutions telles que Apache JClouds visent à créer un environnement cloud multiplateforme en fournissant une API pour accéder aux services de différents fournisseurs de cloud de manière transparente. Les interfaces multiplateformes ont l'avantage de permettre aux applications d'accéder aux services de plusieurs fournisseurs sans avoir à réécrire de code, mais peuvent avoir moins de fonctionnalités ou d'autres limitations par rapport aux solutions spécifiques au fournisseur.

Un des principaux modules de tous les systèmes de gestion de workflow est l'ordonnancement. Ce dernier est un processus qui "mappe" et gère l'exécution de tâches interdépendantes sur des ressources distribuées. Il alloue des ressources appropriées pour les tâches de workflow de sorte que l'exécution puisse être achevée, tout en satisfaisant les objectifs et contraintes imposés par l'utilisateur. Un ordonnancement adéquat peut avoir un impact significatif sur la performance du système. En général, le problème de mapping de tâches sur les services distribués appartient à une classe de problèmes connus comme NP-complets (Brucker, 2007).

Le cloud computing, offre plusieurs avantages pour le déploiement de ces applications. En particulier, les clouds de type Infrastructure-as-a-Service (Iaas) offrent aux WMS une infrastructure facilement accessible, flexible et évolutive en louant des ressources de calcul virtualisées ou des machines virtuelles (VM). Cela permet de regrouper et de déployer facilement les workflows et, plus important encore, de permettre aux WMS d'accéder à un pool pratiquement infini de machines virtuelles hétérogènes qui peuvent être acquises et libérées de manière élastique et sont facturées au paiement à l'utilisation.

De cette façon, les WMS peuvent utiliser les ressources du cloud de manière opportune en fonction du nombre et du type de tâches qui doivent être traitées à un moment donné. Il s'agit d'une fonctionnalité pratique car il est courant que le parallélisme des tâches des workflows scientifiques change considérablement tout au long de leur exécution. Le pool de ressources peut être agrandi et ajusté au fur et à mesure de l'exécution du workflow. Cela facilite la satisfaction des exigences de qualité de service (QoS) en permettant au WMS d'affiner les performances tout en garantissant une utilisation efficace des ressources disponibles.

## 3.4 Exemples de workflows scientifiques

Les workflows scientifiques ont été utilisés dans divers domaines scientifiques. Dans le domaine de l'astronomie, Montage est une application informatique gourmande en données qui peut être modélisée comme un workflow scientifique initialement défini pour le Pegasus WMS. Cette application est le résultat d'un projet d'observation virtuelle qui assemble des carreaux d'images du ciel

provenant de divers observatoires en une seule image photoréaliste (Deelman et al., 2008).

Montage est capable de gérer un large éventail de données d'images astronomiques, y compris l'observatoire Two Micron All Sky Survey, (2MASS), l'observatoire du ciel Digitized Palomar Observatory Sky Survey (DPOSS) et l'observatoire Sloan Digital Sky Survey (SDSS) (Jacob et al., 2010). Chaque image possède d'énormes quantités de données et couvre une partie correspondante du ciel dans les longueurs d'onde visibles ou dans le proche infrarouge. 2MASS a environ 10 téraoctets, DPOSS a environ 3 téraoctets et SDSS contient environ 7,4 téraoctets. Toutes les données peuvent être téléchargées à partir d'un serveur correspondant.

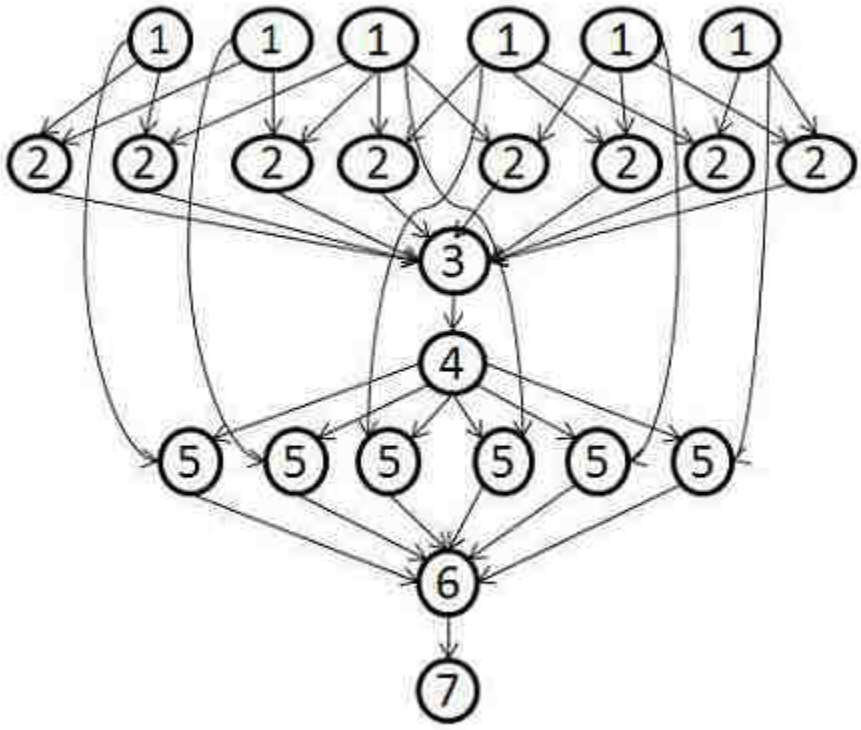



*Figure 3-2 La structure d'un petit workflow Montage (Jacob et al., 2010).*

La structure d'un petit workflow montage est illustrée à la Figure 3-2. Le nombre à l'intérieur d'un nœud représente le niveau d'activité dans le workflow.

Le premier niveau concerne les activités orphelines (sans activités parentales). Le niveau de toute autre activité est le résultat du niveau maximum de l'un de ses parents plus un. Au même niveau, toutes les activités sont indépendantes les unes des autres tandis que chaque niveau se compose du même programme travaillant sur différentes données d'entrée (Jacob et al., 2010).

Les activités du premier niveau exploitent un programme mProject qui projette une seule image à l'échelle définie dans un fichier de modèle d'en-tête pseudo-FITS (un fichier ASCII avec les lignes d'en-tête de l'image de sortie, mais non complété à 80 caractères et avec des nouvelles lignes à la fin de chaque ligne).

Les activités de deuxième niveau utilisent un programme mDiffFit pour créer un tableau des paramètres de différence d'image à image. L'activité de troisième niveau tire parti d'un programme mFitplane pour adapter les images générées par les activités antérieures à une image.

L'activité de quatrième niveau utilise un programme mBgModel pour déterminer de manière interactive un ensemble de corrections à appliquer à chaque image pour obtenir un «meilleur» ajustement global selon le tableau des paramètres de différence d'image à image. Les activités de cinquième niveau suppriment un arrière-plan d'une seule image via un programme mBackground.

L'activité de sixième niveau utilise un programme mImgtbl pour extraire les informations d'en-tête FITS (informations sur un ou plusieurs systèmes de coordonnées scientifiques superposées sur l'image elle-même) à partir d'un ensemble de fichiers et pour créer une table de métadonnées d'image ASCII.

Enfin, la dernière activité regroupe les images projetées en utilisant le modèle d'en-tête FITS uniforme et les informations de la même table de

métadonnées générée par l'activité précédente. Cette activité applique un programme mAdd.

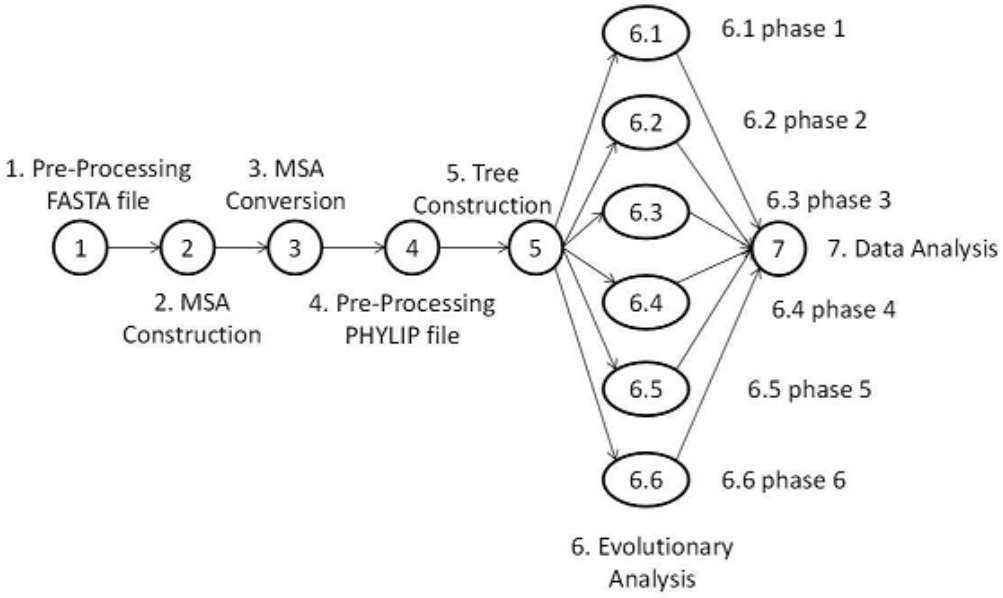



*Figure 3-3 Le Workflow SciEvol.*

Dans le domaine de la bioinformatique, SciEvol (Ocaña, de Oliveira, Horta, et al., 2012) est un workflow pour la reconstruction de l'évolution moléculaire qui vise à inférer des relations évolutives sur des données génomiques. Pour être exécuté dans le WMS Chiron, SciEvol se compose de 12 activités comme indiqué sur la Figure 3-3.

La première activité (prétraitement d'un fichier FASTA) est un script Python pour formater le fichier d'entrée multi-fasta en supprimant les codons d'arrêt dans les séquences. Le fichier FASTA est un format de présentation textuel de séquences nucléotidiques ou de séquences peptidiques.

La deuxième activité (construction MSA) construit un alignement de séquences multiples (MSA) à l'aide d'un programme MAFFT (ou d'autres programmes MSA). Un programme MAFFT est généralement destiné à générer

l'alignement de trois séquences biologiques ou plus (protéines ou acides nucléiques) de longueur similaire. Cette activité reçoit un fichier multi-fasta préformaté en entrée et produit le MSA au format FASTA en sortie. La troisième activité (conversion MSA) exécute ReadSeq pour convertir le MSA au format FASTA en un autre format appelé PHYLIP, qui est utilisé dans l'activité de construction d'arbres phylogénétiques. La quatrième activité (prétraitement du fichier PHYLIP) formate le fichier d'entrée au format PHYLIP (référencé comme «phylip-file-un») selon la définition du format et génère un deuxième fichier au format PHYLIP (référencé comme «phylip-file-deux»)). La cinquième activité (construction d'arbre) reçoit le «phylip-file-un» en entrée et produit un arbre phylogénétique au format Newick en sortie. Les sixièmes activités (analyse évolutive de 6.1 à 6.6) analysent l'arbre phylogénétique par des paramètres correspondants et génèrent un ensemble de fichiers contenant des informations évolutives en sortie. La dernière activité (analyse des données) traite automatiquement les fichiers de sortie obtenus des activités précédentes.

Il existe de nombreux autres workflows gourmands en données en bioinformatique. Par exemple, SciPhylomics (De Oliveira et al., 2013) est conçu pour produire des arbres phylogénomiques basés sur un ensemble d'entrée de séquences protéiques de génomes pour inférer des relations évolutives entre organismes vivants.

SciPPGx (Ocaña, de Oliveira, Dias, et al., 2012) est un workflow d'analyse pharmacophylogénomique et de données intensives pour fournir un support d'inférence approfondi pour les hypothèses pharmacophylogénomiques.

SciPhy (Ocaña et al., 2011) est utilisé pour construire des arbres phylogénétiques à partir d'un ensemble d'enzymes cibles médicamenteuses présentes dans les génomes des protozoaires. Tous ces workflows bioinformatiques ont été exécutés à l'aide du WMS SciCumulus (De Oliveira et al., 2010).

## 3.5 Conclusion

Dans ce chapitre une définition générale du concept de workflow a été donnée ainsi que le concept de workflow scientifique. Le système de gestion de workflows et ses principaux modules ont été présentés. Des exemples d'application de workflow scientifiques réels ont été présentés.

Dans le chapitre suivant, nous allons présenter de façon formelle le problème d'ordonnancement de tâches de workflow, qui représente le sujet de notre contribution dans cette thèse.

# Chapitre 4 : Ordonnancement de tâches de Workflow

## 4.1 Introduction

Le problème d'ordonnancement des applications composées de tâches indépendantes est le plus étudié comparé aux deux autres types d'applications. Les algorithmes d'ordonnancement des tâches indépendantes visent, généralement, à minimiser le temps global d'exécution ou le makespan, dans le cas où les tâches arrivent au fur et à mesure et pas toutes en même temps. Le problème étant très difficile, des algorithmes approchés sont proposés.

Même si ces algorithmes ont l'avantage de garantir la qualité des solutions obtenues au pire cas par rapport à la solution optimale, ils restent très peu exploitable concrètement car ils sont très couteux en temps d'exécution et ne tiennent pas en compte du temps de communication entre les tâches.

## 4.2 Représentation des applications parallèles

On peut distinguer deux catégories principales d'applications parallèles selon les tâches qui les composent et les interactions entre elles, à savoir (Feitelson & Rudolph, 1996) :

1. Le premier type d'applications parallèles où les tâches peuvent s'exécuter indépendamment les unes des autres. Parmi ce type d'applications, il existe celles qui sont constituées de tâches parallèles et celles qui sont

définies par des tâches séquentielles. Il existe un troisième type d'applications appelées applications paramétriques (parameter sweep application). Dans ce dernier cas, une même application est exécutée plusieurs fois avec différents paramètres.

2. Le deuxième type d'applications parallèles où les tâches constituant l'application en question ne peuvent être exécutées de manière indépendante et sont liées par des contraintes de précédences représentant des contraintes temporelles ou d'échange de données. Pour ce type d'applications, il existe deux catégories principales :

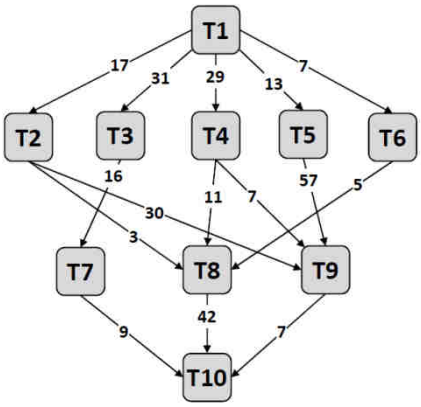
   (a) La première catégorie se réfère aux applications définies par des tâches qui interagissent en s'échangeant des données durant leurs exécutions. Ces applications peuvent être modélisées en utilisant des graphes non orientés appelés Task interaction graphs (TIGs). Les sommets de ces graphes représentent les tâches et les arrêtes représentent les communications entre les ressources.

   (b) La deuxième catégorie où les tâches sont liées par des contraintes de précédence. Autrement dit, une tâche de ces applications ne peut être exécutée que si toutes les tâches qui la précèdent ont été exécutées. Ces applications sont généralement représentées par des graphes orientés sans cycle (Directed Acyclic Graphs (DAGs)). Les sommets de ces graphes représentent les tâches (parallèles ou séquentielles) de l'application à exécuter et les arcs définissent les relations de précédence entre ces tâches (dépendances temporelles ou de données). Dans le cas où toutes les tâches sont séquentielles, on parle d'application exploitant purement le paradigme de

parallélisme de tâches. Dans le cas contraire (i.e. l'application est composée de tâches parallèles et séquentielles), on utilise des Temporal Task Interaction Graphs (TIGs) pour les représenter.

## 4.3 Modélisation du problème

L'ordonnancement des tâches est l'une des étapes les plus importantes lors de l'utilisation de fonctionnalités d'un environnement cloud computing pour exécuter un workflow. Un workflow peut être représenté par un graphe acyclique dirigé (DAG), G = (V, E), comme illustré à la *Figure 4-1, où V est l'ensemble des nœuds vi, un nœud vi représente une tâche du workflow contenant un nombre connu d'instructions qui doivent être exécutées sur le même processeur. E représente l'ensemble des arêtes de communication entre les tâches; chaque arête e(i, j) représente la contrainte de dépendance entre tâches, ce qui signifie que la tâche ti doit être exécutée avant que la tâche tj puisse être lancée. Une matrice de coût de calcul W complémente le DAG, cette matrice est un v* $\times$ *p, où v est le nombre de tâches et p le nombre de processeurs. wij donne le temps estimé pour exécuter la tâche vi sur la machine pj. Chaque arête e(i, j)* $\in$ *E est associée à un poids positif cij représentant le coût de communication entre les tâches vi et vj (Topcuoglu et al., 2002).*




| Tâche | P1 | P2 | P3 |
|---|---|---|---|
| **T1** | 22 | 21 | 36 |
| **T2** | 22 | 18 | 18 |
| **T3** | 32 | 27 | 43 |
| **T4** | 7 | 10 | 4 |
| **T5** | 29 | 27 | 35 |
| **T6** | 26 | 17 | 24 |
| **T7** | 14 | 25 | 30 |
| **T8** | 29 | 23 | 36 |
| **T9** | 15 | 21 | 8 |
| **T10** | 13 | 16 | 33 |

*Figure 4-1 Le DAG d'un Workflow et la matrice de temps d'exécution des tâches sur trois processeurs.*

Dans un graphe de tâches donné, une tâche sans parent s'appelle une tâche d'entrée et une tâche sans enfant s'appelle une tâche de sortie. Un graphe de tâches peut comporter plusieurs tâches d'entrée et plusieurs tâches de sortie.

Lorsque deux tâches ti et tj sont planifiées sur le même processeur, cij devient zéro puisque nous supposons que le coût de la communication intra-processeur est négligeable par rapport au coût de la communication inter-processeur.

L'ordonnancement des tâches d'un workflow à exécuter sur les processeurs d'un système de cloud computing est formellement appelée "a job shop scheduling problem" (JSSP). Le JSSP est l'un des problèmes les plus anciens et les plus difficiles à traiter. Il est reconnu comme l'un des problèmes NP-hard les plus difficiles (Blazewicz et al., 2007) (Sharma & Rashid, 2020).

Un n×m JSSP classique peut être décrit comme suit : Soit un ensemble fini de n travaux $J = \{J_i \mid J_i \in N\ (n)\ \}$ à usiner sur un ensemble fini de m machines $M = \{M_i \mid M_i \in N\ (n)\}$, où N (n) est un sous-ensemble de l'ensemble des nombres naturels, composé des nombres naturels de 1 à n.

Dans un JSSP, chaque tâche doit être traitée sur chaque machine et comporte des opérations prédéterminées qui sont attribuées dans un ordre donné (Blazewicz et al., 1996).

Le problème consiste à déterminer les séquences d'opérations sur chaque machine afin de satisfaire simultanément un ou plusieurs objectifs.

En règle générale, l'objectif le plus courant est l'achèvement de tous les travaux dans les délais les plus brefs. Dans cette thèse, nous supposons que le nombre de tâches est supérieur au nombre de processeurs. Les tâches sont indépendantes. Le temps d'exécution sur chaque processeur est connu. Les tâches ne sont pas autorisées à migrer entre les processeurs. L'objectif est de minimiser le temps d'exécution appelé Makespan qui est égal au délai d'exécution de la dernière tâche.

## 4.4 Métaheuristiques pour l'optimisation mono-objectif difficile

L'un des principes les plus fondamentaux de notre monde, qui occupe aussi une place importante dans le monde informatique, industriel, etc., est la recherche d'un état optimal.

En effet, de nombreux problèmes scientifiques, sociaux, économiques et techniques ont des paramètres qui peuvent être ajustés pour produire un résultat plus souhaitable. Ceux ci peuvent être considérés comme des problèmes d'optimisation et leur résolution est un sujet central en recherche opérationnelle. Des techniques ont été conçues pour résoudre ces problèmes, notamment les problèmes « difficiles » (par exemple ceux qui présentent de nombreux extrema

locaux pauvres), en déterminant des solutions qui ne sont pas rigoureusement optimales, mais qui s'en approchent. Ces méthodes, appelées heuristiques et métaheuristiques, se basent généralement sur des phénomènes physiques, biologiques, socio-psychologiques, et peuvent faire appel au hasard.

### 4.4.1 Problème d'optimisation

En général, un problème d'optimisation est défini par un ensemble de variables, une fonction objective f et un ensemble de contraintes que les variables doivent satisfaire. L'espace de recherche E est formé par l'ensemble des solutions possibles du problème, où chaque dimension correspond à une variable. L'espace de recherche E est fini puisque le domaine de définition de chaque variable est précisé exactement. Suivant le problème posé, on cherche à minimiser ou maximiser la fonction objective f. Un problème d'optimisation peut être mono-objectif ou multi-objectif (i.e. plusieurs fonctions objectifs doivent être optimisées) et avec ou sans contraintes.

Il existe de nombreuses méthodes déterministes (ou exactes) permettant de résoudre certains types de problèmes d'optimisation et d'obtenir la solution optimale du problème, en un temps raisonnable. Ces méthodes nécessitent que la fonction objective présente un certain nombre de caractéristiques telles que la convexité, la continuité ou la dérivabilité.

Nous pouvons citer, parmi les méthodes les plus connues, les méthodes de programmation linéaire (Schrijver, 1998), quadratique, la méthode de Newton (Nocedal & Wright, 2006), la méthode du simplex (Nelder & Mead, 1965) ou encore la méthode du gradient (Avriel, 1976).

### 4.4.2 Optimisation difficile

Les méthodes de résolution exacte ne sont pas adaptées à toutes les problématiques, et donc certains problèmes sont trop complexes à résoudre par ces méthodes.

Dans ce cas, le problème d'optimisation est dit difficile, car aucune méthode exacte n'est capable de le résoudre en un temps raisonnable. Il est alors nécessaire d'avoir recours à des heuristiques de résolution dites méthodes approchées, qui fournissent un résultat sans garantie de l'optimalité.

L'optimisation difficile peut se diviser en deux types de problèmes : les problèmes à variables discrètes et les problèmes à variables continues :

- Un problème d'optimisation à variables discrètes consiste à trouver, dans un ensemble discret, une solution réalisable. Le problème majeur réside ici dans le fait que le nombre de solutions réalisables est généralement très élevé, donc il est très difficile de trouver la meilleure solution dans un temps « raisonnable ». Les problèmes à variables discrètes rassemblent les problèmes de type NP-complets, tels que le problème du voyageur de commerce. L'utilisation d'algorithmes d'optimisation stochastiques, tels que les métaheuristiques, permettant de trouver une solution approchée en un temps raisonnable est donc courante.
- Dans le deuxième type, les variables du problème d'optimisation sont continues. C'est le cas par exemple des problèmes d'identification, où l'on cherche à minimiser l'erreur entre le modèle d'un système et des observations expérimentales. Ce type de problèmes est moins formalisé que le précédent, mais un certain nombre de difficultés sont bien connues,

comme l'existence de nombreuses variables présentant des corrélations non identifiées, la présence de bruit.

.

### 4.4.3 Algorithmes d'optimisation approchée

#### 4.4.3.1 Heuristiques

L'utilisation de méthodes exactes n'est pas toujours possible pour un problème donné à cause d'un certain nombre de contraintes, telles que le temps de calcul souvent important ou bien la difficulté, voire l'impossibilité dans certains cas, d'une définition séparée du problème.

Pour faire face à ces contraintes, nous utilisons des méthodes approchées, appelées heuristiques. Le terme heuristique dérive du grec ancien heuriskêin et signifie « trouver ».

Il qualifie tout ce qui sert à la découverte et à l'exploitation. Il est à souligner ici qu'une méthode heuristique peut être déterministe ou stochastique.

Une heuristique est un algorithme qui fournit rapidement (en un temps polynomial) une solution approchée et réalisable, pas nécessairement optimale, pour un problème d'optimisation difficile. Cette méthode approximative est le contraire d'un algorithme exact qui donne une solution optimale pour un problème donné.

Il y a une multitude d'heuristiques qui ont déjà été proposées dans la littérature. Nous pouvons citer des heuristiques très simples telles que les algorithmes gloutons ou les approches par amélioration itérative (Cormen et al., 1987).

Le principe des méthodes gloutonnes est de faire une succession de choix optimaux localement, jusqu'à ce que l'on ne puisse plus améliorer la solution, et ce, sans retour en arrière possible. Le fonctionnement d'une heuristique gloutonne est similaire à celui d'un algorithme glouton exact. La différence réside dans le fait que nous n'imposons plus que la solution obtenue soit optimale, nous obtenons donc un algorithme d'approximation.

#### 4.4.3.2 Métaheuristiques

Des heuristiques plus poussées, adaptables à un grand nombre de problèmes différents, sans changements majeurs dans l'algorithme, ont été mises au point et ont donné naissance à une nouvelle famille d'algorithmes d'optimisation stochastiques : les métaheuristiques. Le terme métaheuristique a été inventé par Fred Glover en 1986, lors de la conception de la recherche tabou (Glover, 1986).

Les métaheuristiques forment une famille d'algorithmes d'optimisation visant à résoudre des problèmes d'optimisation difficile, pour lesquels nous ne connaissons pas de méthodes classiques plus efficaces. Elles sont généralement utilisées comme des méthodes génériques pouvant optimiser une large gamme de problèmes différents, d'où le qualificatif méta. Leur capacité à optimiser un problème à partir d'un nombre minimal d'informations est contrebalancée par le fait qu'elles n'offrent aucune garantie quant à l'optimalité de la meilleure solution trouvée. Cependant, du point de vue de la recherche opérationnelle, ce constat n'est pas forcément un désavantage, puisque l'on préfère toujours une approximation de l'optimum global trouvée rapidement à une valeur exacte trouvée dans un temps assez long.

Il existe un grand nombre de métaheuristiques différentes, allant de la simple recherche locale à des algorithmes complexes de recherche globale. La plupart des métaheuristiques utilisent des processus aléatoires comme moyens de récolter de l'information et de faire face à des problèmes comme l'explosion combinatoire. Les métaheuristiques peuvent être considérées comme des algorithmes stochastiques itératifs, où elles manipulent une ou plusieurs solutions à la recherche de l'optimum. Les itérations successives doivent permettre de passer d'une solution de mauvaise qualité à la solution optimale. L'algorithme s'arrête après avoir atteint un critère d'arrêt, consistant généralement en l'atteinte du temps d'exécution imparti ou en une précision demandée. Ces méthodes tirent leur intérêt de leur capacité à éviter les optima locaux, soit en acceptant des dégradations de la fonction objectif au cours du traitement, soit en utilisant une population de points comme méthode de recherche.

Les métaheuristiques sont souvent inspirées de processus naturels qui relèvent de la physique (l'algorithme du recuit simulé), de la biologie de l'évolution (les algorithmes génétiques) ou encore de l'éthologie (les algorithmes de colonies de fourmis ou l'optimisation par essaim particulaire).

Les métaheuristiques se caractérisant par leur capacité à résoudre des problèmes très divers, elles se prêtent naturellement à des extensions. Pour illustrer celles-ci, nous pouvons citer :

– Les métaheuristiques pour l'optimisation multi-objectif (Collette & Siarry, 2002) : où il faut optimiser plusieurs objectifs contradictoires. Le but ne consiste pas ici à trouver un optimum global, mais à trouver un ensemble d'optima, qui forment une surface de compromis pour les différents objectifs du problème ;

– Les métaheuristiques pour l'optimisation multimodale (Goldberg et al., 1987) : où l'on ne cherche plus l'optimum global, mais l'ensemble des meilleurs optima globaux et/ou locaux ;

– Les métaheuristiques pour l'optimisation de problèmes bruités : où il existe une incertitude sur le calcul de la fonction objectif, dont il faut tenir compte dans la recherche de l'optimum ;

– Les métaheuristiques pour l'optimisation dynamique (Branke, 2002) : où la fonction objective varie dans le temps, ce qui nécessite d'approcher l'optimum à chaque pas de temps ;

– Les métaheuristiques parallèles (Alba, 2005) : où l'on cherche à accélérer le calcul, en répartissant la charge de calcul sur des unités fonctionnant de concert. Le problème revient alors à adapter les métaheuristiques pour qu'elles soient distribuées.

En général, l'utilisateur demande des méthodes efficaces et rapides permettant d'atteindre un optimum avec une précision acceptable dans un temps raisonnable, mais il a besoin aussi des méthodes simples à utiliser. Un des enjeux des métaheuristiques est donc de faciliter le choix d'une méthode et de simplifier son réglage pour l'adapter au mieux à un problème posé.

## 4.5 Conclusion

Dans ce chapitre, nous avons présenté le problème de l'ordonnancement des applications composées de tâches. Ensuite, nous avons présenté les différents types d'application parallèles, en situant les workflows par rapport à celles-ci. La modélisation du problème ordonnancement des tâches d'un workflow a été ensuite

présentée de façon formelle. Puis le problème d'optimisation difficile a été explicité, ainsi que les notions d'heuristique et de métaheuristique.

Dans le chapitre suivant nous allons dresser un état de l'art de l'ordonnancement de tâches de workflows.

# Chapitre 5 :
# Etat de l'art de l'ordonnancement de tâches de workflow

## 5.1 Introduction

L'ordonnancement de workflow est un problème intéressant qui a fait l'objet de nombreuses recherches. En effet, c'est un problème NP-complet bien connu (Lawler et al., 1982). Par conséquent, de nombreux algorithmes d'optimisation heuristiques ont été proposés pour résoudre ce problème. Dans ce chapitre, nous présentons un bref aperçu des algorithmes d'ordonnancement des tâches. Il existe deux type d'ordonnancement (Cavanaugh et al., 2000)

## 5.2 L'ordonnancement dynamique

L'ordonnancement dynamique produit des plan d'ordonnancement qui distribuent et allouent des tâches exécutables aux nœuds de calcul pendant l'exécution du workflow. Ce type de planification est approprié pour les workflows scientifiques, dans lesquels la charge de travail des tâches est difficile à estimer, ou pour les environnements où les capacités des ordinateurs varient beaucoup pendant l'exécution. Il est à noter que la planification dynamique nécessite du temps pour générer dynamiquement des plans d'ordonnancement pendant l'exécution. Les algorithmes d'ordonnancement dynamique peuvent être basés sur les techniques de file d'attente dans un modèle de publication/abonnement avec différentes stratégies telles que First In First Out (FIFO), adaptatif, etc. Les WMS tels que

Chiron (Ogasawara et al., 2013) et Galaxy (Mangala et al., 2012) exploitent des algorithmes d'ordonnancement dynamique.

## 5.3 Ordonnancement statique

L'ordonnancement statique génère un plan d'ordonnancement (PO) qui alloue toutes les tâches exécutables aux nœuds de calcul avant l'exécution et le WMS respecte strictement le plan d'ordonnancement pendant toute la durée de l'exécution du workflow scientifique. Étant donné qu'elle est antérieure à l'exécution, la planification statique génère peu de surcharge lors de l'exécution. Il est efficace si le WMS peut prédire avec précision le temps d'exécution de chaque tâche, lorsque l'environnement d'exécution varie peu pendant l'exécution du workflow et lorsque le WMS dispose de suffisamment d'informations sur les capacités de calcul et de stockage des ordinateurs correspondants. Cependant, lorsque l'environnement d'exécution subit des changements dynamiques, il est très difficile d'atteindre l'équilibre de charge. Les algorithmes de planification de tâches statiques ont deux types de méthodes de sélection de processeur (Topcuoglu et al., 2002) : basées sur une recherche heuristique et basées sur une recherche aléatoire guidée. La méthode basée sur l'heuristique planifie les tâches selon une règle prédéfinie tandis que la méthode basée sur la recherche aléatoire planifie les tâches de manière aléatoire. Dans notre thèse on s'intéresse à l'ordonnancement de tâches statique.

## 5.4 Algorithmes d'ordonnancement de Workflow

De nombreuses heuristiques ont été étudiées par plusieurs projets pour planifier des workflows dans le cloud. Les heuristiques peuvent être classées au niveau de la tâche ou au niveau du workflow. Les heuristiques au niveau des tâches prennent des décisions de planification basées uniquement sur les informations relatives à une tâche ou à un ensemble de tâches indépendantes, tandis que les heuristiques au niveau du workflow prennent en compte les informations de l'ensemble du workflow.

L'algorithme HEFT est un algorithme d'ordonnancement bien connu (Topcuoglu et al., 2002), il est considéré comme le standard de l'industrie. L'algorithme HEFT comporte deux phases: une priorisation des tâches et une phase de sélection du processeur. Dans la première phase, les priorités sont définies en tant que $rank_u$. $Rank_u$ représente la longueur du plus long chemin reliant la tâche $n_i$ au nœud de sortie, y compris le coût de calcul de $n_i$. La liste des tâches est classée par valeur décroissante de $rank_u$.

Dans la deuxième phase, appelée phase de sélection du processeur, la tâche située en haut de la liste des tâches est attribuée au processeur $p_j$ qui permet le temps EFT (Earliest Finish Time) de la tâche $n_i$. En outre, l'algorithme HEFT tente d'insérer une tâche au plus tôt, le temps mort entre deux tâches déjà planifiées sur un processeur, si l'emplacement est suffisamment grand pour accueillir la tâche.

Dans Jia Yu et al. (Yu & Buyya, 2006), les auteurs ont présenté une approche fondée sur un algorithme génétique pour résoudre les problèmes

d’optimisation d’ordonnancement dans les applications de workflow, basée sur deux contraintes de qualité de service, l’échéance et le budget.

Dans (Yassa et al., 2013), les auteurs ont proposé un nouvel algorithme d'ordonnancement des workflow basé sur un algorithme génétique capable de gérer plus de deux métriques QoS (qualité de service), à savoir: makespan, coût, fiabilité, disponibilité et énergie.

Dans (Verma & Kaushal, 2015) les auteurs ont présenté une optimisation d'essaims de particules permettant de planifier des tâches de workflows sur les ressources disponibles d’un cloud, dans des délais et des contraintes budgétaires bien définis. L'heuristique proposée a été simulée et la comparaison effectuée avec des algorithmes de référence.

Dans (Wu et al., 2013) les auteurs proposent une stratégie de planification hiérarchique orientée marché dans les systèmes de workflow cloud. Plus précisément, la planification du niveau de service traite de l'affectation des tâches des instances de workflows aux services cloud sur les marchés cloud mondiaux en fonction de leurs exigences de QoS fonctionnelles et non fonctionnelles; la planification au niveau des tâches traite de l'optimisation de l'affectation de tâche à VMs (machines virtuelles) dans les centres de données cloud locaux où le coût global de fonctionnement des workflows cloud sera minimisé compte tenu de la satisfaction des contraintes de QoS, un algorithme d'ordonnancement aléatoire basé sur un package de trois algorithmes d'ordonnancement à base de métaheuristiques, notamment l'algorithme génétique (GA), l'optimisation des colonies de fourmis (ACO) et l'optimisation des essaims de particules (PSO) ) sont adaptés, mis en œuvre et analysés.

Wei-Neng et al. (Wei-Neng Chen & Jun Zhang, 2009) ont proposé un algorithme d'optimisation de colonies de fourmis pour planifier des workflows avec divers paramètres de qualité de service. Cet algorithme permet aux utilisateurs de spécifier leurs préférences en matière de qualité de service et de définir les seuils minimum de qualité de service pour une application donnée. L'objectif de cet algorithme est de trouver une solution qui réponde à toutes les contraintes de QoS et optimise le paramètre de QoS préféré de l'utilisateur.

Dans (Lopez, 2016) les auteurs proposent un algorithme évolutif modifié basé sur les algorithmes génétiques pour sélectionner le nombre optimal de ressource pour exécuter le workflow, les opérateurs de croisement et de mutation ont été modifié pour réduire le caractère aléatoire dans l'AG et converger vers une solution finale avec un nombre réduit de générations. Cette étude a aussi exploré le problème de l'analyse de workflows plus volumineux et propose la solution PSO-DS, une adaptation du PSO discret pour résoudre le problème d'ordonnancement. Les auteurs affirment que la solution d'ordonnancement proposée basée sur le PSO converge vers les meilleures solutions après avoir considéré un grand nombre de combinaisons sans nécessiter de temps de traitement excessif.

Dans (Femmam et al., 2018), les auteurs ont proposé une nouvelle approche basée sur un formalisme de "réseau de Petri évolutif", qui est une extension du réseau de Petri, enrichi de deux opérateurs génétiques, le croisement et la mutation. En mappant les problèmes de planification dans les réseaux de Petri, le problème d'ordonnancement a été réduit à la recherche d'une séquence

optimale de transitions menant d'un marquage initial à un marquage final, pour trouver l'ordonnancement optimale.

Dans (Zhou et al., 2019), les auteurs ont proposé l'optimisation conjointe du coût et le makespan (temps d'exécution) des workflows dans le cloud, et propose un nouveau schéma d'ordonnancement des workflows. Dans ce schéma, un algorithme appelé fuzzy dominance sort based heterogeneous earliest-finish-time (FDHEFT) basé sur le tri de dominance floue est développé, il intègre étroitement le mécanisme de tri de la dominance floue avec l'algorithme d'ordonnancement de liste HEFT.

Dans (Ismayilov & Topcuoglu, 2020), les auteurs proposent un algorithme évolutif dynamique multi-objectif basé sur la prédiction, appelé algorithme NN-DNSGA-II, en incorporant un réseau de neurones artificiels avec l'algorithme NSGA-II. Les solutions d'ordonnancement prennent en compte six objectifs: minimisation du makespan, coût, énergie et degré de déséquilibre; et maximisation de la fiabilité et de l'utilisation. L'étude empirique a été menée sur le système de gestion de workflow Pegasus.

| Approche | Algorithme | Paramètres | Facteurs | Environ-nement | Outils |
|---|---|---|---|---|---|
| HEFT workflow scheduling algorithm (Topcuoglu et al., 2002) | HEFT | Makespan | Le rang plus élevé des tâches | Grille | Grid Sim |
| A particle swarm optimization based heuristic (Verma & Kaushal, 2015) | PSO | Temps cpu | DAG | Cloud | Amazon EC2 |
| Market oriented Hierarchial strategy (Wu et al., 2013) | PSO, GA<br>Ant C.O. | Coût, temps cpu | Niveau service et DAG | Cloud | Amazon EC2 |
| Workflow scheduling in cloud using fuzzy dominance sort based HEFT (Zhou et al., 2019) | HEFT<br>fuzzy sort | Makespan, coût | Le rang plus élevé des tâches | Cloud | jMetal |
| Neural network based multi-objective algorithm for workflow scheduling (Ismayilov & Topcuoglu, 2020) | DNSGA-II<br>NN | Makespan, coût, temps cpu, énergie, équilibrage, fiabilité | DAG | Cloud | Système de gestion de workflow Pegasus |
| An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem (Wei-Neng Chen & Jun Zhang, 2009) | Ant colony | Coût, temps cpu | DAG | Cloud | Cloud Sim |

| | | | | | |
|---|---|---|---|---|---|
| Scientific Workflow Scheduling for Cloud Computing<br>(Lopez, 2016) | GA, PSO | Makespan, coût | DAG | Cloud | Système de gestion de workflow Pegasus |
| A genetic algorithm for multi-objective workflow scheduling<br>(Yassa et al., 2013) | GA | Makespan, coût | DAG | Cloud | Cloud Sim |
| Petri nets/genetic algorithm based approach<br>(Femmam et al., 2018) | Réseaux de Pétri / GA | Makespan, coût | DAG | Cloud | Cloud sim |

*Tableau 5-1 Tableau comparatif des approches d'ordonnancement des workflows.*

Nous constatons que la plupart des algorithmes présentés ci-dessus et synthétisé dans le Tableau 5-1 sont des algorithmes bio-inspirées, étant donné que le problème traité est de nature combinatoire, ces approches optimisent un ou deux paramètres, à savoir le makespan tout seul, ou le makespan avec le coût d'exécution, très peu optimisent plus de trois paramètres. Dans le cas de plusieurs paramètres à optimiser, les approches déjà citées procèdent par agrégation des objectifs (un paramètre à la fois) ou par ordonnancement basé pareto, c'est à dire de telle sorte que chaque paramètre à optimiser n'affecte pas l'autre.

## 5.5 Conclusion

Dans ce chapitre, nous avons présenté les deux types d'ordonnancement de tâches en positionnant notre axe de recherche sur l'ordonnancement statique, puis nous avons dressé un état de l'art des différents algorithmes d'ordonnancement de tâches qui existent, enfin une comparaison entre ces algorithmes et une synthèse ont été données.

Très peu de travaux ont été menés sur la parallélisation de métaheuristique pour résoudre le problème d'ordonnancement, et à notre connaissance aucune approche n'a encore utilisé l'algorithme du recuit simulé pour résoudre ce problème de l'ordonnancement de workflow, ce qui nous a motivés pour proposer notre approche à savoir l'utilisation de l'algorithme du recuit simulé implémenté de façon parallèle sur une architecture GPU.

Dans le chapitre suivant nous allons présenter notre contribution dans cette thèse concernant l'ordonnancement de workflow dans un environnement de cloud computing.

# Partie II :

# Contribution

# Chapitre 6 :
# Approche proposée

## 6.1 Introduction

Le cloud offre une plateforme de plus en plus utilisée pour l'exécution des workflows scientifiques, Malheureusement, les services offerts par celui-ci ne sont généralement pas gratuits; un contrat d'accord de niveau de service signé entre le fournisseur et l'utilisateur fixera le prix à payer pour les ressources louées dans un modèle de paiement à l'utilisation.

L'ordonnancement des tâches est une étape importante lors de l'exécution de workflows dans un environnement de cloud computing, il vise principalement à optimiser le makespan qui est le temps nécessaire pour exécuter le workflow, surtout lorsque l'infrastructure d'exécution du workflow est loué ce qui engendre un impact financier non négligeable pour le client.

Le problème de l'ordonnancement de workflows est reconnu comme l'un des problèmes NP-hard les plus difficiles (Blazewicz et al., 2007) (Sharma & Rashid, 2020), ce qui justifie le recours aux méthodes heuristiques pour obtenir la solution la plus optimale possible.

Ce chapitre est composé des sections suivantes, tout d'abord, nous allons commencer par présenter la méthode du recuit simulé, une autre section est consacrée à la parallélisation en général, suivit d'une autre section consacrée à la parallélisation du recuit simulé, ensuite une section sera consacrée à présenter les processeurs graphiques (GPU), enfin on termine par une section qui présente notre approche proposée.

## 6.2 La méthode du recuit simulé

La méthode de recuit simulé (Simulated Annealing) est une généralisation de la méthode MonteCarlo; L'objectif est de trouver une solution optimale à un problème spécifique. Elle a été développé par trois chercheurs d'IBM: S. Kirkpatrick, CD Gelatt et MP Vecchi en 1983 (Kirkpatrick et al., 1983) et de façon indépendante par V. Cerny en 1985 en utilisant l'algorithme Metropolis (Černỳ, 1985); qui décrit le développement d'un système thermodynamique. Le procédé de recuit simulé est basé sur un procédé largement utilisé en métallurgie afin d'obtenir un alliage sans défaut. Ce processus est appelé "recuit".

Dans le processus de recuit simulé utilisé en métallurgie, le métal est d'abord chauffé à une certaine température où il devient liquide (par conséquent, les atomes peuvent circuler librement). Une fois cette étape atteinte, la température est réduite très lentement pour obtenir un solide. Si cette baisse de température se produit brusquement, du verre est obtenu. A l'inverse, si cette baisse de température est trop lente (ce qui laisse le temps aux atomes d'atteindre l'équilibre), une structure régulière est progressivement obtenue jusqu'à ce que l'état d'énergie minimale correspondant à la structure parfaite d'un cristal, le système est dit gelé.

Ne pas abaisser la température assez lentement peut entraîner un échec. Le matériau doit être légèrement réchauffé afin que les atomes puissent retrouver leur liberté de mouvement. Cela facilite la réorganisation des atomes et rend la structure plus stable.

### 6.2.1 Principe

L'idée principale du recuit simulé, repose su l'algorithme de Metropolis proposé en 1953 (Metropolis et al., 1953) qui décrit l'évolution d'un système thermodynamique. Il simule le comportement des matériaux dans le processus de recuit utilisé en métallurgie. L'objectif est d'atteindre un équilibre thermodynamique. Cet état d'équilibre avec une énergie minimale représente la solution optimale recherché par l'algorithme problème du recuit simulé. L'énergie du système est calculée par une fonction de coût (ou fonction objective) qui dépend du problème traité. Des paramètres de température fictifs ont été ajoutés par Kirkpatrick, Gelatt et Vecchi (Kirkpatrick et al., 1983).

Fondamentalement, le principe consiste à générer successivement des configurations à partir d'une solution initiale S0 et d'une température initiale T0 qui diminue tout au long du processus jusqu'à ce qu'une température ou un équilibre final (optimum global) soit atteint.

### 6.2.2 Algorithme

Dans l'algorithme du recuit simulé, on part d'une configuration donnée, et on lui fait subir une modification aléatoire. Si cette modification améliore la fonction objective (ou énergie du système), elle est directement acceptée ; Sinon, elle n'est acceptée qu'avec une probabilité égale à $exp(\Delta E/T)$ (avec E=énergie, et T=température), cette règle est appelée critère de Metropolis.

Cette procédure est appliquée itérativement, pour engendrer une séquence de configurations qui tendent vers l'équilibre thermodynamique :

**Algorithme du recuit simulé**

```
1  Début
2  Choisir une température de départ T=T0;
3  Choisir une solution initiale S=S0 ;
4  Tant que le critère d'arrêt n'est pas satisfait Faire
5      Tant que l'équilibre statistique n'est pas atteint Faire
7          générer une solution aléatoire S' dans le voisinage de la solution actuelle ;
8          Si f(S') < f(S) Alors S_min = S'
10         Sinon appliquer le critère de Metropolis ;
12         Fin
13     Fin
14     décroitre la température;
15     Fin
16 Fin
```

Dans un premier temps, T étant généralement choisi très grand, beaucoup de solutions - même celles dégradant la valeur de f - sont acceptées, et l'algorithme équivaut à une visite aléatoire de l'espace des solutions. Mais à mesure que la température baisse, la plupart des solutions augmentant l'énergie sont refusés, et l'algorithme se ramène à une amélioration itérative classique.

A température intermédiaire, l'algorithme autorise de temps en temps des transformations qui dégradent la fonction objective. Il laisse ainsi une chance au système de s'extraire d'un minima local.

La solution initiale peut être prise au hasard dans l'espace des solutions possibles, elle peut aussi être générée par une heuristique classique. La température initiale doit être assez élevée, car c'est elle qui fixe la probabilité d'accepter ou de refuser les solutions défavorables à l'optimisation de la fonction f.

Deux approches sont possibles pour décroitre la température :

- **Décroissance par paliers :** Pour chaque valeur de la température, on itère l'algorithme jusqu'a atteindre un équilibre, puis on diminue la température. On parle alors de paliers de température.
- **Décroissance continue :** La température est baissée de façon continue, plusieurs lois de décroissance peuvent être utilisées, la plus courante est la suivante : $T_{i+1}=\alpha.T_i$ / $\alpha<1$ (en général $\alpha$=0.9 à 0.99). Le paramètre α est à choisir avec précaution ; En effet, s'il est choisi trop grand, la température baissera très rapidement et l'algorithme pourra être bloqué dans un minima local ; Si au contraire il est choisi trop petit, la température baissera très lentement et le temps de calcul sera très grand.

## 6.3 Classification de la parallélisation des métaheuristiques

Crainic et Laporte (Crainic & Laporte, 2012) ont établi une classification des différentes implémentations parallèles des métaheuristique s, cette classification divisée en trois catégories, est basée sur l'impact de la parallélisation sur la structure algorithmique de la métaheuristique.

- La première catégorie représente une stratégie de parallélisation à l'intérieur d'une itération de la métaheuristique. Considérée comme étant une parallélisation de bas niveau ou interne elle a pour but d'accélérer les calculs sans chercher à effectuer une meilleure exploration. Le nombre d'opérations total est identique pour les deux versions parallèle et séquentielle. Il s'agit de profiter de la puissance de calcul disponible et

ainsi améliorer la recherche de solutions en utilisant cette stratégie sans changer fondamentalement l'algorithme.

- La deuxième catégorie consiste à diviser le problème en sous-problèmes et à utiliser la puissance de calcul afin de résoudre ces sous-problèmes en parallèle. Le comportement de l'algorithme parallèle est modifié par rapport à l'algorithme séquentiel. Le modèle maître-esclave est généralement utilisé dans cette catégorie, le maître détermine séquentiellement les partitions initiales et qui peuvent être modifiées durant l'exécution. Les esclaves explorent de façon concurrente et indépendante l'espace de recherche qui leur a été assigné par le maître, à la fin de l'algorithme le maître assemble les solutions partielles trouvées par les esclaves en une solution complète au problème.
- Dans la troisième catégorie, plusieurs processus de recherche (multisearch threads) sont exécutés avec des degrés de synchronisation et de coopération différents. Dans cette catégorie, on cherche à effectuer une recherche plus complète en mettant en œuvre plusieurs processus qui opèrent en parallèle. La métaheuristique peut être calibrée différemment pour chaque processus. Les processus peuvent communiquer entre eux durant leur recherche ou seulement à la fin afin d'identifier la meilleure solution.

## 6.4 Parallélisation du recuit simulé

Nous supposons que nous avons p processeurs disponibles simultanément et chaque processeur dispose de suffisamment de mémoire et de ressources informatiques pour exécuter son propre programme séquentiel.

### 6.4.1 Multiple Independent Run (MIR)

On commence par une température initiale T et une configuration initiale E. On Donne à chaque processeur cette température et cette configuration. Chaque processeur exécute l'algorithme SA en série, jusqu'à l'obtention d'un certain nombre de succès.

Chaque processeur envoie sa configuration actuelle à un processeur maître. Un algorithme de décision est appliqué pour sélectionner l'une de ces configurations. En général la configuration avec la fonction d'énergie la plus basse est choisie. Ensuite la configuration choisie par l'algorithme de décision est redistribuée à chacun des processeurs et répétez les étapes précédentes sont répétées jusqu'à ce que le nombre de réussite requis ne soit pas atteint. L'algorithme est alors terminé.

Dans cette approche on s'attend à une accélération quasi linéaire. Cela est dû au fait que, avec p processeurs, nous recherchons un facteur de p plus de configurations possibles (p étant le nombre de processeurs), nous augmentons donc les chances de "trébucher" sur la configuration correcte plus rapidement. C'est une approche facile à mettre en œuvre, le seul message transmis est transmis aux étapes de synchronisation.

Les inconvénients à reprocher à cette approche sont le fait que si un processeur obtient le nombre de succès requis avant un autre, il doit attendre que cet autre processeur termine.

Une collecte et une retransmission globales de configurations volumineuses peuvent prendre beaucoup de temps (Azencott, 1992).

### 6.4.2 Recherche simultanée avec interaction périodique (Simultaneous periodically interacting searcher)

Dans cette approche, on commence par une température initiale T et une configuration initiale E. On Donne à chaque processeur cette température et cette configuration.

Chaque processeur exécute l'algorithme SA en série, comme décrit jusqu'à ce qu'un intervalle de temps prédéfini soit atteint (par exemple, 10 secondes).

Chaque processeur envoie sa configuration actuelle à un processeur maître. Un algorithme de décision est appliqué pour sélectionner l'une de ces configurations. En général, la configuration avec la fonction d'énergie la plus basse est choisie. On distribue la configuration choisie par l'algorithme de décision à chacun des processeurs et on répète les étapes précédentes jusqu'à ce que le nombre de réussite requis ne soit pas atteint. L'algorithme est alors terminé.

On obtient une amélioration substantielle par rapport au recuit séquentiel. Un inconvénient majeure est que cette approche génère plus de coûts de communication que l'approche MIR, si le coût de la communication est élevé, elle sera moins efficace que MIR (Azencott, 1992)  (Aarts & Van Laarhoven, 1987).

### 6.4.3 Essais multiples (Multiple Trials)

Un schéma de recuit séquentiel arbitraire avec espace de configuration E et programme de refroidissement ($T_n$). À l'instant n, pour modifier la configuration actuelle $X_n$, on sélectionne un voisin aléatoire $Y_n$ de $X_n$, puis un autre choix aléatoire est effectué pour décider de conserver la configuration $X_n$ ou de la remplacer par la nouvelle configuration $Y_n$. Cette séquence de deux choix aléatoires s'appelle un essai.

On commence par une température initiale T et une configuration initiale E. On Donne à chaque processeur cette température et cette configuration. Chaque processeur exécute l'algorithme SA en série avec un seul essai.

Chaque processeur envoie sa configuration actuelle à un processeur maître. Un algorithme de décision est appliqué pour sélectionner l'une de ces configurations. En général, la configuration avec la fonction d'énergie la plus basse. La configuration choisie par l'algorithme de décision est distribuée à chacun des processeurs et les étapes précédentes sont répétées jusqu'à ce que le nombre de réussite requis ne soit pas atteint. L'algorithme est alors terminé.

On note que dans cette approche il n'y a pas d'étapes de synchronisation coûteuses. Les communications sont plus petites mais plus fréquentes.

A basse température, le taux d'accélération fourni par ce schéma de parallélisation est proche du nombre p de processeurs. À des températures élevées, il est préférable d'utiliser un petit nombre p de processeurs, tandis qu'à basse température p peut être beaucoup plus important (Azencott, 1992).

### 6.4.4 Parallélisation massive (Massive parallelization)

Dans cette approche il faut tout d'abord introduire la notion d'ensemble actif: supposons que N processeurs sont disponibles simultanément. Fixez une séquence $T_n$ de températures décroissantes. À l'instant n, chaque processeur $P_i$ décide aléatoirement, indépendamment du passé et de tous les autres processeurs, s'il appartiendra à l'ensemble actif $A_n$ avec le taux de probabilité fixe.

On commence par une température initiale T et une configuration initiale, on donne à chaque processeur en ACTIVE SET cette température et cette configuration.

Chaque processeur dans ACTIVE SET exécute l'algorithme SA en série, comme décrit dans le chapitre précédent, lors d'un seul essai.

Chaque processeur dans ACTIVE SET envoie sa configuration actuelle à un processeur maître. Un algorithme de décision est appliqué pour sélectionner l'une de ces configurations. En général, la configuration avec la fonction d'énergie la plus basse est choisie.

On distribue la configuration choisie par l'algorithme de décision à chacun des processeurs et on répète les étapes précédentes jusqu'à ce que le nombre de réussite requis ne soit pas atteint. L'algorithme est alors terminé.

Dans certains cas, si le coût de la communication est trop élevé, l'algorithme parallèle sera encore moins efficace que l'algorithme séquentiel. Nous pouvons obtenir de meilleures performances en laissant de côté un petit ensemble aléatoire de processeurs dans un recuit massivement parallèle (Azencott, 1992).

### 6.4.5 Partitionnement des configurations

Considérons un espace de configuration E et supposons que pour chaque configuration plausible x dans E, nous puissions écrire $x = (x_1, x_2,\ldots, x_p)$, où $x_j \in E_j$ et $E_j$ est un ensemble (plus petit) de configurations locales. Inversement, supposons que chaque fois que $x_1,\ldots, x_p$ appartient à $E_1 *\ldots * E_p$ et vérifie un ensemble de conditions de limites $B_{ij}$ $(x_i, x_j)$ pour toutes les paires i, j, il existe alors un x dans E tel que $x = (x_1 , x_2,\ldots, x_p)$,

Puis définissons pour chaque j = 1… p un système de voisinage $V_j+(x_j)$ de $x_j$ dans $E_j$. Considérons p processeurs $\pi_1,\ldots, \pi_p$ disponibles simultanément. Puis on Sélectionne un programme de refroidissement de la température commun $T_n \geq 0$ et laissez chaque $\pi_j$ effectuer les étapes de recuit séquentielles en commençant par la configuration $X_0$. À chaque étape, définir un protocole de communication garantissant que les conditions de compatibilité des limites restent valables pour la nouvelle configuration partielle $X_{n+1}$. Dans cette approche, le débit de communication doit être maintenu élevé lorsque le nombre de tétraèdres est faible. Travailler en parallèle n'est efficace que lorsque les domaines ont la taille minimale raisonnable.

Néanmoins, le recuit parallèle par partitionnement des configurations reste un schéma efficace du point de vue de l'accélération potentielle sur des architectures matérielles mieux adaptées (Azencott, 1992) (Bonomi & Lutton, 1984).

## 6.5 Les processeurs graphiques GPU

### 6.5.1 Classification des architectures parallèles

Plusieurs classifications des architectures parallèles ont été proposées dans la littérature. Celui de Flynn (Flynn, 1966), le plus adopté, est basé sur l'organisation des flots d'instructions transmises à partir d'une unité de contrôle vers un ou plusieurs processeurs et des flots de données provenant de la mémoire et se dirigeant vers un processeur ou inversement.

| | | **Flot de Données** | |
|---|---|---|---|
| | | **Simple** | **Multiple** |
| **Flot d'Instructions** | **Simple** | SISD (Von Neumann) | SIMD (Vectorielle et cellulaire) |
| | **Multiple** | MISD (pipeline) | MIMD (Multiprocesseur et passage de message) |

*Tableau 6-1 Taxonomie de Flynn*

La première classe d'architecture, selon la classification de Flynn, est la classe SISD (Single Instruction stream, Single Data stream) illustré dans la Figure 6-1 , qui est celle des ordinateurs séquentiels classiques, ils utilisent un unique flot d'instructions et opèrent sur un unique flot de données.

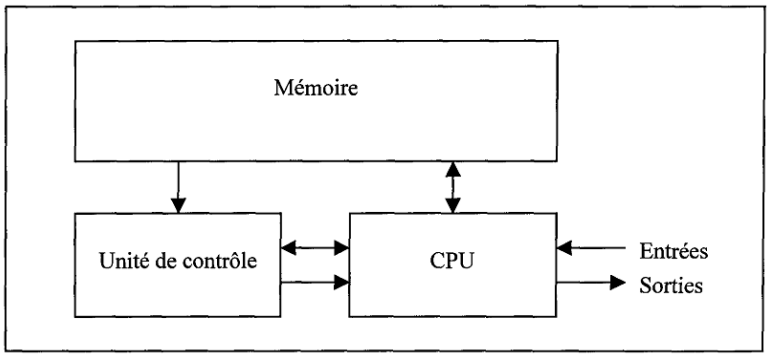



*Figure 6-1 Architecture d'un ordinateur séquentiel*

Dans une machine SIMD (Single Instruction stream, Multiple Data stream), plusieurs processeurs sont supervisés par la même unité de contrôle tel qu'illustré à la Figure 6-2. Les p processeurs (notés $P_i$) reçoivent la même instruction de l'unité de contrôle, mais opèrent de façon synchrone sur des données distinctes qui proviennent de différents flots de données, chaque processeur exécutant la même instruction à chaque unité de temps.

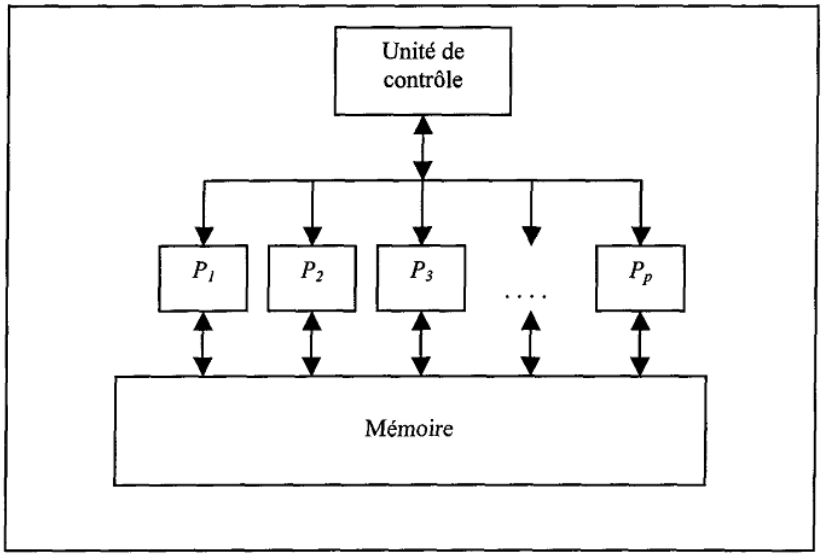



*Figure 6-2 Architecture SIMD*

La classe de machines MISD (Multiple Instruction stream, Single Data stream) applique plusieurs flots d'instructions à un unique flot de données.

Dans la classe des machines fonctionnant en mode MIMD (Multiple Instruction stream, Multiple Data stream), les processeurs sont indépendants les uns des autres et fonctionnent de façon asynchrone, chacun possédant sa propre unité de contrôle (notée $UC_i$). Un schéma d'architecture MIMD est illustré à la Figure 6-3. Comme en SIMD, chacun utilise ses propres données mais y applique, en plus, ses propres flots d'instructions.

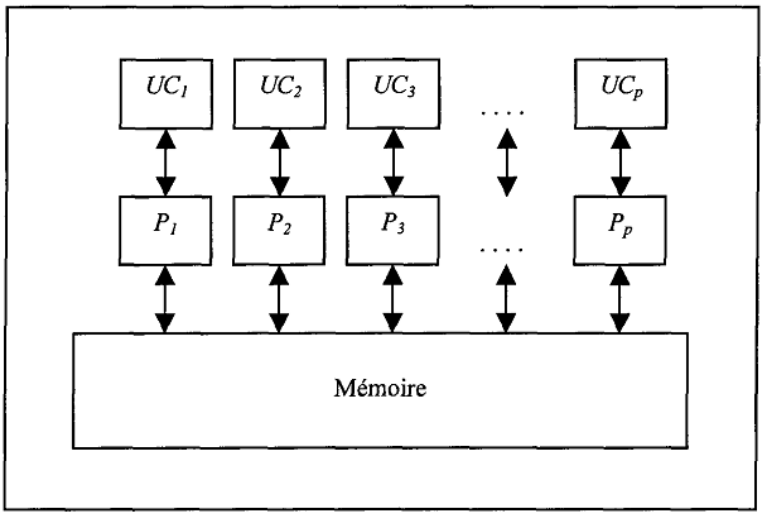



*Figure 6-3 Architecture MIMD*

Les microprocesseurs basés sur une seule unité centrale (CPU), tels que ceux des gammes Intel Pentium et AMD Opteron, ont permis d'augmenter rapidement les performances et de réduire les coûts des applications informatiques pendant plus de deux décennies. Ces microprocesseurs transmettaient giga (milliards) d'opérations à virgule flottante par seconde (GFLOPS) au PC desktop et des centaines de GFLOPS aux serveurs en cluster. Cette recherche incessante d'amélioration des performances a permis aux logiciels d'application de fournir davantage de fonctionnalités, de meilleures interfaces utilisateur et de générer des résultats plus utiles. Les utilisateurs, à leur tour, exigent encore plus d'améliorations une fois qu'ils se sont habitués à ces améliorations, créant ainsi un cycle positif pour l'industrie informatique.

Durand cette période, la plupart des développeurs de logiciels se sont appuyés sur les avancées matérielles pour accélérer la vitesse de leurs applications. Le même logiciel s'exécute simplement plus rapidement à chaque nouvelle génération de processeurs. Toutefois, cette tendance a ralenti depuis 2003 en raison de problèmes de consommation d'énergie et de dissipation de

chaleur qui ont limité l'augmentation de la fréquence d'horloge et le niveau d'activités productives pouvant être effectuées à chaque période d'horloge au sein d'une même CPU. Pratiquement tous les fournisseurs de microprocesseurs sont passés à des modèles dans lesquels plusieurs unités de traitement, appelées cœurs de processeur, sont utilisées dans chaque puce pour augmenter la puissance de traitement. Ce changement a eu un impact considérable sur la communauté des développeurs de logiciels (Sutter & Larus, 2005).

Traditionnellement, la grande majorité des applications logicielles sont écrites sous forme de programmes séquentiels, comme décrit par Von Neumann dans son rapport fondateur. L'exécution de ces programmes peut être comprise par un humain parcourant séquentiellement le code. Historiquement, les utilisateurs d'ordinateurs s'étaient habitués à attendre que ces programmes s'exécutent plus rapidement avec chaque nouvelle génération de microprocesseurs. Une telle attente n'est plus strictement valable à partir de ce jour. Un programme séquentiel ne fonctionnera que sur l'un des cœurs du processeur, ce qui ne sera pas beaucoup plus rapide que ceux utilisés aujourd'hui. Sans amélioration des performances, les développeurs d'applications ne seront plus en mesure d'introduire de nouvelles fonctionnalités dans le logiciel au fur et à mesure de l'introduction de nouveaux microprocesseurs, ce qui réduira les opportunités de croissance de l'ensemble du secteur informatique.

Au lieu de cela, les logiciels d'application qui continueront à améliorer les performances avec chaque nouvelle génération de microprocesseurs seront des programmes parallèles, dans lesquels plusieurs threads d'exécution coopéreront pour achever le travail plus rapidement. Cette nouvelle incitation à la création de programmes parallèles considérablement accrue a été qualifiée de «révolution de

la concurrence» (Sutter & Larus, 2005). La pratique de la programmation parallèle n'est nullement nouvelle.

La communauté informatique de haute performance développe des programmes parallèles depuis des décennies. Ces programmes fonctionnent sur des ordinateurs coûteux et à grande échelle. Seules quelques applications élites peuvent justifier l'utilisation de ces ordinateurs coûteux, limitant ainsi la pratique de la programmation parallèle à un petit nombre de développeurs d'applications. Maintenant que tous les nouveaux microprocesseurs sont des ordinateurs parallèles, le nombre d'applications devant être développées en tant que programmes parallèles a considérablement augmenté. Les développeurs de logiciels ont maintenant grand besoin de se familiariser avec la programmation parallèle.

### 6.5.2 Les GPU comme ordinateurs parallèles

Depuis 2003, l'industrie des semi-conducteurs s'est fixée deux grandes trajectoires pour la conception de microprocesseurs (Hwu et al., 2008). L'approche multicore cherche à maintenir la vitesse d'exécution des programmes séquentiels tout en se déplaçant dans plusieurs cœurs. Les processeurs multicores ont au départ pris la forme de processeurs à deux cœurs, le nombre de cœurs doublant à peu près à chaque génération de processus semi-conducteur.

En revanche, l'approche multicore est davantage axée sur le débit d'exécution des applications parallèles. Les nombreux noyaux ont commencé avec un grand nombre de noyaux beaucoup plus petits et, encore une fois, le nombre de noyaux double avec chaque génération. Un exemple est le NVIDIA GeForce Unité de traitement graphique (GPU) GTX 280 avec 240 cœurs, chacun

d’eux étant un processeur d’émissions fortement multithread, en ordre et à instruction unique qui partage son cache de contrôle et d’instructions avec sept autres cœurs.

Les processeurs many-cores, en particulier les GPU, sont en tête de la course aux performances en virgule flottante depuis 2003. Alors que l'amélioration des performances des microprocesseurs à usage général a considérablement ralenti, les GPU ont continué de s'améliorer sans relâche. À partir de 2009, le rapport entre les processeurs graphiques multi-cœurs et les processeurs multicores pour le débit de calcul maximal en virgule flottante est d'environ 10 à 1. Il ne s'agit pas nécessairement de vitesses d'application réalisables, mais simplement de la vitesse brute que les ressources d'exécution peuvent potentiellement prendre en charge dans ces puces. : 1 téraflops (1000 gigaflops) contre 100 gigaflops en 2009.

Un tel écart de performance entre l'exécution parallèle et séquentielle s'est traduit par une accumulation de «potentiel électrique» significative, et à un moment donné, quelque chose devra céder. Nous avons atteint ce point maintenant. À ce jour, cet écart important en termes de performances a déjà motivé de nombreux développeurs d'applications à déplacer les parties de leur logiciel, qui exigent beaucoup de calculs, vers des GPU pour exécution. Il n’est donc pas surprenant que ces composants informatiques intensifs constituent également la principale cible de la programmation parallèle: lorsque vous avez encore du travail à faire, vous avez plus de possibilités de répartir le travail entre les travailleurs parallèles ayant coopéré.

On pourrait se demander pourquoi il y a un tel écart de performances entre les GPU à plusieurs cœurs et les processeurs multicœurs polyvalents. La réponse

réside dans les différences de philosophie de conception fondamentale entre les deux types de processeurs, illustrées à la Figure 6-4. La conception d'un processeur est optimisée pour les performances de code séquentiel. Il utilise une logique de contrôle sophistiquée pour permettre aux instructions d'un seul thread d'exécution de s'exécuter en parallèle ou même en dehors de leur ordre séquentiel, tout en maintenant l'apparence de l'exécution séquentielle. Plus important encore, de grandes antémémoires sont fournies pour réduire les latences d'instruction et d'accès aux données d'applications complexes de grande taille. Ni la logique de contrôle ni les mémoires cache ne contribuent à la vitesse de calcul maximale. Depuis 2009, les nouveaux microprocesseurs multi-cœur polyvalents ont généralement quatre grands cœurs de processeur conçus pour offrir des performances de code séquentiel élevées.

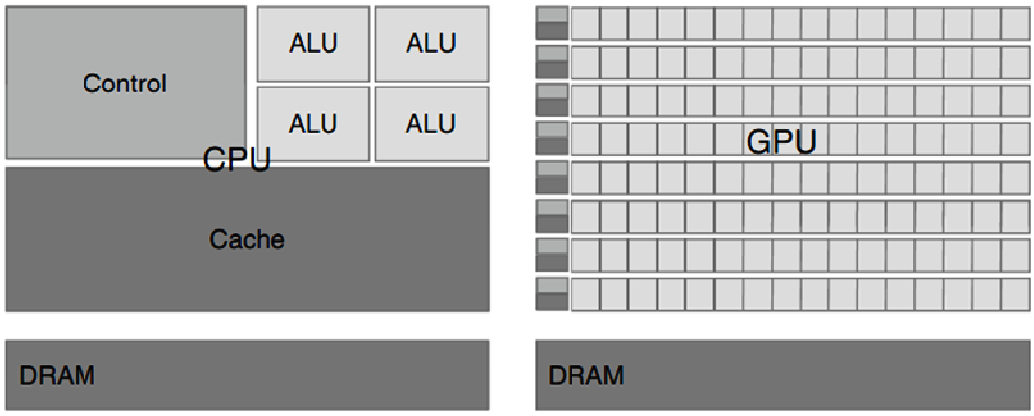



*Figure 6-4 Philosophie de conception des CPUs et GPUs*

La bande passante mémoire est un autre problème important. Les puces graphiques ont fonctionné à environ 10 fois la bande passante des puces de processeur disponibles simultanément. Fin 2006, la GeForce 8800 GTX, ou plus simplement la G80, était capable de déplacer des données à environ 85 gigaoctets par seconde (Go/s) dans et hors de sa principale mémoire vive dynamique

(DRAM). En raison des exigences en matière de mémoire tampon de trames et du modèle de mémoire assoupli (façon dont divers logiciels système, applications et périphériques d'entrée/sortie (E/S) attendent de leurs accès mémoire), les processeurs à usage général doivent satisfaire aux exigences des systèmes d'exploitation, des applications et les périphériques d'E/S qui rendent la bande passante mémoire plus difficile à augmenter. En revanche, avec des modèles de mémoire plus simples et moins de contraintes héritées, les concepteurs de GPU peuvent obtenir plus facilement une bande passante mémoire supérieure. Le récent NVIDIA La carte GeForce RTX 3090 prend en charge environ 936 Go/s de bande passante mémoire, une mémoire de 24 Go et 10496 cœurs CUDA (*NVIDIA*, 2020) .

La philosophie de conception des GPU est façonnée par le secteur en plein essor du jeu vidéo, qui exerce une pression économique énorme pour pouvoir effectuer un nombre considérable de calculs en virgule flottante par image vidéo dans des jeux avancés. Cette demande motive les fournisseurs de GPU à rechercher des moyens d'optimiser la surface de la puce et le budget énergétique dédié aux calculs en virgule flottante. La solution qui prévaut à ce jour consiste à optimiser le débit d'exécution d'un grand nombre de threads. Le matériel utilise un grand nombre de threads d'exécution pour trouver du travail lorsque certains d'entre eux attendent des accès mémoire à latence longue, minimisant ainsi la logique de contrôle requise pour chaque thread d'exécution. De petites mémoires cache sont fournies pour aider à contrôler les besoins en bande passante de ces applications, de sorte que plusieurs threads qui accèdent aux mêmes données en mémoire ne doivent pas tous accéder à la mémoire vive dynamique. En

conséquence, une plus grande surface de puces est dédiée aux calculs en virgule flottante.

Il devrait être clair maintenant que les GPU sont conçus comme des moteurs de calcul numériques et qu'ils ne fonctionneront pas bien dans certaines tâches sur lesquelles les CPU sont conçus pour bien fonctionner. Par conséquent, il est à prévoir que la plupart des applications utiliseront à la fois des processeurs et des GPU, en exécutant les parties séquentielles sur le CPU et les parties à forte intensité numérique sur les GPU. C'est pourquoi CUDA, le modèle de programmation (Compute Unified Device Architecture), introduit par NVIDIA en 2007, est conçu pour prendre en charge l'exécution conjointe d'une application par le processeur et le processeur graphique.

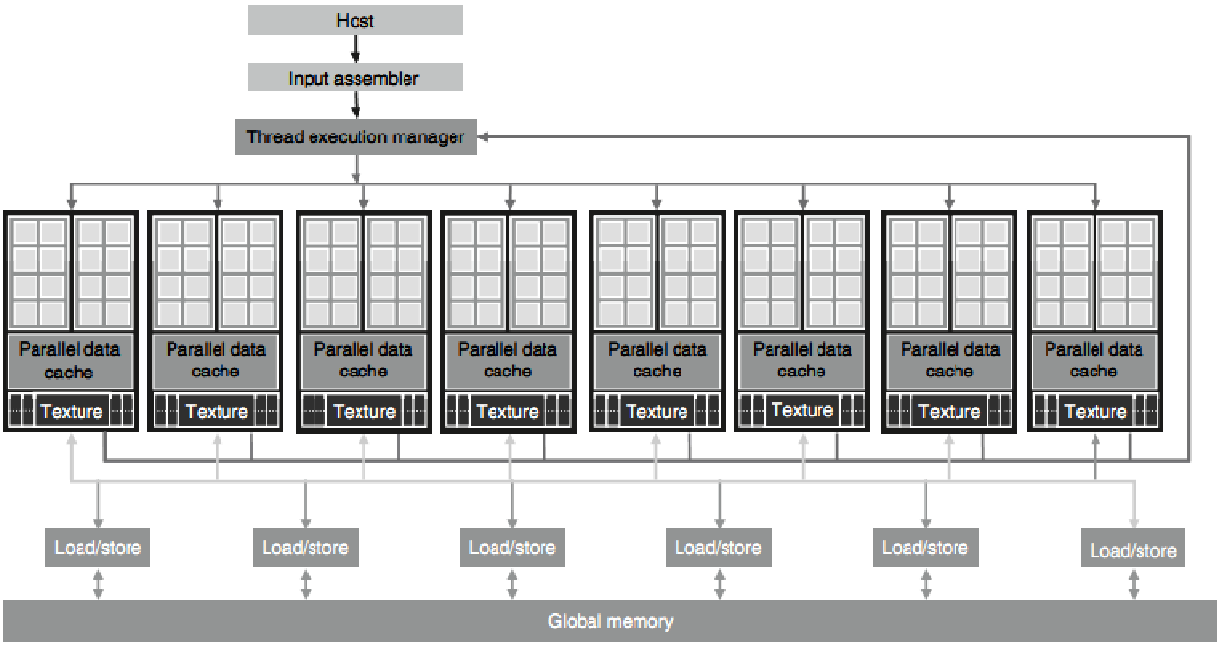



*Figure 6-5 Architecture d'un GPU (Kirk & Wen-Mei, 2016)*

Il est également important de noter que les performances ne sont pas le seul facteur de décision lorsque les développeurs d'applications choisissent les processeurs pour exécuter leurs applications. Plusieurs autres facteurs peuvent être encore plus importants. D'abord et avant tout, les processeurs de choix doivent avoir une très grande présence sur le marché, appelée la base d'installation du processeur. La raison est très simple. Le coût du développement de logiciels est mieux justifié par un très grand nombre de clients. Les applications exécutées sur un processeur peu présent sur le marché n'auront pas une base de clients importante. Cela a été un problème majeur avec les systèmes informatiques parallèles traditionnels dont la présence sur le marché est négligeable par rapport aux microprocesseurs à usage général.

Seules quelques applications élites financées par le gouvernement et de grandes entreprises ont été développées avec succès sur ces systèmes informatiques parallèles traditionnels. Cela a changé avec l'avènement des GPU à plusieurs cœurs. En raison de leur popularité sur le marché des ordinateurs personnels, des centaines de millions de GPU ont été vendus. Les processeurs G80 et leurs successeurs ont livré plus de 200 millions. C'est la première fois qu'un calcul massivement parallèle est réalisable avec un produit grand public. Une présence aussi importante sur le marché a rendu ces GPU attrayants sur le plan économique pour les développeurs d'applications.

D'autres facteurs de décision importants sont les facteurs de forme pratiques et l'accessibilité facile. Jusqu'en 2006, les applications logicielles parallèles fonctionnaient généralement sur des serveurs de centres de données ou des grappes de départements, mais de tels environnements d'exécution ont tendance à limiter l'utilisation de ces applications. Par exemple, dans une

application telle que l'imagerie médicale, il est judicieux de publier un document basé sur une machine en grappe à 64 nœuds, mais les applications cliniques réelles sur les machines d'imagerie par résonance magnétique (IRM) reposent toutes sur une combinaison d'un PC et de logiciels spéciaux. La raison en est simple: des fabricants tels que GE et Siemens ne peuvent pas vendre d'IRM avec des groupes de grappes à des environnements cliniques, mais cela est courant dans les établissements universitaires. En fait, les Instituts nationaux de la santé (NIH) ont refusé de financer des projets de programmation parallèle pendant un certain temps. Ils ont estimé que l'impact des logiciels parallèles serait limité, car d'énormes machines à grappes ne fonctionneraient pas en clinique. Aujourd'hui, GE expédie des produits IRM avec GPU et NIH finance ses recherches à l'aide de l'informatique GPU.

Un autre élément important à prendre en compte lors de la sélection d'un processeur pour l'exécution d'applications informatiques numériques est la prise en charge de la norme à point flottant de l'Institut des ingénieurs électriciens et électroniciens (IEEE). La norme permet d'obtenir des résultats prévisibles entre les processeurs de différents fournisseurs. Bien que la prise en charge de la norme à virgule flottante IEEE n'était pas forte dans les premiers GPU, elle a également changé pour les nouvelles générations de GPU depuis l'introduction du G80. La prise en charge du processeur graphique par le standard à virgule flottante IEEE est devenue comparable à celle des processeurs. En conséquence, on peut s'attendre à ce que davantage d'applications numériques soient portées sur des GPU et génèrent des valeurs comparables à celles des CPU.

Aujourd'hui, il reste un problème majeur: les unités arithmétiques en virgule flottante des GPU sont principalement à simple précision. Les applications

qui nécessitent réellement une virgule flottante double précision ne conviennent pas à l'exécution du processeur graphique. Toutefois, cela a changé avec les récents GPU, dont la vitesse d'exécution en double précision est environ deux fois moins rapide que celle en simple précision, un niveau atteint par les cœurs de processeur haut de gamme. Cela rend les GPU adaptés à encore plus d'applications numériques.

Les processeurs graphiques (GPU) se sont imposés comme une force majeure dans le calcul haute performance au cours des dernières années. Les processeurs de type "données multiples/une seule instruction" des GPU modernes permet l'exécution en parallèle de centaines de threads en même temps.

Il existe aujourd'hui quatre API GPU principales, dont deux sont spécifiques au fournisseur et les deux API restantes sont universelles. NVidia CUDA, une API spécifique au fournisseur, est considérée comme un langage de programmation GPU prédominant dans le domaine du calcul haute performance. Son langage, CUDA-C (Kirk & Wen-Mei, 2016), est une extension du langage de programmation C qui permet le développement de routines GPU appelées noyaux. Chaque noyau définit une séquence d'instructions qui sont exécutées sur le GPU par plusieurs threads en même temps suivant le modèle SIMD parallèle aux données.

## 6.6 Approche proposée

Ci-dessous, nous présentons notre approche proposée pour résoudre le problème de planification des tâches d’un workflow dans un contexte de cloud computing, appelé Recuit Simulé Parallèle (PSA), l’idée principale de notre approche consiste à paralléliser l’algorithme de recuit simulé et une architecture GPU. Afin de

valider les performances de notre approche, une version séquentielle de l'algorithme de recuit simulé a été implémentée et appelée SSA dans la suite de la thèse.

Le recuit simulé a été introduit pour la première fois par Kirkpatrick et al. (Kirkpatrick et al., 1983). L'algorithme repose sur une combinaison d'idées issues de l'optimisation combinatoire et de la physique statistique. D'une part, l'algorithme peut être considéré comme une généralisation de l'approche d'amélioration itérative bien connue des problèmes d'optimisation combinatoire, d'autre part, il peut être considéré comme l'analogue d'un algorithme utilisé en physique statistique pour la simulation informatique du recuit d'un solide à l'état avec un minimum d'énergie.

Le recuit simulé accepte parfois des transitions vers des solutions de qualité inférieure afin d'éviter d'être piégé dans des minima locaux médiocres selon le critère dit de Metropolis. Plus précisément, si w et w' sont les deux solutions à choisir, l'algorithme continue avec la solution w' avec une probabilité d'acceptation donnée par l'équation (1):

$$Accept(t) \stackrel{\text{def}}{=} \begin{cases} 1 & si\ coût(\omega\prime) \leq coût(\omega) \\ e^{(coût(\omega) - coût(\omega\prime))/t} & si\ coût(\omega\prime) > coût(\omega) \end{cases} \quad (1)$$

Où t est un paramètre de contrôle positif qui sera progressivement réduit à zéro lors de l'exécution de l'algorithme, t représente la température du processus de recuit physique. Cette probabilité d'acceptation diminue pour les valeurs décroissantes de t et les transitions dans lesquelles le coût des solutions diminue sont toujours acceptées.

Dans notre algorithme, nous utilisons le même modèle de planification que celui utilisé dans l'algorithme HEFT (Topcuoglu et al., 2002). L'algorithme HEFT planifie les tâches dans un ordre basé sur la valeur d'un attribut de rang. Cela garantit qu'une tâche n'est affectée à un processeur que si tous ses prédécesseurs sont affectés. L'attribut rank de chaque tâche est calculé, comme indiqué dans l'équation (2), en fonction des temps de communication et de calcul des tâches.

$$rank(n_i) = w_i + \max_{n_j \in succ(n_i)}(c_{ij} + rank(n_j)) \quad (2)$$

Où succ (ni) est l'ensemble des successeurs immédiats de la tâche ni, cij est le coût de communication moyen de edge (i, j) et wi est le coût moyen de calcul de la tâche ni. Le rang est calculé de manière récursive en parcourant le graphe de tâches en partant de la tâche de sortie.

Pour appliquer le recuit simulé au problème d'ordonnancement de workflow, les notions de configuration, de fonction de coût et de structure de voisinage doivent être définies avec précision.

| Tâches | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Processeur | 2 | 1 | 2 | 3 | 2 | 3 | 2 | 1 | 3 | 1 |

*Tableau 6-2 Allocation des processeurs aux tâches*

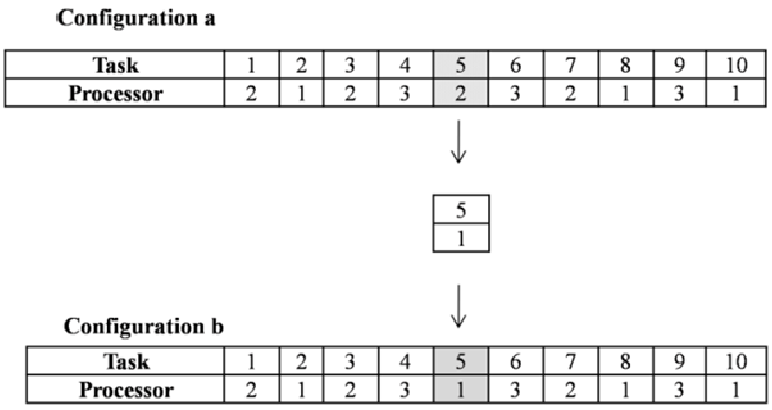



*Figure 6-6 Configuration de voisinage*

À partir de la configuration a, une nouvelle solution est générée en affectant une tâche choisie de manière aléatoire (tâche 5 comme dans la Figure 6-6) à un nouveau processeur également choisi de manière aléatoire.

La fonction de coût implémentée dans notre algorithme est définie comme suit: Soit une configuration (allocation de processeurs), comme dans le Tableau 6-2, représentée graphiquement à la Figure 6-8 (solution Heft), où chaque tâche du workflow est affectée à un processeur particulier. La fonction de coût calcule le temps nécessaire à l'exécution du workflow, appelé makespan, en tenant compte du temps d'exécution de chaque tâche sur le processeur auquel elle a été affectée, ainsi que du temps de transfert des données entre les tâches dépendantes si celles-ci sont exécutées sur différents processeurs.

### 6.6.1 Description de l'algorithme PSA

Le parallélisme dans le recuit simulé peut être classé en trois approches (1) algorithmes de type série, (2) algorithmes générés modifiés et (3) algorithmes asynchrones (Greening, 1990). Chaque approche fait un compromis entre

l'exactitude de la fonction de coût, la génération d'états, le parallélisme et le surcoût de communication. D'après l'étude réalisée par (Greening, 1990), il semble que les algorithmes de génération asynchrones et modifiés ont fourni la meilleure accélération globale. C'est pourquoi nous avons choisi d'adopter la technique de parallélisation asynchrone pour paralléliser l'algorithme du recuit simulé. L'organigramme de l'algorithme PSA est décrit dans la Figure 6-7 :

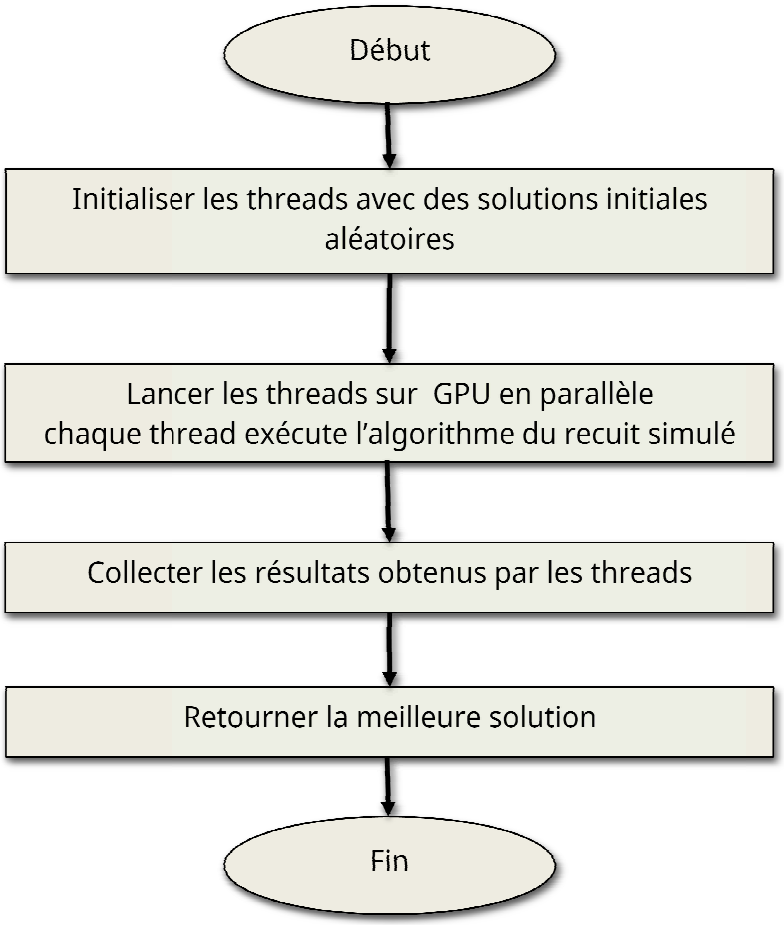



*Figure 6-7 Organigramme de l'algorithme PSA*

L'algorithme du recuit simulé utilisé dans notre approche peut être décrit comme suit :

**Algorithme du recuit simulé de notre approche**

```
1  Début
2  Sélectionner une solution initiale ω ∈ Ω;
3  Initialiser le compteur de changement de température k=0;
4  Sélectionner un schéma de refroidissement t_k ;
5  Sélectionner une température initiale T = t_0 >= 0;
6  Définir M_k le nombre d'itérations exécutés à chaque température;
7  Répéter
8      Mettre compteur de répétition m ← 0;
9      Répéter
10         Générer une solution ω' ← Voisinage(ω) ;
11         Calculer Δ_{ω,ω'} = coût(ω') - coût(ω);
12         Si Δ_{ω,ω'} <= 0 alors ω ← ω';
13         Si Δ_{ω,ω'} > 0 alors ω ← ω' avec une probabilité exp(-Δ_{ω,ω'}/t_k);
14         m ← m+1;
15         Jusqu'à m = M_k
16     k ← k+1;
17     Jusqu'à critère d'arrêt vérifié
18 Fin
```

L'algorithme part d'une solution initiale et se déplace itérativement vers d'autres solutions voisines, tout en se rappelant la meilleure solution trouvée jusqu'ici. Généralement, l'algorithme ne se déplace que vers des solutions voisines qui sont meilleures que la solution actuelle. Cependant, pour permettre à l'algorithme de s'échapper des minima locaux, passer d'une solution ω' moins bonne ω est autorisée avec une probabilité donnée, c'est le critère Metropolis, cette probabilité d'acceptation diminue avec la température.

L'approche adoptée dans cette thèse pour paralléliser l'algorithme du recuit simulé est la suivante : les threads sont initialement créés sur le GPU et initialisés avec des solutions aléatoires. Chaque thread exécute l'algorithme du recuit simulé décrit précédemment de manière asynchrone, ce qui signifie que la

communication et la synchronisation ne sont pas autorisées entre les threads, et quand il termine, il retourne un résultat. Nous choisissons ensuite le meilleur résultat parmi tous les résultats obtenus.

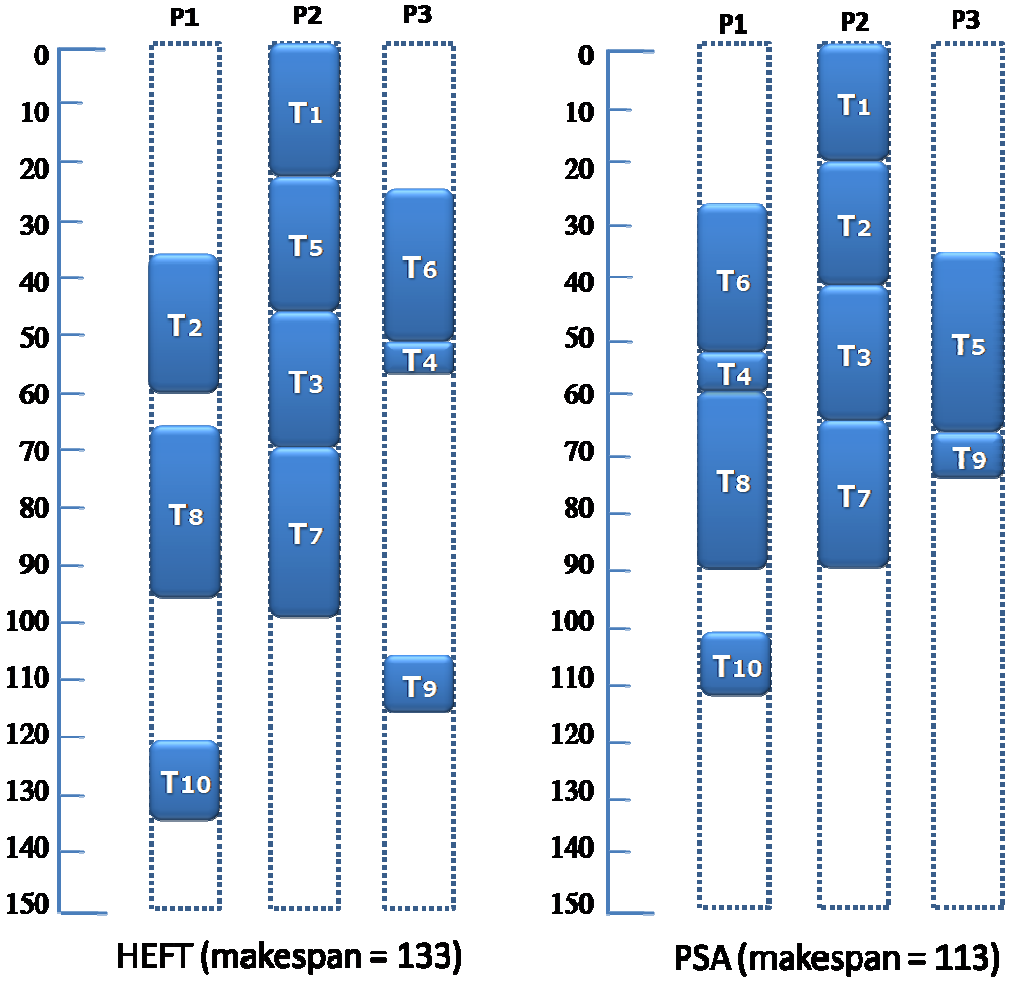



*Figure 6-8* Ordonnancements des tâches du graphe de la *Figure 4-1* obtenus avec HEFT (makespan=133) et PSA (makespan=113)

La Figure 6-8 montre les résultats de l'ordonnancement pour le DAG exemple représenté à la Figure 4-1 obtenus par les algorithmes HEFT et PSA.

En comparant les ordonnancements de HEFT et de PSA, nous pouvons voir que PSA nous donne un ordonnancement avec un makespan inférieur (113) à

celui obtenu par HEFT (133). Cet exemple n'est utilisé que pour illustrer l'algorithme, mais comme indiqué dans la section des résultats, le PSA donne de meilleurs résultats que l'algorithme référence HEFT.

## 6.7 Conclusion

Ce chapitre a été consacré à la présentation de notre approche pour l'ordonnancement des tâches de workflow dans un environnement de cloud computing. « appelée Parallel Simulated Annealing ».

Notre approche implémente l'algorithme du recuit simulé de façon parallèle en utilisant les possibilités offertes par les cartes GPU. Notre approche est mono objective, c'est-à-dire qu'elle optimise un seul paramètre qui est le makespan qui indique le temps d'exécution du workflow et qui est sans doute le paramètre le plus important lorsqu'il s'agit d'exécuter un workflow,

L'algorithme du recuit simulé utilisé dans notre approche est un algorithme basé sur une combinaison d'idées issues de l'optimisation combinatoire et de la physique statistique, il a été introduit pour la première fois par Kirkpatrick et al. (Kirkpatrick et al., 1983). Il est considéré comme une généralisation de l'approche d'amélioration itérative bien connue des problèmes d'optimisation combinatoire.

Une classification des différentes implémentations de la parallélisation des métaheuristiques en général et de la parallélisation du recuit simulé en particulier a été présentée, suivi d'une présentation des unités de traitement graphique (GPU) qui sont devenues une force majeure du calcul haute performance au cours des

dernières années, elles permettent l'exécution en parallèle de centaines de threads en même temps suivant le modèle SIMD de façon parallèle aux données.

Dans le chapitre suivant nous allons présenter l'environnement d'expérimentation ainsi que les résultats obtenus lors de la phase de validation de notre approche proposée.

# Chapitre 7 :
# Expérimentation et résultats

## 7.1 Introduction

Dans ce chapitre, nous présentons la phase de validation de notre approche proposée, ce chapitre est découpé en trois sections. La première présente la méthodologie appliquée pour exécuter les simulations ainsi que le générateur de graphes aléatoires utilisé, la seconde présente les métriques utilisées pour mener à bien les comparaisons des résultats obtenus des différents algorithmes. La troisième expose et commente les graphiques des résultats de ces simulations obtenus avec l'algorithme HEFT et l'algorithme PSA, en utilisant différentes configurations, en exposant des analyses détaillées sur les métriques utilisés.

## 7.2 Le générateur de graphes aléatoires (Random Graph Generator)

Afin d'évaluer la performance relative des algorithmes, nous avons considéré les graphes de workflow générés aléatoirement. À cette fin, nous avons utilisé un programme synthétique de génération de DAG appelé DAGGen (Suter & Hunold, 2013) . DAGGen est un générateur de graphe de tâches synthétique utilisé pour générer des graphes de tâches synthétiques aléatoires à des fins de simulation. Cet outil est largement utilisé pour évaluer les algorithmes de planification qui doivent être testés sur une large gamme de configurations de workflows.

Comme expliqué ci-après, cinq paramètres définissent la forme du DAG:

• n: nombre de nœuds de calcul dans le DAG (c.-à-d. tâches);

• fat: ce paramètre affecte la hauteur et la largeur du DAG.

• densité: détermine le nombre d'arêtes entre deux niveaux du DAG

• régularité: la régularité détermine l'uniformité du nombre de tâches dans chaque niveau.

• saut: indique qu'une arête peut aller du niveau l au niveau l + saut.

Nous avons utilisé les paramètres suivants pour obtenir les coûts de calcul et de communication de la structure DAG synthétique:

n = [10; 20; 50; 100]

fat = 0.8

densité = 0.8

saut = 1

régularité = 0: 8

## 7.3 Métriques de Comparaison

Les paramètres de comparaison utilisés sont le makespan, l'efficacité et la comparaison par paires du nombre d'occurrences de meilleures solutions (Arabnejad & Barbosa, 2014).

### 7.3.1 Makespan

La métrique la plus couramment utilisée pour évaluer une planification pour un DAG est le makespan. Le makespan ou la longueur de l'ordonnancement est l'heure de fin de la dernière tâche du DAG planifié et est définie par l'équation (3).

$$makespan = max\{AFT(nexit)\} \quad (3)$$

où AFT (nexit) désigne l'heure de fin réelle du nœud de sortie. S'il existe plusieurs nœuds de sortie, le makespan est l'heure maximale de fin réelle de tous les nœuds de sortie.

### 7.3.2 Le nombre d'occurrences de meilleurs Solutions

Cette comparaison par paires est présentée sous la forme d'un tableau, comme dans le Tableau 7-1, où le pourcentage de solutions meilleures, égales et pires produites par notre algorithme est comparé à celui d'autres algorithmes.

| | | PSA | SSA | HEFT |
|---|---|---|---|---|
| PSA | Better | ● | 76,75% | 93,84% |
| | Worse | | 0% | 0,58% |
| | Equal | | 23,25% | 5,58% |
| SSA | Better | 0% | ● | 92,16% |
| | Worse | 76,75% | | 2,17% |
| | Equal | 23,25% | | 5,67% |
| HEFT | Better | 0,58% | 2,17% | ● |
| | Worse | 93,83% | 92,16% | |
| | Equal | 5,58% | 5,67% | |

*Tableau 7-1 Comparaison des performances de planification par paire de HEFT et PSA*

Le Tableau 7-1 montre le pourcentage de résultats meilleurs, égaux et pires pour PSA par rapport aux algorithmes SSA et HEFT, basés sur le makespan. Par rapport à HEFT, PSA réalise un meilleur ordonnancement dans 93,84% des exécutions, des ordonnancements équivalents dans 5,58% des exécutions et des ordonnancements moins bons dans 0,58% des exécutions.

### 7.3.3 Efficacité

L'efficacité est définie par l'accélération divisée par le nombre de processeurs utilisés dans chaque exécution. L'accélération est définie comme le rapport entre le temps d'exécution séquentiel et le temps d'exécution parallèle.

Le temps d'exécution séquentielle est calculé en affectant toutes les tâches à un seul processeur qui minimise le coût total de l'exécution des tâches du graphe, comme le montre l'équation suivante (4).

$$Speedup = \frac{\min_{pj \in P}\left[\sum_{n_i \in V} \omega(i,j)\right]}{makespan(solution)} \quad (4)$$

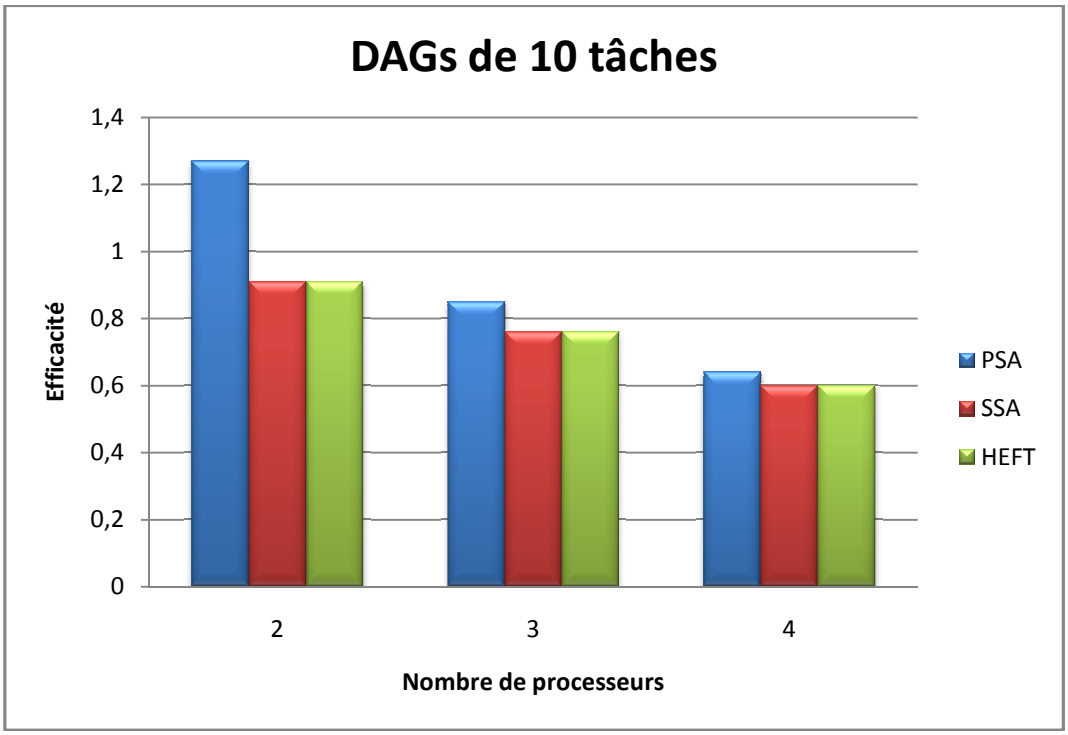



*Figure 7-1 Efficacité de* HEFT, SSA et PSA en fonction du nombre de processeurs avec des DAGs aléatoires de 10 tâches.

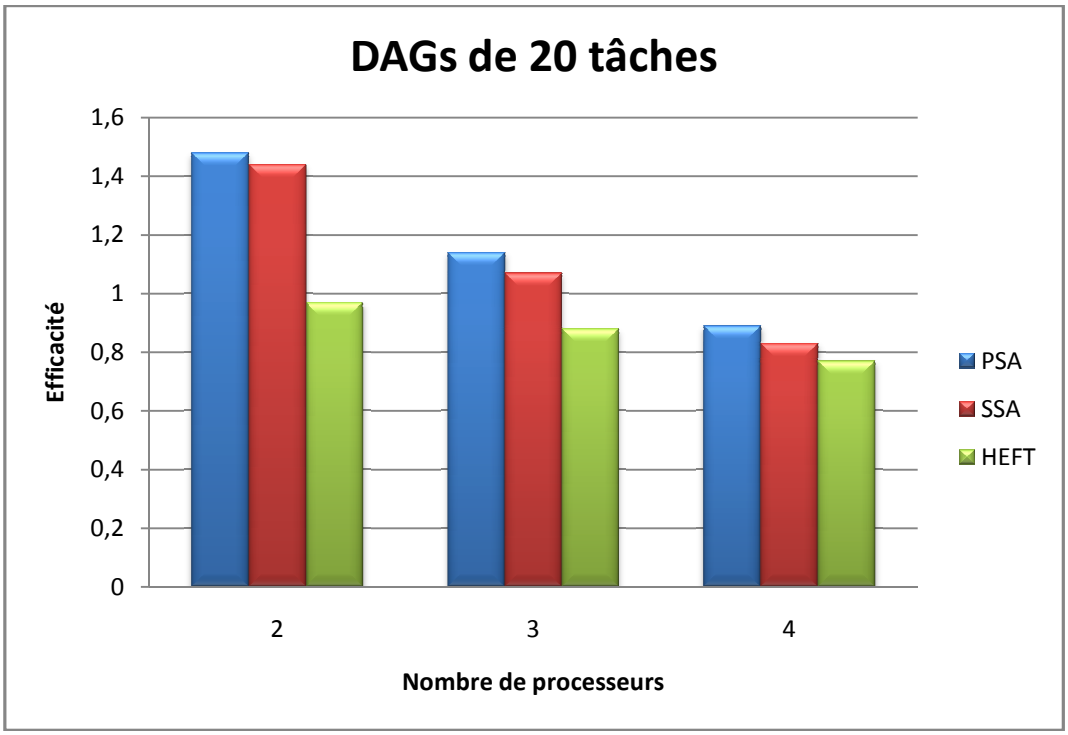



*Figure 7-2 Efficacité de* HEFT, SSA et PSA en fonction du nombre de processeurs avec des DAGs aléatoires de 20 tâches.

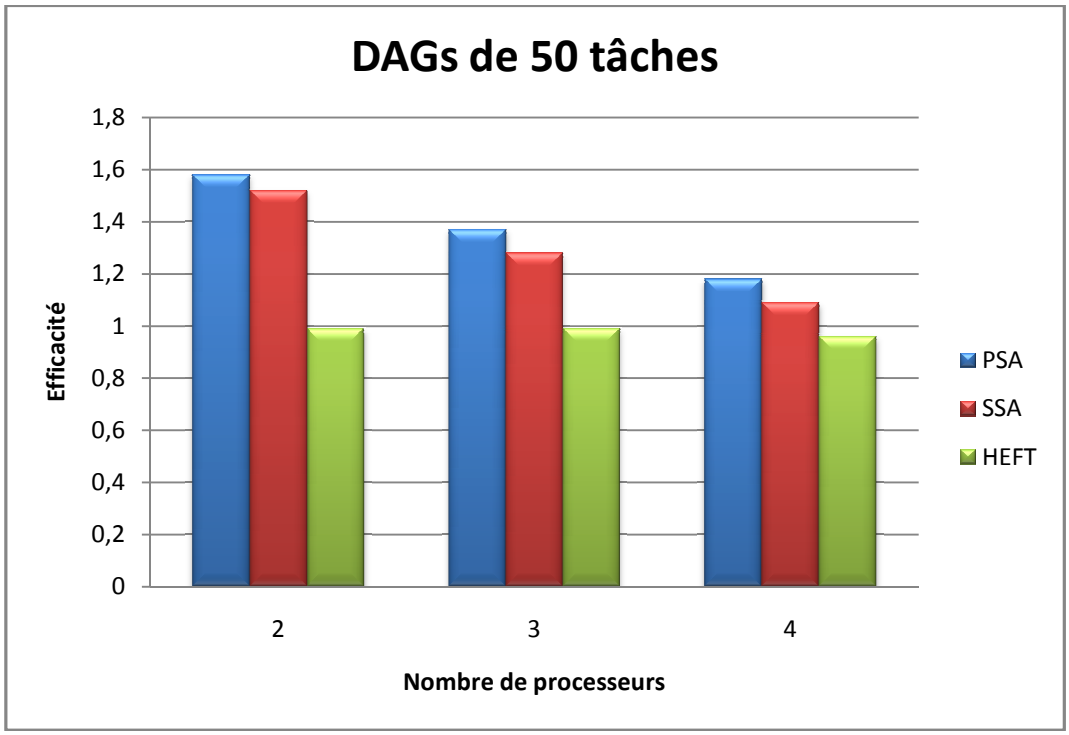



*Figure 7-3 Efficacité de* HEFT, SSA et PSA en fonction du nombre de processeurs avec des DAGs aléatoires de 50 tâches.

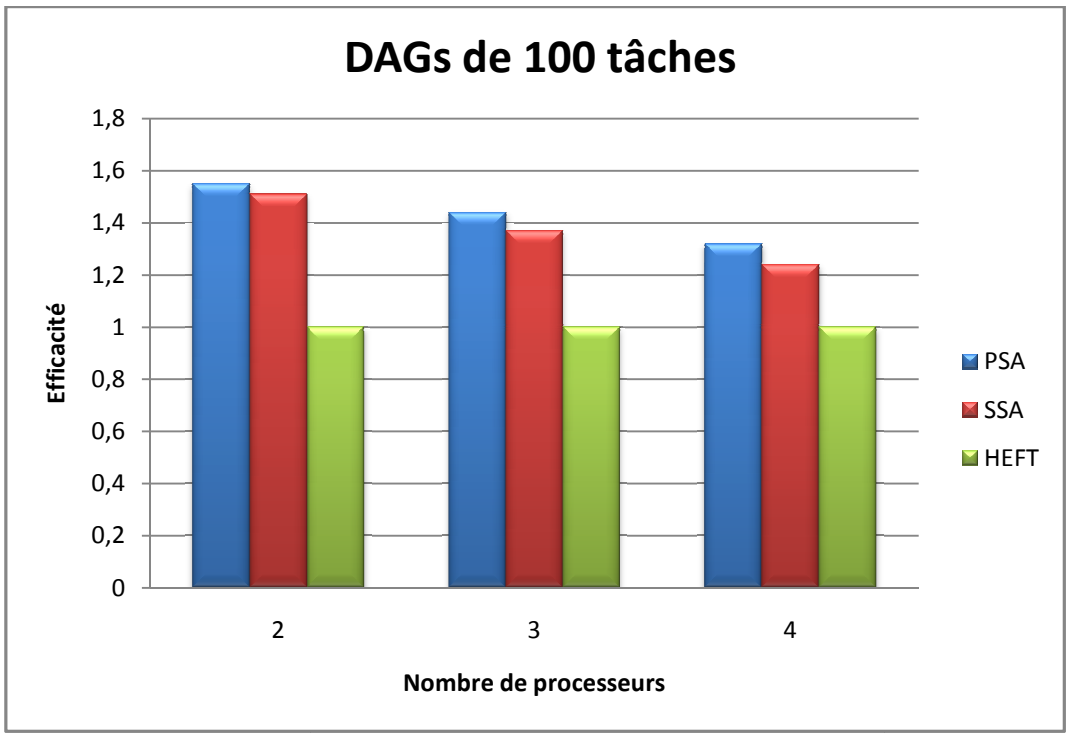



*Figure 7-4 Efficacité de* HEFT, SSA et PSA en fonction du nombre de processeurs avec des DAGs aléatoires de 100 tâches.

Dans La Figure 7-1, on constate que lorsque l'efficacité de notre algorithme est étudiée pour des graphes (DAG) de 10 tâches, ce dernier affiche une efficacité de 39% plus que HEFT lorsqu'on essaye d'affecter 2 processeurs pour exécuter nos DAGs, 12% plus que HEFT avec 3 processeurs et 6% plus que HEFT avec 4 processeurs. Dans tous les cas la version séquentielle de notre algorithme (SSA) fait aussi bien que HEFT.

Dans la Figure 7-2, avec des graphes de 20 tâches, notre algorithme enregistre une efficacité de 52% de plus que HEFT et 3% plus que SSA lorsqu'on essaye d'affecter 2 processeurs pour exécuter nos DAGs, 63% plus que HEFT et 6% plus que SSA avec 3 processeurs et 15% d'efficacité de plus que HEFT et 7% que SSA avec 4 processeurs.

Dans la Figure 7-3, avec des graphes de 50 tâches, notre algorithme affiche une efficacité de 60% de plus que HEFT et 4% plus que SSA lorsqu'on essaye d'affecter 2 processeurs pour exécuter nos DAGs, 38% plus que HEFT et 7% plus que SSA avec 3 processeurs et 23% d'efficacité de plus que HEFT et 8% que SSA avec 4 processeurs.

Dans la Figure 7-4, avec des graphes de 100 tâches cette fois, notre algorithme donne 55% d'efficacité de plus que HEFT, 3% que SSA lorsqu'on essaye d'affecter 2 processeurs pour exécuter nos DAGs, 44% d'efficacité de plus que HEFT et 5% plus que SSA avec 3 processeurs, 32% d'efficacité de plus que HEFT et 6.5% que SSA avec 4 processeurs.

D'après les résultats, on peut dire que l'efficacité des trois algorithmes diminue lorsque le nombre de processeurs augmente, et il est clair que PSA

surpasse HEFT et SSA dans tous les cas, mais surtout lorsque le nombre de processeurs est inférieur.

## 7.4 Conclusion

Dans ce chapitre nous avons présenté la partie expérimentation de notre travail ainsi que les résultats obtenus, ce qui a permis de mettre en avant les performances de notre approche.

La deuxième section de ce chapitre a présenté toute la méthodologie adoptée pour la mise en place des simulations que nous avons menées pour évaluer les performances de notre algorithme. Les résultats obtenus ont été présenté dans la section trois, ils montrent clairement que notre algorithme proposé (PSA) utilisé pour l'ordonnancement de workflows dans un environnement de cloud computing en optimisant le temps d'exécution (makespan), donne des résultats meilleures de façon notable que celles obtenus avec HEFT un algorithme d'ordonnancement considéré comme le standard de l'industrie.

# Chapitre 8 :
# Conclusion générale

Dans cette thèse, nous avons abordé le problème d'ordonnancement des tâches des applications de workflow dans un environnement de cloud computing, en s'intéressant uniquement à l'optimisation du makespan qui est le temps d'exécution du workflow. Une nouvelle approche, basée sur la parallélisation de la métaheuristique du recuit simulé sur une architecture GPU, appelée Parallel Simulated Annealing (PSA), a été proposée.

Un benchmark de workflows a été généré aléatoirement afin de valider notre approche, et une étude comparative entre les performances de notre approche et celle de l'algorithme HEFT a été menée.

Les résultats obtenus de l'étude comparative montrent clairement que notre algorithme proposé surpasse l'algorithme HEFT qui est considéré comme le standard de l'industrie et qu'il peut être utilisé efficacement dans un environnement de cloud computing pour ordonnancer efficacement les tâches de workflow en minimisant leur temps d'exécution.

Dans nos travaux futurs, nous avons l'intention d'améliorer notre algorithme afin de prendre en charge d'autres métriques de qualité de service, telles que les contraintes budgétaires et la consommation d'énergie à travers la technique de la variation dynamique de la tension et de la fréquence des processeurs (DVFS) et donc de passer, sur le plan formel, d'un problème d'optimisation mono-objective à un problème d'optimisation multi-objective.

Passage de la simulation à l'ordonnancement réel en incorporant notre algorithme dans un système de gestion de workflow tel OpenNebula (*OpenN*, 2020).

Dans cette thèse, nous avons traité uniquement l'ordonnancement statique des workflows L'ordonnancement dynamique des workflows en prenant en compte l'état de l'infrastructure du cloud et cela grâce à un feedback en temps réel est une piste de recherche à explorer.

# Bibliographie


Alba, E. (2005). *Parallel metaheuristics : A new class of algorithms* (Vol. 47). John Wiley & Sons.

*Amazon EC2*. (2020, juin). http://aws.amazon.com/ec2/

Arabnejad, H., & Barbosa, J. G. (2014). List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table. *IEEE Transactions on Parallel and Distributed Systems*, *25*(3), 682‑694. https://doi.org/10.1109/TPDS.2013.57

Avriel, M. (1976). Nonlinear Programming : Analysis and Methods. Printice-Hall. *Inc, Englewood Cliffs, New Jersey*.

*AWS*. (2020, juin). Amazon Web Services. aws.amazon.com

Benhammouda, M., & Malki, M. (2019). A GPU Based Approach for Solving the Workflow Scheduling Problem. *International Journal of Information Retrieval Research (IJIRR)*, *9*(4), 1–12.

Blazewicz, J., Domschke, W., & Pesch, E. (1996). *The job shop scheduling problem : Conventional and new solution techniques*.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., & Weglarz, J. (2007). *Handbook on scheduling : From theory to applications*. Springer Science & Business Media.

Branke, J. (2002). Optimization in dynamic environments. In *Evolutionary Optimization in Dynamic Environments* (p. 13–29). Springer.

Brucker, P. (2007). *Scheduling algorithms* (5th ed). Springer.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., & Brandic, I. (2009). Cloud computing and emerging IT platforms : Vision, hype, and reality for

delivering computing as the 5th utility. *Future Generation Computer Systems*, *25*(6), 599‑616. https://doi.org/10.1016/j.future.2008.12.001

Cavanaugh, C. D., Welch, L. R., Shirazi, B. A., & Anwar, S. (2000). Quality of service negotiation for distributed, dynamic real-time systems. *International Parallel and Distributed Processing Symposium*, 757--765.

Černỳ, V. (1985). Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm. *Journal of optimization theory and applications*, *45*(1), 41–51.

Collette, Y., & Siarry, P. (2002). *Optimisation multiobjectif*. Editions Eyrolles.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., Hanson, M. J., & McNamee, D. J. (1987). CS702 : Advanced Algorithms Analysis & Design. *Journal ofthe Association for Computing Machinery*, *34*(3), 596–615.

Crainic, T. G., & Laporte, G. (2012). *Fleet management and logistics*. Springer Science & Business Media.

Dahal, K., Tan, K. C., & Cowling, P. I. (2007). *Evolutionary scheduling* (Vol. 49). Springer Science & Business Media.

De Oliveira, D., OcañA, K. A., Ogasawara, E., Dias, J., GonçAlves, J., BaiãO, F., & Mattoso, M. (2013). Performance evaluation of parallel strategies in public clouds : A study with phylogenomic workflows. *Future Generation Computer Systems*, *29*(7), 1816–1825.

De Oliveira, D., Ogasawara, E., Baião, F., & Mattoso, M. (2010). Scicumulus : A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. *2010 IEEE 3rd International Conference on Cloud Computing*, 378–385.

Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., & Koranda, S. (2003). *Mapping Abstract Complex Workflows onto Grid Environments*. 15.

Deelman, E., Singh, G., Livny, M., Berriman, B., & Good, J. (2008). The cost of doing science on the cloud : The montage example. *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, 1–12.

*Eucal*. (2020, juin). Eucalyptus. https://www.eucalyptus.cloud/

Feitelson, D. G., & Rudolph, L. (1996). Toward convergence in job schedulers for parallel supercomputers. *Workshop on Job Scheduling Strategies for Parallel Processing*, 1–26.

Femmam, M., Kazar, O., Kahloul, L., & Fareh, M. E.-K. (2018). Labelled evolutionary Petri nets/genetic algorithm based approach for workflow scheduling in cloud computing. *International Journal of Grid and Utility Computing*, *9*(2), 157–169.

Flynn, M. J. (1966). Very high-speed computing systems. *Proceedings of the IEEE*, *54*(12), 1901–1909.

Foster, I., Zhao, Y., Raicu, I., & Lu, S. (2008). Cloud computing and grid computing 360-degree compared. *arXiv preprint arXiv:0901.0131*.

Glover, F. (1986). Future paths for integer programming and links to ar tifi cial intelli g en ce. *Computers operations research*, *13*(5), 533–549.

Goldberg, D. E., Richardson, J., & others. (1987). Genetic algorithms with sharing for multimodal function optimization. *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, 41–49.

Greening, D. R. (1990). Parallel simulated annealing techniques. *Physica D: Nonlinear Phenomena*, *42*(1‑3), 293‑306. https://doi.org/10.1016/0167-2789(90)90084-3

Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., & Good, J. (2008). On the use of cloud computing for scientific workflows. *2008 IEEE fourth international conference on eScience*, 640–645.

Hwu, W., Keutzer, K., & Mattson, T. G. (2008). The concurrency challenge. *IEEE Design & Test of Computers*, *25*(4), 312–320.

*IBMC*. (2020, juin). IBM CLOUD. www.ibm.com/cloud

Ismayilov, G., & Topcuoglu, H. R. (2020). Neural network based multi-objective evolutionary algorithm for dynamic workflow scheduling in cloud computing. *Future Generation Computer Systems*, *102*, 307‑322. https://doi.org/10.1016/j.future.2019.08.012

Jacob, J. C., Katz, D. S., Berriman, G. B., Good, J., Laity, A. C., Deelman, E., Kesselman, C., Singh, G., Su, M.-H., Prince, T. A., & others. (2010). Montage : A grid portal and software toolkit for science-grade astronomical image mosaicking. *arXiv preprint arXiv:1005.4454*.

Kirk, D. B., & Wen-Mei, W. H. (2016). *Programming massively parallel processors : A hands-on approach*. Morgan kaufmann.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, *220*(4598), 671–680.

Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1982). Recent developments in deterministic sequencing and scheduling : A survey. In *Deterministic and stochastic scheduling* (p. 35–73). Springer.

Lin, C., Lu, S., Fei, X., Chebotko, A., Pai, D., Lai, Z., Fotouhi, F., & Hua, J. (2009). A reference architecture for scientific workflow management systems and the VIEW SOA solution. *IEEE Transactions on Services Computing*, *2*(1), 79–92.

Lopez, I. C. (2016). *Scientific Workflow Scheduling for Cloud Computing Environments* [Doctor of Philosophy Ph.D., University of Sydney; Faculty of Engineering & Information Technologies; Center for Distributed & High Performance Computing]. http://hdl.handle.net/2123/16769

Mangala, N., Ch, J., Shashi, S., Subrata, C., & others. (2012). Galaxy workflow integration on garuda grid. *2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 194–196.

Mell, P., & Grance, T. (2011). *The NIST Definition of Cloud Computing*. 7.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, *21*(6), 1087–1092.

*MicAz*. (2020, juin). Microsoft Azure. https://azure.microsoft.com/en-us/

Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *The computer journal*, *7*(4), 308–313.

*Nimb*. (2020, juin). Nimbus. http://www.nimbusproject.org/,

Nocedal, J., & Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media.

*NVIDIA*. (2020, octobre). NVIDIA. https://www.nvidia.com/en-me/geforce/graphics-cards/30-series/rtx-3090/

Ocaña, K. A., de Oliveira, D., Dias, J., Ogasawara, E., & Mattoso, M. (2012). Discovering drug targets for neglected diseases using a pharmacophylogenomic cloud workflow. *2012 IEEE 8th International Conference on E-Science*, 1–8.

Ocaña, K. A., de Oliveira, D., Horta, F., Dias, J., Ogasawara, E., & Mattoso, M. (2012). Exploring molecular evolution reconstruction using a parallel cloud based scientific workflow. *Brazilian Symposium on Bioinformatics*, 179–191.

Ocaña, K. A., de Oliveira, D., Ogasawara, E., Dávila, A. M., Lima, A. A., & Mattoso, M. (2011). SciPhy : A cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes. *Brazilian Symposium on Bioinformatics*, 66–70.

Ogasawara, E., Dias, J., Silva, V., Chirigati, F., de Oliveira, D., Porto, F., Valduriez, P., & Mattoso, M. (2013). Chiron : A parallel engine for algebraic scientific workflows. *Concurrency and Computation: Practice and Experience*, *25*(16), 2327–2341.

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M. R., Wipat, A., & Li, P. (2004). Taverna : A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, *20*(17), 3045‑3054. https://doi.org/10.1093/bioinformatics/bth361

*OpenN*. (2020, juin). OpenNebula. http://www.opennebula.org

Plummer, D. C., Cearley, D. W., & Smith, D. M. (2008). Cloud computing confusion leads to opportunity. *Gartner Report*.

Rodriguez, M. A., & Buyya, R. (2017). Scientific workflow management system for clouds. In *Software Architecture for Big Data and the Cloud* (p. 367–387). Elsevier.

Schrijver, A. (1998). *Theory of linear and integer programming*. John Wiley & Sons.

Sharma, C., & Rashid, M. (2020). Scheduling of Scientific Workflow in Distributed Cloud Environment Using Hybrid PSO Algorithm. In *Trends in Cloud-based IoT* (p. 113–123). Springer.

Suter, F., & Hunold, S. (2013). *DAGGEN: A synthetic task graph generator*. www.github.com/frs69wq/daggen

Sutter, H., & Larus, J. (2005). Software and the concurrency revolution. *Queue*, *3*(7), 54–62.

Topcuoglu, H., Hariri, S., & Min-You Wu. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, *13*(3), 260‑274. https://doi.org/10.1109/71.993206

Verma, A., & Kaushal, S. (2015). Cost Minimized PSO based Workflow Scheduling Plan for Cloud Computing. *International Journal of Information Technology and Computer Science*, *7*(8), 37‑43. https://doi.org/10.5815/ijitcs.2015.08.06

Wei-Neng Chen, & Jun Zhang. (2009). An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *39*(1), 29‑43. https://doi.org/10.1109/TSMCC.2008.2001722

WfMC, W. M. C. (1999). Workflow management coalition terminology and glossary. *Workflow Management Coalition*.

Wu, Z., Liu, X., Ni, Z., Yuan, D., & Yang, Y. (2013). A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, *63*(1), 256–293.

Yassa, S. (2014). *Allocation optimale multicontraintes des workflows aux ressources d'un environnement Cloud Computing* [PhD Thesis]. Cergy-Pontoise.

Yassa, S., Sublime, J., Chelouah, R., Kadima, H., Jo, G. S., & Granado, B. (2013). A genetic algorithm for multi-objective optimisation in workflow scheduling with hard constraints. *International Journal of Metaheuristics*, *2*(4), 415. https://doi.org/10.1504/IJMHEUR.2013.058475

Yu, J., & Buyya, R. (2006). A budget constrained scheduling of workflow applications on utility Grids using genetic algorithms. *2006 Workshop on Workflows in Support of Large-Scale Science*, 1‑10. https://doi.org/10.1109/WORKS.2006.5282330

Yu, J., & Buyya, R. (2004). A novel architecture for realizing grid workflow using tuple spaces. *Fifth IEEE/ACM International Workshop on Grid Computing*, 119–128.

Zhou, X., Zhang, G., Sun, J., Zhou, J., Wei, T., & Hu, S. (2019). Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT. *Future Generation Computer Systems*, *93*, 278‑289. https://doi.org/10.1016/j.future.2018.10.046