République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option: Réseaux et Systèmes Distribués (RSD)

Thème

Ordonnancement des tâches multi-objectif dans le Cloud computing

Réalisé par :

- MALTI Arslan Nedhir
- KHEDDAOUI Med Badreddine

Présenté le 24 Juin 2020 devant le jury composé de :

- Mr BENMOUNA Youcef — Président
- Mr BELHOCINE Amin — Examinateur
- Mr BENMAMMAR Badr — Encadrant
- Mr BOUAFIA Zoheir — Co-Encadrant

Année universitaire: 2019-2020

Remerciement

En premier lieu, Nous remercions ALLAH le tout puissant qui nous a aidé et nous a donné le courage, la patience, la force et la volonté pour la réalisation de ce travail.

Nous tenons à exprimer notre profonde gratitude et nos sincères remerciements à notre encadrant Mr BENMAMMAR Badr ainsi qu'à notre Co-encadrant Mr BOUAFIA Zoheir pour leurs temps plus précieux qu'ils nous ont prêté, leurs soutiens et aussi pour leurs précieuses directives, conseils, orientations, encouragements tout au long de ce travail.

Nos remerciements s'adressent également aux membres du jury Mr BENMOUNA Youcef et Mr BELHOCINE Amin pour l'immense honneur qu'ils nous ont fait en acceptant d'examiner et d'évaluer ce travail et pour le temps qu'ils ont consacré pour la lecture de ce mémoire.

Nos plus vifs remerciements à nos professeurs qui ont contribué à notre formation le long de nos cursus universitaires et qui ont bien voulu par leur grande générosité partager leur savoir avec leurs étudiants.

Ce travail n'aurait pas été possible sans le soutien de nos familles, nous tenons à remercier nos parents ainsi que nos proches pour tous les encouragements à mener à bien ce travail.

Merci à tous !

Dédicace

Je dédie ce modeste travail

A mes chers parents pour leur patience, leur amour, leur soutien et leurs encouragements,

A mon frère et ma sœur,

A ma grand-mère et tous mes proches,

A mon binôme Badreddine Mohammed KHADDAOUI,

A tous mes amies.

Arslan Nedhir

Table des matières

Liste des figures

Liste des tables

Liste des abréviations

Acronyme	Signification
API	Application Program Interface
BA	Bat Algorithm
CPU	Central Processing Unit
CS	Cuckoo Search
DC	Data Center
FPA	Flower Pollination Algorithm
IDE	Integrated Development Environment
IHM	Interactions Homme-Machine
JVM	Java Virtual Machine
LGPL	Lesser General Public Licence
MCDM	Multi-Criteria Decision Making
MIPS	Million Instructions Per Second
NP	Nondeterministic Polynomial Time
PE	Processing Element
QoS	Quality Of Service
RAM	Random Access Memory
SP	Service Provider
SLA	Service Level Agreement
TOPSIS	Technic for Order Performance by Similarity to Ideal Solution
VM	Virtual Machine
VPN	Virtual Private Network
WPM	Weighted Product Model
WSM	Weight Sum Method

Résumé :

L’ordonnancement de tâches est un enjeu important qui influence considérablement les performances de l'environnement de Cloud computing. Les fournisseurs et les utilisateurs des services Cloud ont des objectifs et des exigences conflictuels. Un bon ordonnanceur doit offrir un compromis acceptable entre ces objectifs. Ainsi, l’ordonnancement de tâches dans le Cloud devient un problème d’optimisation multi objectif. Notre contribution dans le cadre de ce travail consiste à traiter le problème de l'ordonnancement des tâches dans le Cloud computing afin de trouver la meilleure affectation des tâches à l’ensemble des machines virtuelles. La métaheuristique FPA a été utilisée en évaluant sa fonction objectif avec trois métriques de QoS qui sont, le makespan, le cout et la fiabilité. Les simulations et les évaluations ont été réalisées à l’aide de CloudSim et les résultats obtenus sont très satisfaisants.

Mots clés: Cloud computing, Optimisation multi-objectif, QoS, FPA, CloudSim.

Abstract:

Task scheduling is an important factor that greatly influences the performance of the Cloud computing environment. Cloud service providers and users have conflicting objectives and requirements. A good scheduler must offer an acceptable compromise between these objectives. Thus, the Cloud tasks scheduling becomes a multi-objective optimization problem. Our contribution through this work consists in dealing with the problem of task scheduling in Cloud computing in order to find the best assignment of tasks to all virtual machines. The FPA metaheuristics was used by evaluating its objective function with three QoS metrics which are, the makespan, the cost and the reliability. The simulations and evaluations were carried out using CloudSim and the results obtained are very satisfactory.

Key words: Cloud computing, Multi-objective optimization, QoS, FPA, CloudSim.

ملخص:

جدولة المهام هي عامل مهم تؤثر بشكل كبير على أداء بيئة الحوسبة السحابية. لدى مقدمي الخدمات السحابية والمستخدمين أهداف ومتطلبات متضاربة. يجب على المجدول الجيد تنفيذ حل وسط مقبول بين هذه الأهداف. وبالتالي، تصبح جدولة المهام في السحابة مشكلة تحسين متعدد الأهداف. الهدف من هذا العمل هو معالجة مشكلة جدولة المهام في الحوسبة السحابية من أجل العثور على أفضل تعيين مهام للأجهزة الافتراضية باستخدام FPA métaheuristique المطبق على مقاييس جودة الخدمة الثلاثة التالية: وقت التنفيذ والتكلفة والموثوقية. نفذت عمليات المحاكاة والتقييم باستخدام CloudSim والنتائج التي تم الحصول عليها مرضية للغاية.

الكلمات المفتاحية: الحوسبة السحابية، التحسين متعدد الأهداف، مقاييس جودة الخدمة، FPA، CloudSim.

Introduction générale

Le Cloud computing est une technologie moderne dans le domaine informatique. Il est apparu avec l'augmentation de la puissance des serveurs et la baisse de leurs couts de fonctionnement ainsi qu'avec l'apparition de l'Internet haut débit. Il permet de fournir plusieurs services flexibles et évolutifs permettant l'accès au réseau à la demande. Les ressources sont partagées et la puissance de calcul est configurable par rapport aux besoins des utilisateurs. Le Cloud computing se caractérise par sa souplesse et par ses avantages parmi lesquels, l'automatisation de la maintenance des applications [1].

Dans le domaine du Cloud computing, l'ordonnancement des tâches est un enjeu important qui influence considérablement les performances de cet environnement. Les fournisseurs et les utilisateurs de services Cloud ont des exigences et des objectifs conflictuels. Un bon ordonnanceur doit mettre en œuvre un compromis acceptable entre ces objectifs. Ainsi, l'ordonnancement de tâches dans le Cloud computing devient un problème d'optimisation multi-objectif. C'est un problème NP-complet qui nécessite d'utiliser des métaheuristiques pour sa résolution.

Notre contribution dans le cadre de ce projet de fin d'études consiste à utiliser un algorithme pour l'optimisation de plusieurs objectifs dans le Cloud computing. Cet algorithme est basé sur la métaheuristique FPA (Flower Pollination Algorithm). La problématique traitée concerne l'ordonnancement de tâches multi-objectif dans le Cloud computing par rapport à 3 critères qui sont le coût, la fiabilité et le makespan.

Le simulateur CloudSim a été utilisé dans le cadre de ce projet de fin d'études et les performances de notre algorithme ont été évaluées par rapport aux Round Robin, Max Min et Min Min pour le mono objectif (makespan). Les résultats obtenus sont très satisfaisants.

Nous avons comparé également deux versions de la fonction objectif de l'algorithme FPA à savoir la somme pondérée et le principe de Pareto en se basant sur l'algorithme TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution). Ce dernier permet de choisir la solution optimale parmi celles obtenues par le front Pareto. Les résultats de comparaison sont très intéressants.

Le reste de ce rapport est organisé comme suit :

Le premier chapitre sera consacré à la présentation du Cloud computing qui est le contexte de travail de notre projet.

Le deuxième chapitre sera dédié à la présentation des métaheuristiques en se focalisant en particulier sur l'algorithme FPA qui a été utilisé dans la partie réalisation de ce travail.

Enfin, notre contribution est détaillée dans le troisième chapitre. Ce dernier présente le simulateur CloudSim, l'IHM réalisée et surtout les résultats des différentes simulations effectuées. Les graphes de comparaison entre FPA, Round Robin, Max Min et Min Min sont détaillés dans ce chapitre ainsi que la comparaison entre Pareto/TOPSIS et la somme pondérée pour la fonction objectif de FPA.

CHAPITRE 1:
Cloud computing

1.1 Introduction

Le concept de Cloud computing consiste à offrir des ressources informatiques (serveurs, stockage, réseau, applications…) en tant que services à la demande sur un réseau. Il s'agit d'une notion qui est très utilisée aujourd'hui à l'exemple de Gmail ou de la sauvegarde des données sur Internet en général. Le Cloud computing offre aussi la possibilité d'utiliser une puissance de calcul et une capacité de stockage offerte par des serveurs distants [1].

L'objectif de ce premier chapitre est de présenter les principales notions liées au Cloud computing : sa définition, ses caractéristiques, les conditions qui encouragent à l'utiliser ainsi que les problèmes qui se posent avec son utilisation.

1.2 Historique du Cloud computing

Dans les débuts des années 1960 et 1970, les entreprises avaient de grands ordinateurs centraux qui fournissaient des services aux travailleurs qui avaient accès à eux par des terminaux mais ces ordinateurs centraux coutaient très chers.

Dans les années 1980, les entreprises ont réalisé que les serveurs basés sur des ordinateurs ordinaires pourraient être installés à moindre coût que les ordinateurs centraux en donnant aux utilisateurs plus de contrôle sur leurs actions.

Dans les années 1990, avec le début de l'utilisation globale d'Internet, l'accès à des serveurs centraux est revenu, des serveurs web avec beaucoup de puissance étaient nécessaires pour tenir les demandes des usagers.

Depuis cette époque, jusqu'aujourd'hui de plus en plus de services sont offerts sur Internet et de plus en plus de stockage est nécessaire pour les utilisateurs de ces services.

Avec l'augmentation de la vitesse d'accès au réseau et de la bande passante, il est devenu logique d'héberger les applications sur les serveurs à travers Internet.

Aujourd'hui, le calcul massif nécessaire peut être résolu par les fournisseurs qui sont dédiés à cela, de cette manière plusieurs utilisateurs peuvent partager la même infrastructure maximisant l'efficacité de celui-ci et en minimisant les coûts.

A la fin des années 1990, tous les Data Center utilisaient moins de 10% de leurs capacités parce qu'ils voulaient réserver le reste pour des pics occasionnels tel qu'en période de fêtes. A l'époque, Amazon a fait un grand effort pour résoudre ce problème en louant ses serveurs à la demande et en proposant à ses clients ses outils S3 (Simple Storage Service) et EC2 (Elastic Compute Cloud) offrant respectivement des services de stockage de données et de calcul. Amazon a pu rentabiliser ses propres investissements en matériel informatiques, et c'était l'un des concepts du Cloud computing [2].

1.3 Notions de base

1.3.1 Définition

Le Cloud computing fournit un ensemble de matérielle (serveurs, réseaux) et de logicielle à la demande. Il permet d'accéder rapidement à des ressources informatiques n'importe où dans le monde et avec n'importe quel appareille connecté. Il s'agit d'une technologie qui permet de gérer la puissance de calcul et le stockage à travers des serveurs distants. Le Cloud computing est aussi caractérisé par sa souplesse permettant aux fournisseurs d'adapter une capacité de stockage, et une puissance de calcul selon les besoins des utilisateurs [1].

Le Cloud computing est basé sur la consommation des ressources, des matérielles et des logiciels. Des services qui sont gratuits ou payants selon l'utilisation et sont caractérisés par ce qui suit [1] :

- Une demande libre de service.
- Un accès en diffusé via un réseau.
- La mise en commun des ressources.
- L'élasticité rapide.
- Un service mesuré.

1.3.2 Avantages et inconvénients

1.3.2.1 Avantages

Le Cloud computing présente les avantages suivants [3] :

- **Sauvegarde et restauration** : toutes les données sont stockées dans le Cloud, ce qui rend la sauvegarde et la restauration relativement plus facile, de plus les fournisseurs de services Cloud sont assez compétents pour gérer la récupération des informations.
- **Mises à jour logicielles automatiques** : le fournisseur décharge son client de toute responsabilité de maintenance.
- **Partage de données** : les collègues d'une entreprise peuvent synchroniser et travailler ensemble sur des documents ou applications partagées.
- **Travailler partout** : avec le Cloud computing, il suffit d'avoir une bonne connexion Internet pour être au travail.

1.3.2.2 Inconvénients

Malgré ses avantages, le Cloud computing présente quelques inconvénients [3,4] :

- **Besoin d'une connexion Internet de haut débit** : le Cloud utilise de manière intensive le transfert de données, pour bien profiter il faut une excellente connexion Internet qui peut revenir excessivement chère.

- **Le coût du Cloud** : plus une entreprise est grande plus ses ressources et besoins sont grands, il est donc parfois plus judicieux de construire son propre Cloud plutôt que de louer les services d'un Cloud externe qui va coûter une fortune en consommation.

- **La pérennité (continuité) de service** : l'arrêt d'activité du fournisseur Cloud peut poser un grand problème à l'entreprise et changer d'hébergeur peut prendre du temps.

- **Sécurité** : les risques d'attaque et de perte de confidentialité sont augmentés dans le cas des Clouds publics car la connexion aux serveurs applicatifs passe par le réseau internet et donc l'entreprise s'expose à des risques supplémentaires.

- **Le cadre légal** : les données transférées dans le Cloud ne sont pas forcément présentes sur le territoire national ; elles peuvent l'être, comme elles peuvent être dans un autre pays. Par conséquent, sauf mention contraire du prestataire de service, on ne sait pas précisément à quel endroit sont stockées les données. De plus, on n'a aucun accès physique à ces données [4].

1.3.3 Virtualisation

La virtualisation est un mécanisme informatique qui consiste à faire fonctionner plusieurs systèmes, serveurs ou applications, sur un même serveur physique. Il s'agit d'un composant technique clé dans le Cloud computing [5].

La virtualisation est utilisée pour plusieurs raisons, elle fournit une flexibilité, une plus grande utilisation des ressources sachant qu'un serveur qui tourne à 90% de ses capacités ne consomme pas beaucoup plus d'énergie qu'un serveur qui en consomme que 10%, de permettre aux services hétérogènes de coexister sur le même matériel physique et enfin d'assurer une gestion plus facile des ressources.

La virtualisation repose sur ce qui suit [6] :

- Un système d'exploitation principal (appelé "système hôte") est installé sur un serveur physique unique. Ce système sert d'accueil à d'autres systèmes d'exploitation.

- Un logiciel de virtualisation est installé sur le système d'exploitation principal. Il permet de créer les environnements clos sur lesquels seront installés d'autres systèmes d'exploitation (les systèmes invités). Ces environnements sont appelés les machines virtuelles.

- Un système invité est installé dans une machine virtuelle qui fonctionne indépendamment des autres systèmes invités qui sont installés dans les autres machines virtuelles. Chaque

machine virtuelle dispose d'un accès aux ressources du serveur physique (mémoire, espace disque…).

La figure 1.1 représente une comparaison entre l'architecture traditionnelle et l'architecture virtuelle.

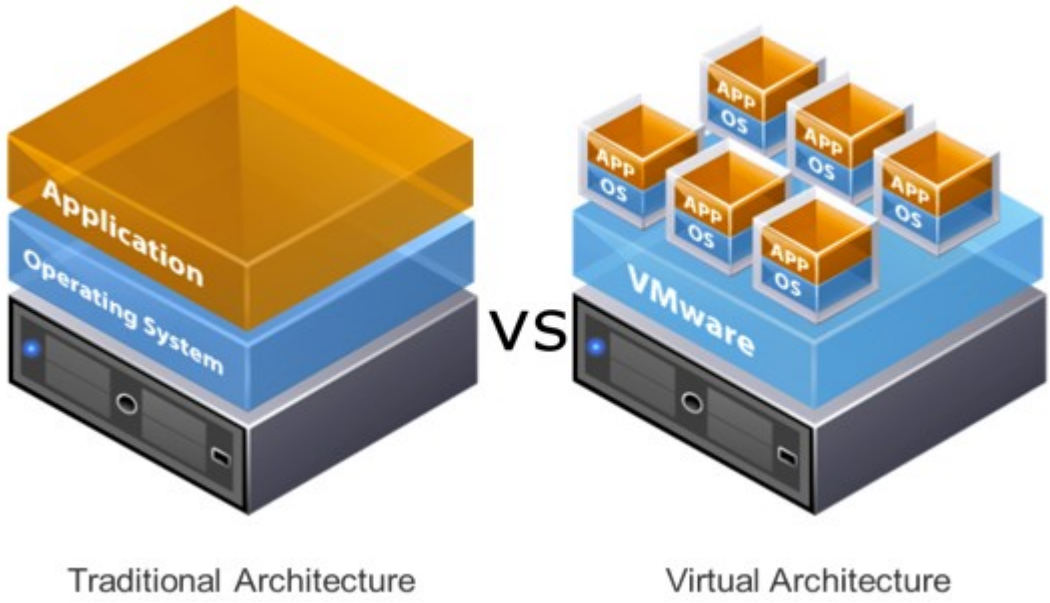


Figure 1.1 : Architecture traditionnelle vs Architecture virtuelle [7].

1.3.4 DataCenter

Un DataCenter ou centre de données, est une infrastructure composée d'un réseau d'ordinateurs et d'espaces de stockage. Cette infrastructure peut être utilisée par les entreprises pour organiser, traiter, stocker et entreposer de grandes quantités de données.

Un DataCenter est un ensemble d'éléments qui regroupe des serveurs, des sous-systèmes de stockage, des commutateurs de réseau, des routeurs et des firewalls [8].

1.3.5 Service provider

Le fournisseur de services (service provider ou SP) est une entreprise qui fournit des services au client, du matérielle virtuelle, des logicielle et d'autre service loués et gérés par le fournisseur basées sur un accord de niveau de service (Service Level Agreement ou SLA) et une négociation entre le fournisseur de services et le consommateur [1].

1.3.6 SLA

Le SLA est un contrat de niveau de service entre un fournisseur de services Cloud et un client qui garantit le maintien d'un niveau de service minimal. Il garantit des niveaux de fiabilité, de disponibilité et de réactivité aux systèmes et aux applications, tout en spécifiant qui gouvernera en cas d'interruption de service. Autrement dit, il s'agit d'une clause contractuelle qui définit les objectifs

précis et le niveau de service qu'est en droit d'attendre un client de la part du prestataire signataire [1].

Le SLA permet de garantir aux clients certains niveaux de sécurité dans le stockage et la gestion de leurs données à caractère personnel.

1.3.7 Consolidation

La consolidation survient généralement pendant les fusions et les acquisitions d'entreprises. Lorsque l'entreprise majoritaire n'a pas besoin des centres de données utilisées par les firmes qu'elles rachètent, donc l'entreprise avec plusieurs Data Center peut choisir de les consolider, en réduisant leur nombre pour minimiser les coûts des opérations [8].

1.3.8 Machine virtuelle

Une machine virtuelle (Virtual Machine ou VM) est un environnement d'application ou de système d'exploitation installé sur un logiciel qui imite un matériel dédié. Côté utilisateur final, l'interaction avec une machine virtuelle est la même qu'avec un matériel dédié [6].

1.3.9 Broker

Le Broker est l'intermédiaire entre les VMs et les Cloudlets (les tâches). Il est le responsable de la médiation des négociations entre l'utilisateur et les prestataires de service selon les conditions de qualité de service (QoS) des utilisateurs.

1.4 Types de Cloud computing

Les entreprises ont le choix de louer ou construire un Cloud privé, souscrire à des offres d'un Cloud public ou de choisir un Cloud hybride qui est une composition de deux ou plusieurs Clouds (privé ou public).

1.4.1 Cloud privé

Un Cloud privé est utilisé par un groupe ou une organisation spécifique avec un accès limité, il peut être géré par l'entreprise utilisatrice elle-même.

Le Cloud privé peut se déployer sous deux formes distinctes [9]:

- **Cloud privé interne:** hébergé par l'entreprise elle-même, parfois partagé ou mutualisé en mode privatif avec les filiales.
- **Cloud privé externe:** hébergé chez un tiers, il est entièrement dédié à l'entreprise et accessible via des réseaux sécurisés de type VPN.

1.4.2 Cloud public

Un Cloud public repose sur le modèle standard de Cloud computing dans lequel le fournisseur de services rend des ressources accessibles à tout utilisateur qui a une connexion Internet. Il offre à ses utilisateurs l'évolutivité, la flexibilité et aussi des options supplémentaires de contrôle. Les services de Cloud public peuvent être gratuits ou facturés à l'utilisation [1].

1.4.3 Cloud hybride

Le Cloud hybride est une combinaison de Clouds publics et privés. Une entreprise peut avoir une partie de ses services dans sa propre infrastructure, mais aussi dans les Clouds public. Ou peut utiliser le Cloud public juste au moment où il y a des pics d'utilisation. C'est aussi une bonne option pour les entreprises voulant protéger ses données critiques en locale sans les exposer en ne voulant pas trop investir dans les infrastructures [10].

Le Cloud hybride est donc un bon compromis entre les Clouds publics et les Clouds privés.

1.5 Architecture de Cloud computing

IaaS, PaaS et SaaS sont actuellement les 3 principaux modèles de services qui s'offrent aux entreprises désireuses de s'orienter vers le Cloud computing.

1.5.1 Infrastructure as a Service

Infrastructure As A Service (IaaS) signifie l'externalisation de l'infrastructure matérielle du service informatique (réseaux, stockage et serveurs) chez un fournisseur tiers. Les ressources informatiques sont hébergées sur des serveurs externes et les utilisateurs peuvent y accédez via une connexion Internet. Ils conservent à leurs charges la gestion des systèmes d'exploitation et du middleware : bases de données, intégration, environnement d'exécution, ainsi que les applications [1].

- **Avantages** : grande flexibilité, contrôle total des systèmes (administration à distance), et donc ainsi permettre d'installer tout type de logiciel métier [11].
- **Inconvénient** : besoin d'administrateurs système comme pour les solutions de serveurs classiques sur site [11].

1.5.2 Platform as a Service

Platform as a Service (PaaS) permet aux utilisateurs de louer des serveurs virtualisés pour exécuter ou développer ses propres applications à l'aide du kit de développement fourni par le Cloud computing. Le PaaS facilite à travers des outils et des services le flux de travail lors de la conception, du développement, du test, du déploiement et de l'hébergement d'applications [1].

- **Avantages** : le déploiement est automatisé, pas de logiciel supplémentaire à acheter ou à installer [11].

- **Inconvénients** : limitation à une ou deux technologies. Pas de contrôle de machines virtuelles sous-jacentes [11].

1.5.3 Software as a Service

Software as a Service (Saas) représente des applications complètes et offertes en tant que service à la demande, les utilisateurs peuvent exécuter ces applications à l'aide d'un navigateur web via une interface, abstrait totalement sur le matériel et le logiciel que l'utilisateur utilise et à partir de là, il a accès à certaines informations et fonctionnalités [12].

- **Avantages** : plus d'installation, plus de mise à jour (elles sont continuées chez le fournisseur), plus de migration de données etc. Paiement à l'usage. Test de nouveaux logiciels avec facilité [11].

- **Inconvénients :** limitation par définition au logiciel proposé. Pas de contrôle sur le stockage et la sécurisation des données associées au logiciel [11].

Nous pouvons dire finalement qu'avec le SaaS on utilise une application, qu'avec le PaaS on la développe et enfin avec l'IaaS on l'héberge.

La figure 1.2 représente les différentes architectures du Cloud computing de l'architecture la moins visible à la plus visible pour les utilisateurs.

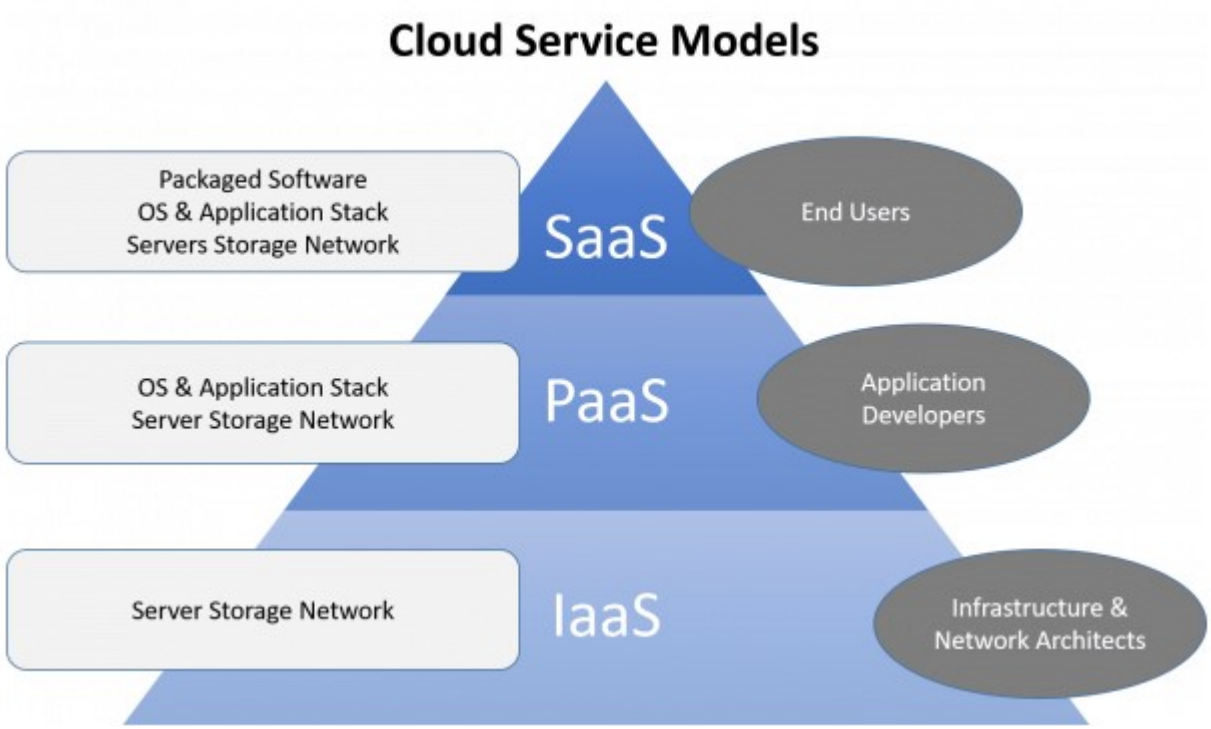


Figure 1.2 : Les trois couches du Cloud computing [13].

1.6 Conclusion

Ce chapitre a été consacré à la présentation du Cloud computing, le domaine qui représente le contexte de travail de notre projet de fin d'études. Nous avons présenté donc les notions fondamentales et nécessaires à la compréhension de ce domaine, en particulier son architecture, ses types, ses avantages et ses inconvénients mais également quelques mots clé relatifs à sa compréhension comme les VMs, le SLA, le Datacenter, la virtualisation, … etc.

Le deuxième chapitre de notre manuscrit sera consacré à la présentation des métaheuristiques.

CHAPITRE 2:
Les métaheuristiques

2.1 Introduction

De nombreux résultats dans la littérature indiquent que les méthodes de résolution classique sont incapables de trouver une solution exacte pour des problèmes difficiles puisqu'il n'existe pas d'algorithmes qui ont un temps d'exécution polynomial pour ce type de problèmes. Les algorithmes existants permettant d'aboutir à une solution optimale ont une complexité très élevée, et plus la taille de données augmente pour ce problème, plus le temps nécessaire pour le résoudre croît exponentiellement.

Pour ce type de problèmes, nous pouvons affirmer que la recherche d'une solution exacte est donc impossible. Pour cela, diverses études ont été menées pour développer de nouvelles méthodes qui utilisent des algorithmes fournissant des solutions proches de l'optimalité. La recherche s'est d'abord orientée vers des heuristiques spécifiques aux problèmes, puis par des méthodes plus générales appelées métaheuristiques. Ces derniers sont devenus un élément important de l'optimisation moderne.

Dans ce chapitre, nous commençons par donner des notions fondamentales sur les méthodes de résolutions. Nous présentons par la suite la classification générale des méthodes d'optimisation combinatoire. Nous présenterons également quelques métaheuristiques en s'intéressant à un algorithme en particulier. Il s'agit de l'algorithme de pollinisation des fleurs (Flower Pollination Algorithm (FPA) en anglais), un algorithme inspiré du comportement coopératif de certaines espèces dans leur milieu naturel.

2.2 Les méthodes de résolution

Dans la littérature, il existe essentiellement deux classes principales pour la résolution de problèmes : les méthodes exactes et les méthodes approchées.

2.2.1 Les méthodes exactes

Cette catégorie est une approche de résolution qui assure de trouver une solution exacte et de prouver son optimalité pour toutes les instances d'un problème d'optimisation combinatoire en se basant sur une recherche complète de l'espace des combinaisons. Par contre, le temps nécessaire pour trouver la solution optimale du problème augmente généralement avec l'augmentation de la taille du problème. De façon pratique, ces méthodes sont dédiées à la plupart du temps aux problèmes de petite échelle ou de complexité limitée.

2.2.2 Les méthodes approchées

Les méthodes exactes sont parfois inappropriées dans certains problèmes pratiques, où nous devons trouver une solution optimale à un problème donné sous des contraintes qui peuvent être très complexes. Cette tâche est très difficile ou même parfois impossible. Pour cela, Il est nécessaire de trouver d'autre méthodes de résolution qui permettent d'aboutir à des solutions satisfaisables dans un temps de calcul raisonnable ; des solutions non optimales mais de bonne qualité. Pour cette raison, les méthodes approchées sont utilisées pour fournir une bonne solution réalisable dans un temps satisfaisant. Ces méthodes n'offrent aucune garantie quant à l'optimalité

de la meilleure solution trouvée, mais qui est néanmoins suffisante pour atteindre un objectif immédiat.

L'avantage principale de ces méthodes est qu'elles peuvent être utilisées pour accélérer le processus de recherche lorsque la recherche d'une solution optimale est impossible ou il est difficile de trouver un optimum global, en plus elles peuvent s'appliquer à n'importe quelle classe de problèmes faciles ou très difficile (NP-difficile). Les méthodes approchées englobent deux classes, heuristiques et méta-heuristiques. Les métaheuristiques sont aussi classées en deux sous catégories : les métaheuristiques à solution unique et les métaheuristiques à base de population.

La figure 2.1 donne un aperçu global sur les méthodes de résolution.

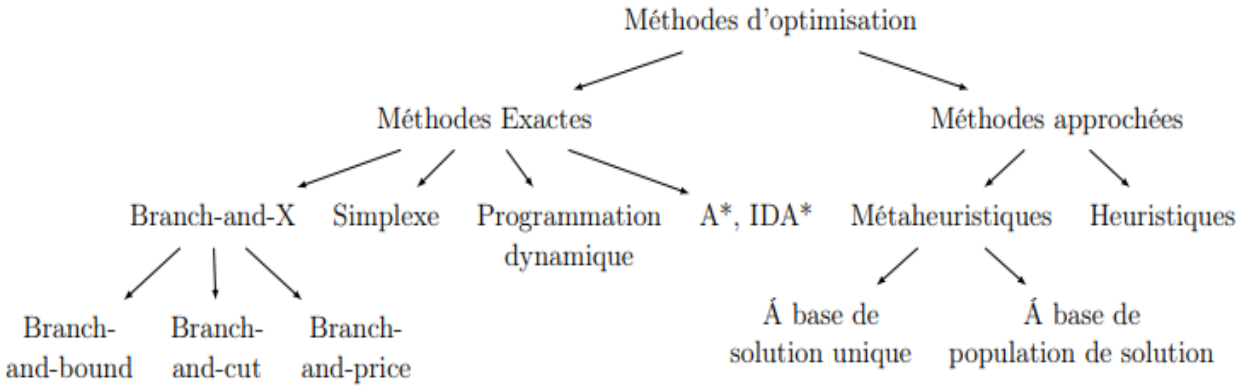


Figure 2.1 : Classes des méthodes de résolutions [14].

2.2.2.1 Heuristique

Une heuristique est une règle d'estimation, une stratégie, une astuce, une simplification, ou toute autre sorte de système qui limite drastiquement la recherche des solutions dans l'espace des configurations [15].

L'heuristique est une méthode de calcul approchée ; simple permettant d'identifier au moins une solution réalisable en un temps de calcul aussi faible que possible. En contrebalance, impossible d'être assuré que la solution trouvée est la meilleure surtout pour les problèmes NP-difficiles.

Les heuristiques se caractérisent par leurs simplicités, la rapidité de leur exécution, et par la facilité adaptée ou combinée avec d'autres types de méthodes, une chose qui augmente considérablement les possibilités de leurs utilisations.

Généralement une heuristique est conçue pour un problème précis et ne peut pas être généralisée, en s'appuyant sur des règles empiriques simples basées sur l'expérience. Une chose qui permet d'aboutir à calculer une solution approchée d'un problème donné.

Les heuristiques peuvent être classées en deux catégories :

- **Méthodes constructives :** ils permettent de construire la solution par une suite de choix optimal, donc à partir d'une solution initialement vide, et de façon incrémentale, les meilleurs éléments seront ajoutés dans l'espoir d'obtenir un résultat final optimal.

- **Méthodes de fouilles locales** : ils partent d'une solution initialement complète, et de manière répétitive tente d'améliorer cette solution en explorant son voisinage immédiat.

2.2.2.2 Métaheuristiques

Le terme Métaheuristique est dérivé de la composition deux de mot grec : Heuristique qui signifie "trouver" et Meta qui est un suffixe signifiant 'au-delà', c'est-à-dire dans un niveau supérieur [16].

Les heuristiques ont rencontré des difficultés pour avoir une solution réalisable et de bonne qualité aux problèmes d'optimisation difficiles, pour cela les métaheuristiques ont fait leur apparition. Ces algorithmes sont plus complets et plus complexes qu'une simple heuristique, et permettent généralement d'obtenir une solution de très bonne qualité pour des problèmes issus des domaines de la recherche opérationnelle ou de l'ingénierie dont la résolution du problème nécessite un temps élevé ou une grande mémoire de stockage [17].

Une heuristique est conçue pour un problème précis et ne peut pas être généralisée. Par contre, une métaheuristique utilise un haut niveau d'abstraction lui permettant de s'adapter facilement aux extensions et aux divers types de problèmes, tout en minimisant ou maximisant une fonction objectif qui décrit la qualité d'une solution d'un problème.

Les métaheuristiques représentent une stratégie de résolution de problème polyvalente, non-déterministes qui coordonne d'autres heuristiques dont le but est la résolution des problèmes d'optimisation difficile. Ils permettent d'obtenir une solution optimale et de haute qualité en progressant vers un optimum global.

Plusieurs classifications des métaheuristiques ont été proposées ; la plupart distinguent globalement deux catégories en fonction du nombre de solutions qu'elles manipulent : les méthodes à base de solution unique, qui travaillent sur un seul point de l'espace de recherche à un instant donné, appelées aussi méthodes à base de voisinage, et les méthodes à base de population, qui travaillent sur un ensemble de points de l'espace de recherche en parallèle.

- **Métaheuristiques à base de solution unique :** cette sorte de métaheuristiques lance la recherche avec une solution initiale, et au cours de la procédure de recherche, elles essayent d'améliorer sa qualité au fur et à mesure tout en choisissant une nouvelle solution dans son voisinage. Elles sont appelés aussi les méthodes de recherche locale ou méthodes de trajectoire car elles construisent une trajectoire dans l'espace des solutions en se redirigeons vers des solutions optimales. Les méthodes de recherche locale (méthode de la descente), la recherche tabou, recuit simulé sont des exemples typiques des méthodes à base de solution unique [18].

- **Métaheuristiques à base de population de solutions :** quant à ce type de métaheuristiques, il débute la recherche avec un ensemble de solutions dites population, et durant les itérations du processus de recherche elles essayent au fur et à mesure d'améliorer leurs qualités afin d'aboutir à des solutions de meilleure performance. Elles sont parfois nommés des méthodes évolutives parce qu'elles font évoluer une population d'individus selon des règles bien précises. L'intérêt de cette forme de métaheuristiques est d'utiliser la population comme facteur de diversité pour augmenter la possibilité d'apparition de bonnes solutions en termes de qualité [19]. Les algorithmes évolutionnaires, l'optimisation par essaim de particules, les algorithmes de colonies de fourmis, les algorithmes de chauves-souris et la recherche coucou sont des instances les plus connus de cette catégorie.

2.3 Quelques algorithmes basés sur l'intelligence en essaim

Durant ces dernières années, plusieurs travaux ont concentré leurs recherches au développement de nouveaux algorithmes basés sur l'intelligence en essaim en s'inspirant du comportement coopératif de certaines espèces dans leur milieu naturel.

En imitant des comportements sociaux dans la nature d'insectes, d'oiseaux, d'animaux, etc., des métaheuristiques ont été proposées pour la résolution des problèmes d'optimisation.

Dans ce qui suit, nous présentons quelques métaheuristiques appartenant à cette catégorie à savoir : l'algorithme de la recherche coucou (Cuckoo Search ou CS), l'algorithme des chauves-souris (Bat Algorithm ou BA) et l'algorithme de pollinisation des fleurs (Flower Pollination Algorithm ou FPA). Nous donnons plus de détails à ce dernier algorithme car c'est celui-là qui sera utilisé dans la partie réalisation de notre projet de fin d'études.

2.3.1 Algorithme de la recherche coucou

La recherche coucou (Cucko Search ou CS) est une approche d'optimisation basée sur l'intelligence en essaim qui a été proposée par Yang et Deb en 2009 [20].

Cet algorithme a été développé en s'inspirant du comportement parasitaire de certaines espèces d'oiseaux. Dans le monde réel, les coucous pondent leurs œufs dans les nids des autres espèces d'oiseaux, en choisissant souvent un nid où l'oiseau hôte pose juste ses propres œufs. Dans la plupart des cas, l'oiseau hôte croit que les œufs déposés sont les siens et par conséquent il en prend soin. En outre, les œufs de coucou éclosent un peu plus tôt que les œufs des hôtes. Une fois que le premier poussin coucou est hachuré, la première action de l'instinct est qu'il va expulser les œufs hôtes aveuglément en les propulsant hors du nid, ce qui augmente la part de nourriture du poussin coucou fourni par son oiseau hôte. Cependant, dans certains cas l'oiseau hôte peut découvrir que les œufs ne sont pas les siens, et par conséquent ils seront soit jetés ou bien le nid est abandonné et construit un nouveau nid dans un nouvel emplacement.

Inspirés par ce comportement, Yang et Deb ont formulé l'algorithme CS en considérant comme génération actuelle l'ensemble des œufs présents dans le nid et l'œuf du coucou comme étant une nouvelle solution candidate générée.

2.3.2 Algorithme des chauves-souris

L'algorithme Bat (BA) est un algorithme bio-inspiré développé par Xin-She Yangin en 2010 [21]. Il a été inspiré du comportement des chauves-souris pendant leur vol et leur chasse, ce comportement appelée l'écholocation, qui est une sorte de sonar, la chauve-souris émet un son d'impulsion fort et court. Lorsque le son atteint un objet, l'écho revient à leurs oreilles en peu de temps, les chauves-souris reçoivent et déterminent leur environnement. Ils peuvent ainsi grâce à ce son émis détecter la distance et faire la différence entre l'orientation de la cible (proie) et l'emplacement des obstacles.

Généralement, les chauves-souris envoient de fortes impulsions, mais lorsqu'elles rencontrent de la nourriture, leur taux d'émission d'impulsions diminue et la fréquence augmente, ce qui leur permet de pouvoir suivre la proie avec plus de précision.

2.3.3 Algorithme de pollinisation des fleurs

La nature a résolu des problèmes difficiles au cours de millions et de milliards d'années, et de nombreux systèmes biologiques ont évolué avec une efficacité intrigante et surprenante pour maximiser leurs objectifs évolutifs tels que la reproduction. Sur la base des caractéristiques réussies des systèmes biologiques, de nombreux algorithmes inspirés de la nature ont été développés au cours des dernières décennies [22]. L'algorithme de pollinisation de fleurs fait partie de cette catégorie d'algorithmes.

2.3.3.1 Caractéristiques de la pollinisation des fleurs

Plus de 80% de toutes les plantes sur Terre sont des plantes à fleurs, une chose qui a attiré les chercheurs pour simuler l'évolution des fleurs. À l'origine, Xin-She Yang en 2012 [22] a proposé un algorithme de pollinisation des fleurs bio-inspiré naturel tirant sa métaphore du processus de pollinisation des plantes à fleurs. Récemment, l'algorithme de pollinisation des fleurs (Flower Pollination Algorithm ou FPA) acquiert une grande prise de conscience en raison de son applicabilité efficace aux problèmes d'optimisation dans divers domaines de la vie réelle.

Du point de vue de l'évolution biologique, l'objectif principal de la pollinisation des fleurs est la reproduction optimale des plantes en survivant des fleurs les plus aptes dans les plantes à fleurs [22].

Tous ces facteurs impliqués dans ce processus interagissent systématiquement entre eux pour obtenir une reproduction optimale des plantes à fleurs.

Pour mieux comprendre le principe de cette technique d'optimisation, nous commençons par donner une brève description de ses fondements biologiques.

Généralement, la pollinisation des fleurs est associée au transfert d'une substance chimique appelée pollen, et ce transfert est souvent lié à certaines créatures appelées pollinisateurs ou parfois vecteurs polliniques, qui peuvent être de divers types comme les insectes, les oiseaux, les chauves-souris et d'autres animaux. Lorsqu'un insecte visite une fleur, les grains de pollen collent à son corps. Si l'insecte visite une autre fleur, le pollen est transféré vers la stigmatisation de la fleur

visitée, ce qui facilite la pollinisation.

En fait, il existe deux formes principales dans le processus de pollinisation basé sur les pollinisateurs : biotique et abiotique. Environ 90% des plantes à fleurs appartiennent à la pollinisation biotique, dans laquelle le pollen est transféré par un pollinisateur spécifique, on estime qu'il existe au moins 200000 variétés de pollinisateurs tels que les insectes, les chauves-souris et les oiseaux [22]. Concernant la seconde forme, qui a une occurrence limitée car elle ne nécessite aucun organisme, le vent, la gravité ou la diffusion dans l'eau contribuent à transférer du pollen et l'herbe est le bon exemple pour cette pollinisation.

Pour plus de détail la figure 2.2 présente les deux formes de pollinisateurs et les types de pollinisation.

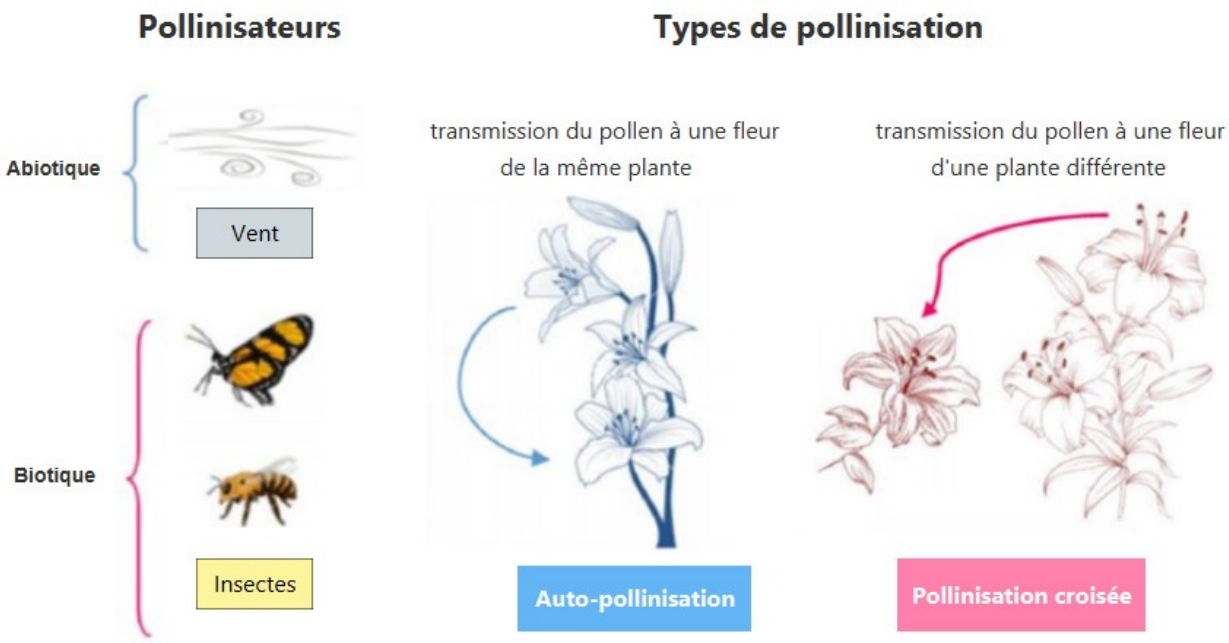


Figure 2.2 : Les pollinisateurs et les types de pollinisation.

Les fleurs peuvent avoir des pétales colorés, des parfums et du nectar qui rendent une plante plus visible pour les pollinisateurs [23]. Par exemple, certaines fleurs ne peuvent qu'attirer et ne peuvent dépendre que d'une espèce spécifique d'insectes pour une pollinisation réussie [22]. De plus, en visitant les mêmes espèces de fleurs, les pollinisateurs peuvent assurer l'approvisionnement en nectar avec moins d'exploration et préserver la constance des fleurs [23].

Les abeilles sont un bon exemple de pollinisateur représentant un facteur essentiel dans une forme de pollinisation biotique, elles peuvent également développer la constance florale [24]. Autrement dit, ces pollinisateurs ont tendance à visiter certaines espèces de fleurs exclusives tout en passant par d'autres espèces de fleurs, ce comportement est appelé la constance florale. Cette dernière peut avoir des avantages évolutifs car cela maximisera le transfert de pollen de fleurs aux mêmes plantes, et maximisera ainsi la reproduction des mêmes espèces de fleurs. Une telle constance florale peut également être avantageuse pour les pollinisateurs, car ils seront sûrs que

l'approvisionnement en nectar est disponible avec leur mémoire limitée et leur coût minimum d'apprentissage ou d'exploration. Plutôt que de se concentrer sur certaines nouvelles espèces de fleurs imprévisibles mais potentiellement plus gratifiantes, la constance des fleurs peut nécessiter un minimum coût d'investissement et plus probablement une consommation garantie de nectar [25].

La pollinisation peut être réalisée par autopollinisation ou pollinisation croisée. Le premier type de pollinisation, appelé aussi pollinisation locale, se produit lorsque le pollen d'une fleur pollinise la même fleur ou d'autres fleurs de la même plante à l'aide de facteurs environnementaux [22]. Tandis que la pollinisation croisée, également connue sous le nom de pollinisation mondiale, se produit sur des longues distances lorsque le pollen est livré à une fleur à partir d'une autre plante par le biais d'une intervention directe ou indirecte de pollinisateurs suivant un comportement de vol de Levy [26].

2.3.3.2 Description de l'algorithme

De toute évidence, chaque plante peut avoir plusieurs fleurs, et chaque fleur libère souvent des millions, voire des milliards de gamètes de pollen. Cependant, pour simplifier, nous supposons que chaque plante n'a qu'une seule fleur et que chaque fleur ne produit qu'un gamète de pollen. Ainsi, cet argument signifie qu'une fleur et / ou un pollen est équivalent à une solution candidate au cours du processus d'optimisation [23], il n'est pas nécessaire de distinguer un gamète de pollen, une fleur, une plante ou une solution à un problème.

Dans l'algorithme FPA, au cours du processus d'optimisation, l'exploration de l'espace de recherche se fait par pollinisation biotique et croisée où le mouvement du pollen est représenté par le vol de Lévy. Ce dernier est une marche aléatoire basée sur une étape aléatoire tirée de la distribution de Lévy, provoquant un mouvement beaucoup plus long depuis sa position actuelle.

Par conséquent, nous pouvons idéaliser les caractéristiques du processus de pollinisation, de la constance des fleurs et du comportement des pollinisateurs sur la base de quatre règles principales énumérées comme suit :

- R1 : La pollinisation biotique, croisée agissant comme un processus de pollinisation mondial via le vol de Lévy.
- R2 : L'abiotique et l'autopollinisation sont considérés comme une pollinisation locale.
- R3 : La constance des fleurs peut être impliquée en raison de la similitude de deux fleurs.
- R4 : La pollinisation locale et la pollinisation globale sont contrôlées par une probabilité de commutation $p \in [0,1]$.

Sur la base de ces quatre règles, les étapes de base de l'algorithme FPA standard peuvent être résumées sous la forme du pseudo-code suivant :

Fonction objectif $\min$ ou $\max$ $f(x), x=(x_1,x_2,...)$
Initialiser la population de n fleurs / pollen avec des solutions aléatoires
Trouver la meilleure solution g_* dans la population initiale
Définir une probabilité de commutation $p \in [0,1]$
Tant que $(t < MaxGeneration)$ ou (critère d'arrêt) **faire**
    Pour $i=1:n$ (toutes les n fleurs de la population)
    Si $rand() < p$ **alors**
        Tirer une taille de pas L qui obéit à une distribution Levy
        Pollinisation globale via $x_i^{t+1} = x_i^t + L(x_i^t - g_*)$
    Sinon
        Tirer ε d'une distribution uniforme dans $[0,1]$
        Choisissez au hasard j et k parmi toutes les solutions
        Pollinisation locale via $x_i^{t+1} = x_i^t + \varepsilon(x_j^t - x_k^t)$
    Fin Si
        Evaluer de nouvelles solutions
        Si la nouvelle solution générée est meilleure, remplacez x_i^t par x_i^{t+1}
    Fin Pour
    Trouvez la meilleure solution actuelle g_*
Fin tant que

L'algorithme commence par générer aléatoirement la population initiale qui sera évaluée afin de déterminer la meilleure solution actuelle. Pour calculer une nouvelle solution, le type de pollinisation doit être déterminé dans un premier temps selon une probabilité prédéterminée p (R4). C'est-à-dire qu'un nombre aléatoire $r \in [0,1]$ est généré et si r est inférieur à p, alors la pollinisation globale et la constance de la fleur (R1 et R3) peuvent avoir lieu comme suit :

$$x_i^{t+1} = x_i^t + L(x_i^t - g_*) \tag{2.1}$$

Où x_i^t est une solution i à l'itération t, x_i^{t+1} est le vecteur de solution généré à l'itération $t+1$, g_* est la meilleure solution actuelle trouvée parmi toutes les solutions dans la génération. Le paramètre L est une taille de pas basée sur les vols de Lévy. Étant donné que les insectes peuvent se déplacer sur une longue distance avec différents pas de distance, cela est modélisé en utilisant une distribution de Lévy [26] selon l'équation suivante:

$$L \sim \frac{\lambda \Gamma(\lambda) \sin(\frac{\pi\lambda}{2})}{\pi} \frac{1}{s^{1+\lambda}}, \quad (s > 0). \tag{2.2}$$

Dans cette équation, $\Gamma(\lambda)$ est la fonction gamma standard, et cette distribution est valable pour les grands pas $s > 0$ qui sont généré selon l'équation suivante:

$$s = \frac{U}{|V|^{1/\lambda}} \tag{2.3}$$

Où U et V sont deux échantillons aléatoires tirés d'une distribution normale gaussienne avec une moyenne égale à zéro et des écarts-types σ_u et σ_v :

$$U \sim (0, \sigma_u^2), V \sim (0, \sigma_v^2). \tag{2.4}$$

$$\sigma_u = \left[\frac{\Gamma(1+\lambda)}{\lambda \Gamma((1+\lambda)/2)} \cdot \frac{\sin(\frac{\pi\lambda}{2})}{2^{(\lambda-1)/2}} \right]^{1/\lambda}, \sigma_v = 1. \tag{2.5}$$

Sinon, si r est supérieur à p , la pollinisation locale et la constance des fleurs (R2 et R3) sont effectuées comme :

$$x_i^{t+1} = x_i^t + \varepsilon(x_j^t - x_k^t) \tag{2.6}$$

Où x_j^t et x_k^t sont deux solutions différentes choisies au hasard et $\varepsilon \in [0,1]$ un nombre aléatoire pour rapprocher cette sélection d'une marche aléatoire locale [22]. Après cela, la nouvelle solution sera évaluée en fonction de sa fitness (fonction objectif). Par la suite, la nouvelle génération sera également évaluée pour sélectionner la plus prometteuse et le processus de recherche reprend jusqu'à ce que le critère d'arrêt soit satisfait.

2.4 Conclusion

Dans ce chapitre, nous avons présenté les méthodes de résolution de problèmes, à savoir les méthodes exactes et les méthodes approchées en se focalisant sur les métaheuristiques. Une attention particulière a été donnée à la métaheuristique FPA (Flower Pollination Algorithm) ou l'algorithme de pollinisation des fleurs en présentant ses caractéristiques ainsi que sa description sous forme de pseudo code.

Dans le cadre de notre PFE, nous nous intéressons, dans le chapitre suivant, à l'application de l'algorithme FPA pour l'ordonnancement de tâches multi-objectif dans le domaine du Cloud computing.

CHAPITRE 3:

Implémentation de l’application et évaluation des résultats obtenus

3.1 Introduction

Le problème d'ordonnancement des tâches dans le Cloud computing est caractérisé par le fait qu'il comporte des objectifs complexes et souvent conflictuels. Il est considéré comme un problème d'optimisation multi-objectif, dont la complexité augmente avec l'augmentation des contraintes et les paramètres considérés. Dans ce contexte, trouver un bon compromis entre les objectifs reste un défi majeur.

En général, le problème d'ordonnancement est connu pour être NP-complet, où il est impossible de trouver la solution optimale en utilisant des algorithmes classiques. Par conséquent, ce problème a un enjeu majeur nécessitant de trouver une manière plus efficace pour évaluer tous ces différents objectifs en un temps raisonnable d'où la nécessité d'utiliser les métaheuristiques.

Dans ce contexte, l'objectif de l'ordonnancement consiste à une répartition des tâches sur des ressources délivrées par le fournisseur de services Cloud de façon à optimiser plusieurs métriques de qualité de services (QoS). Notons que la plupart des travaux sont concentrés souvent sur l'optimisation d'une seule métrique de QoS qui est le makespan.

L'objectif de ce chapitre est de décrire en détail notre contribution dans le cadre de ce projet de fin d'études. Cette contribution consiste à utiliser un algorithme basé sur la métaheuristique FPA pour l'ordonnancement des tâches multi-objectif dans le Cloud computing par rapport à 3 critères qui sont : le makespan, le coût et la fiabilité. Nous présentons également une évaluation mono-objectif de l'algorithme FPA à travers une comparaison avec 3 heuristiques classiques dans le domaine de l'ordonnancement des tâches à savoir, Max-Min, Min-Min et Round Robin.

Dans le reste de ce chapitre, nous commençons par présenter les outils de simulations utilisés dans le cadre de ce PFE, à savoir, le langage java, l'IDE NetBeans, l'outil Jfreechart et le simulateur CloudSim. Pour ce simulateur, nous nous intéressons à son architecture, ses classes fondamentales ainsi que les classes que nous avons intégrées pour la réalisation de notre travail.

Nous présentons par la suite les principes de base de l'optimisation multi-objectif, notamment, quelques méthodes pour mesurer la qualité d'un ensemble de solutions non-dominées. Nous présentons également l'IHM que nous avons développée dans le cadre de ce PFE ainsi qu'une étude comparative par rapport aux résultats obtenus.

3.2 Environnement de développement et de simulation

3.2.1 Java

Java est un langage de programmation informatique orienté objet et un environnement d'exécution informatique portable, créé par James Gosling et Patrick Naughton employés de Sun Microsystems avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld [27].

Le langage Java a la particularité principale que les logiciels écrits avec ce dernier sont très facilement portables sur plusieurs systèmes d'exploitation tels que : Unix, Microsoft Windows, Mac OS ou Linux [28].

Java est un langage compilé et interprété car le compilateur java produit un code intermédiaire (Byte-code) qui sera interprété par une JVM (Java Virtual Machine). Cette dernière représente la garantie de la portabilité du code qui a fait la réussite de ce langage.

3.2.2 Netbeans

NetBeans est un environnement de développement intégré (EDI) open source, conçu en Java par Sun en juin 2000. Il est principalement destiné pour le développement en Java, mais permet également de prendre en charge d'autres langages, notamment Python, C, C ++, HTML, PHP et JavaScript. Il comprend toutes les caractéristiques d'un EDI moderne (éditeur en couleur, projets multi-langage, éditeur graphique d'interfaces et de pages Web, etc) à l'aide de plugins [29].

NetBeans est multiplateforme et disponible sous différentes versions indépendantes des systèmes d'exploitation. Il fonctionne sur Windows, Linux, Mac OS X, et d'autres plates formes supportant une JVM compatible.

3.2.3 Jfreechart

JFreeChart est une API Java open source sous licence LGPL (Lesser General Public Licence) qui permet aux développeurs de visualiser facilement des graphiques de qualité professionnelle dans leurs applications. Elle propose de nombreuses options de configuration pour personnaliser le rendu des graphes.

- Composants Swing.
- Images (PNG ou JPEG).
- Images vectorielles PDF, EPS et SVG.

3.2.4 CloudSim

Les recherches récentes de conception et de développement de technologie sur le Cloud computing se concentrent sur de nouveaux mécanismes et politiques pour une gestion efficace des infrastructures Cloud. Pour tester ces mécanismes et politiques nouvellement développées, il est nécessaire d'un outil où l'on peut reproduire des tests et évaluer l'hypothèse avant le déploiement dans un environnement réel.

Pour répondre à ces exigences, le laboratoire Cloud Computing and Distributed Systems (CLOUDS) de Melbourne en Australie, a développé une boîte à outils CloudSim pour la modélisation et la simulation d'environnements de Cloud computing à grande échelle [30].

CloudSim est un outil de simulation généralisé et extensible qui prend en charge à la fois la modélisation du système et du comportement des composants du Cloud computing tels que les DataCenters, les machines virtuelles (VM), les cloudlets (tâches) et les brokers. De plus, il expose des interfaces personnalisées pour la mise en œuvre de stratégies et de techniques de provisionnement pour l'allocation des ressources. CloudSim implémente des techniques génériques d'application qui peuvent être étendues facilement et avec un effort limité.

Deux politiques de planification des tâches sont définies dans ce simulateur :

- **Space shared :** Dans space shared, une seule tâche s'exécute sur la VM jusqu'à la fin et après sa terminaison, une autre tâche sera lancée sur cette VM.
- **Time shared :** Dans time shared, plus d'une tâche peut être exécutée sur la même VM dans une tranche de temps différente.

La structure logicielle de CloudSim et ses composants est organisée sous formes de plusieurs couches comme c'est indiqué dans la figure 3.1.

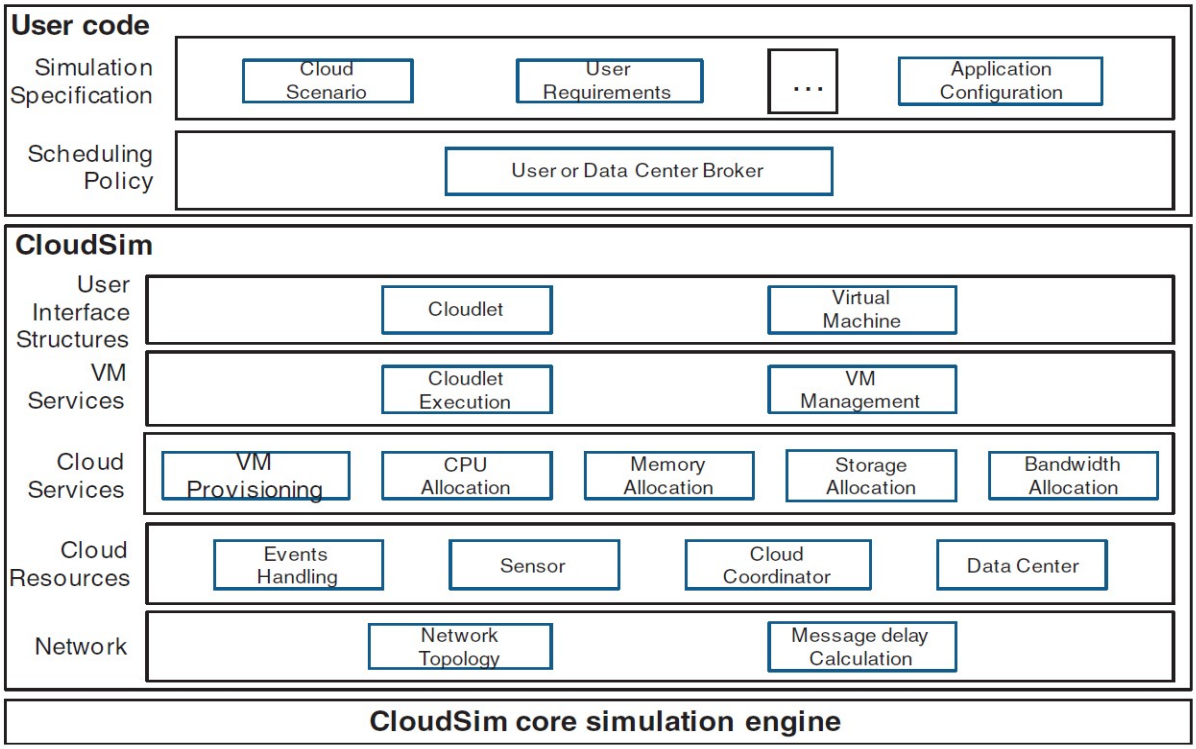


Figure 3.1 : Architecture de CloudSim [31].

La couche la plus haute de la pile de simulation, est **User code** qui expose les entités de base, qui doivent être gérées par l'utilisateur telle que les hôtes (nombre de machines, leurs spécifications, etc.), les applications (nombre de tâches et leurs besoins), les machines virtuelles, le nombre d'utilisateurs et les politiques de planification des brokers.

Dans la deuxième couche, on trouve **CloudSim** qui prend en charge plusieurs fonctionnalités de base requises pour la simulation au niveau supérieur, telles que les files d'attente, le traitement des événements, la création des entités du système (services, hôte, Datacenter, Broker, machines virtuelles), la communication entre les entités et la gestion de l'horloge de simulation. La simulation des problèmes fondamentaux, tels que l'approvisionnement d'hôtes aux machines virtuelles, la gestion de l'exécution des applications et la surveillance de l'état dynamique du système sont traités aussi par cette couche [31].

Le simulateur CloudSim est composé de plusieurs classes. Parmi les classes fondamentales qui forment des éléments constitutifs du simulateur, nous pouvons citer:

- **DC :** cette classe modélise les principaux services dans le Cloud computing. Elle encapsule un ensemble de machines physiques appelées hosts qui peuvent être homogènes ou hétérogènes selon leurs configurations matérielles (CPU, mémoire, capacité et stockage). En outre, chaque Datacenter implémente un ensemble de stratégies pour allouer la bande passante, la mémoire et les périphériques de stockage aux hosts et aux machines virtuelles.

- **DatacenterBroker :** cette classe modélise un courtier (Broker), qui est responsable de la médiation des négociations entre les fournisseurs de services Cloud et leurs clients. Ces négociations sont motivées pour l'allocation de ressources qui répond aux besoins de QoS des utilisateurs, afin d'augmenter leurs mesures de performance. Elle permet aussi de déployer les tâches de service à travers les Clouds.

- **Host :** cette classe représente un serveur informatique physique comme un serveur de calcul ou de stockage. Elle encapsule des informations telles que la quantité de mémoire et de stockage, une liste des types de cœurs de traitement (pour représenter une machine multi-core), une stratégie d'allocation de partage de la puissance du traitement entre les machines virtuelles et des stratégies d'attribution de mémoire et de bande passante pour les VM.

- **Vm :** cette classe modélise une instance de machine virtuelle (VM) gérée et hébergée par un host. Chaque composant de machine virtuelle a accès à un composant qui stock les caractéristiques liées à la VM : mémoire, processeur, capacité de stockage.

- **Cloudlet :** cette classe modélise les services d'application basés sur le Cloud tels que la diffusion de contenu, les réseaux sociaux. CloudSim représente la complexité d'une application en termes d'exigences de calcul. Chaque service d'application a une taille d'instruction et une quantité de flux de transfert de données pré-attribués qu'il doit entreprendre pendant son cycle de vie. Cette classe peut également être étendue pour prendre en charge la modélisation d'autres mesures de performances.

3.3 Techniques d'optimisation multicritères

Dans la plupart des problèmes d'optimisation, il ne s'agit pas d'optimiser seulement un seul critère mais plutôt d'optimiser simultanément plusieurs critères et qui sont généralement conflictuels. L'optimisation mono-objectif s'intéresse à minimiser un seul objectif définis par une seule fonction objectif où le but est de trouver la solution optimale qui est facilement définie suivant une seule performance du problème étudié. De l'autre part, l'optimisation multi-objectif également connue sous le nom "optimisation multicritères" vise à optimiser plusieurs objectifs qui sont souvent contradictoires. Par exemple, le développement d'une nouvelle voiture, implique la maximisation de sa puissance tout en minimisant la consommation du carburant. Ce qui signifie que l'atteinte de l'optimum pour un objectif exige un compromis sur un ou plusieurs autres objectifs. En d'autres termes, l'amélioration de l'un entraîne la détérioration de l'autre ou des autres critères.

De ce fait, les problèmes d'optimisation multicritères sont considérés comme une partie indissociable de l'optimisation dont le but est de rechercher un compromis entre les différents objectifs. Cela a conduit les chercheurs à proposer des méthodes de plus en plus performantes, pour résoudre ces problèmes d'optimisation multicritères.

Les méthodes de prise de décision multicritères (Multi-Criteria Decision Making ou MCDM) ont été appliquées à différentes applications pour choisir la meilleure alternative. Les méthodes MCDM donnent une bonne prise de décision dans les domaines où la sélection de la meilleure alternative est très complexe. Ils aident à choisir les meilleures alternatives où de nombreux critères ont vu le jour [32].

La théorie de la prise de décision multicritère propose des méthodes pour gérer un ensemble de critères où nous devons choisir une seule solution parmi un ensemble d'actions ou d'alternatives. Ces méthodes diffèrent selon la façon d'évaluer les solutions en fonction des critères retenus.

Parmi les méthodes multicritères les plus connues, on peut citer la méthode de la somme pondérée (Weight Sum Method ou WSM), la méthode de produit pondéré (Weighted Product Model ou WPM), AHP (Analytic Hierarchie Process), ELECTRE (Outranking Method) ainsi que TOPSIS (Technic for Order Performance by Similarity to Ideal Solution).

Nous allons détailler dans ce qui suit les deux méthodes qui ont été utilisées dans le cadre de ce PFE, à savoir WSM et TOPSIS. Cette dernière méthode a été combinée avec le principe de Pareto.

3.3.1 Méthode de la somme pondérée

En théorie de la décision, la méthode de la somme pondérée (Weight Sum Methode ou WSM) est la plus connue des méthodes multicritères dû à sa simplicité et sa rapidité permettant une réaction efficace en temps réel pour évaluer un certain nombre d'alternatives en termes d'un certain nombre de critères de décision. Il est très important d'indiquer ici qu'elle n'est applicable que lorsque tous les critères soient quantitatifs, qu'ils aient exactement tous la même unité.

L'approche principale retenue de cette méthode consiste à fusionner les problèmes de façon à ce qu'il ne reste qu'une seule valeur à optimiser. En d'autres termes, elle consiste à reformuler un ensemble d'objectifs d'un problème multi-objectif en un problème d'optimisation mono-objectif en multipliant par une valeur de poids w_i prédéfinie relativement pour chaque fonction objectif afin que les optimiseurs puissent se concentrer sur certains objectifs.

De cette manière, une fois le problème et l'importance de chaque critère sont définis, une seule solution est générée comme le montre l'équation suivante.

$$f(x) = \sum_{i=1}^{n} w_i \, f_i \tag{3.1}$$

Les poids w_1, w_2, …, w_n doivent satisfaire les deux contraintes suivantes :

$$\sum_{i=1}^{n} w_i = 1, \quad 0 \le w_i \le 1 \tag{3.2}$$

D'un autre côté, vu que le conflit qui se trouve entre les objectifs, il est clair que l'on ne peut pas tout optimiser équitablement. Ainsi, l'utilisation de la notion de poids permettra à l'utilisateur d'exprimer ses préférences en mettant l'accent sur un objectif donné.

L'idée de base c'est que le critère qui est en contradiction avec celui qui est favorisé reçoit la valeur minimale parmi les poids possibles (ceux disponibles après l'opération 1 – le poids du critère pertinent) [33].

Par conséquent, le processus d'optimisation se lance dans la quête d'un compromis entre l'ensemble des objectifs, tout en favorisant celui qui s'assortit avec la préférence de l'utilisateur.

3.3.2 TOPSIS avec Pareto

L'optimisation multi-objectif exige que l'importance relative de chaque objectif doit être spécifié à l'avance, ce qui nécessite une connaissance préalable des solutions possibles. Mais, en utilisant le concept de Pareto, il est possible d'éviter le besoin de connaître à l'avance les solutions possibles. En effet, ceci est l'une des popularités de ces approches basées sur Pareto [32].

La résolution d'un problème multicritère est très complexe, car cette résolution fournie comme résultat final un ensemble de solutions optimales au lieu de fournir une solution unique et optimale.

Dans ce qui suit, nous définissons certains concepts pertinents liés à l'optimisation multi-objectif basée sur Pareto.

Dominance : dans un problème de minimisation, on dit qu'une solution X domine une solution Y si $\forall i = 1, 2, ..., k.\ f_i(X) \leq f_i(Y)$, et il existe $k \in \{1, 2, ..., k\}$ tel que $f_k(X) < f_k(Y)$.

Solutions Pareto-optimales : une solution X est appelée Pareto-optimale si elle n'est dominée par aucune autre solution dans l'espace de décision. Les solutions Pareto-optimales aussi appelées solutions non dominées sont les solutions qui ne se dominent pas, c'est-à-dire il est impossible de trouver une solution permettant d'améliorer la valeur d'un objectif sans dégrader celle d'au moins un autre objectif. Ces solutions sont définies comme suit :

$$P^* = \{x \in X \ / \ \nexists\ x' \in X, x' < x\} \tag{3.3}$$

Comme la relation de dominance de Pareto définit un ordre partiel, plusieurs solutions peuvent être Pareto-optimale. Le front de Pareto contient toutes les solutions Pareto-optimales, il est défini comme suit :

$$PF^* = F(x), x \in P^* \tag{3.4}$$

Dans un problème multicritères, nous cherchons donc à générer un front de Pareto ou une approximation de cet ensemble dont les valeurs des fonctions objectifs sont proches des solutions Pareto-optimales.

Souvent, le front de Pareto contient un nombre de solutions exponentiel, où il est nécessaire de quantifier l'intérêt d'un ensemble de solutions non-dominées pour évaluer la qualité du résultat

produit. De ce fait, l'utilisation d'une méthode multicritères comme TOPSIS est nécessaire pour sélectionner la solution la plus compromettante du front de Pareto.

La méthode TOPSIS est une méthode d'analyse de décision multicritères qui a été développée à l'origine par Hwang et Yoon en 1981 [34].

L'idée principale de TOPSIS est que l'alternative choisie devrait avoir la distance la plus courte de la solution idéale (idéale positive) et la plus éloignée de la solution anti-idéale (idéale négative). Cette méthode est utilisée à des fins de classement et pour obtenir les meilleures performances dans la prise de décision multicritères.

Dans ce qui suit, nous allons détailler les différentes étapes de l'algorithme TOPSIS. Supposons que nous avons une matrice X avec m alternatives et n critères et nous avons la valeur de chaque alternative par rapport à chaque critère. Où x_{ij} représente la valeur d'alternative i par rapport au critère j.

Etape 1 : Normaliser la matrice de décision pour obtenir une nouvelle matrice R d'élément r_{ij} tel que :

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} {x_{ij}}^2}} \quad pour \quad j = 1, 2, ..., n \tag{3.5}$$

Etape 2 : Calculer la matrice normalisée pondérée. Les poids w_j sont donnés par les décideurs pour représenter leurs préférences entre les critères avec $\sum_{j=1}^{n} w_j = 1.$

$$v_{ij} = w_j * r_{ij} \quad pour \quad i = 1, 2, ..., m. \quad j = 1, 2, ..., n \tag{3.6}$$

Etape 3 : Définir l'idéal positif s^+ et l'idéal négatif s^- :

$$S^+ = \left\{v_{1j}, v_{2j}, v_{3j}, ..., v_{mj}\right\} = \left\{\max v_{ij} \ \forall \ j \in n\right\} \partial \tag{3.7}$$

$$S^- = \left\{v_{1j}, v_{2j}, v_{3j}, ..., v_{mj}\right\} = \left\{\min v_{ij} \ \forall \ j \in n\right\} \partial \tag{3.8}$$

Etape 4 : Calculer pour chaque alternative, la distance euclidienne entre l'idéal positif et l'idéal négatif, notées D_i^+ et D_i^- respectivement :

$$D_i^+ = \sqrt{\sum_{j=1}^{n} (v_{ij} - s_j^+)^2} \quad pour \quad i = 1, 2, ..., m \tag{3.9}$$

$$D_i^- = \sqrt{\sum_{j=1}^{n} (v_{ij} - s_j^-)^2} \quad pour \quad i = 1, 2, ..., m \tag{3.10}$$

Etape 5 : Calculer le degré de proximité au positif idéal. Plus est important, plus l'alternative est proche de l'idéal positif et loin de l'idéal négatif :

$$C_i = \frac{D_i^-}{D_i^- + D_i^+} \quad pour \quad i = 1, 2, ..., m, \quad 0 \le C_i \le 1 \tag{3.11}$$

Etape 6 : Finalement, trier les solutions par rapport à C_i . Les alternatives seront alors classées par ordre de préférence. La plus grande valeur C_i est choisie comme meilleure solution pour le problème multi-objectif.

3.4 Approche et démarche proposée

Dans cette section, nous décrivons notre approche utilisée dans le cadre de ce PFE d'une manière formelle. Dans ce contexte, l'objectif de notre ordonnancement consiste à affecter des tâches aux VMs de façon à optimiser plusieurs métriques de QoS. Pour plus de détails, nous allons utiliser deux scénarios.

Dans un premier scénario, nous traitons un ordonnancement mono-objectif par rapport à une seule métrique qui est le makespan. Nous évaluons par conséquent les performances de notre algorithme FPA par rapport aux heuristiques Round Robin, Max Min et Min Min que nous présenterons par la suit.

Dans un second scénario, nous se focalisant sur un ordonnancement multi-objectif par rapport à 3 critères qui sont le makespan, le coût et la fiabilité. Nous utiliseront également deux versions d'évaluation des solutions dans l'algorithme FPA, à savoir la somme pondérée et le principe de Pareto basé sur l'algorithme TOPSIS pour aboutir à un choix optimal.

3.4.1 Évaluation Mono-objectif

Nous présentons dans ce qui suit, les algorithmes d'ordonnancement et d'allocation de ressources utilisé dans notre PFE pour l'optimisation du critère makespan.

3.4.1.1 Min-Min

L'algorithme Min-Min commence par un ensemble de tâches non planifiées, puis détermine les temps d'exécution minimum pour chaque tâche sur toutes les machines. Ensuite, la tâche avec un temps d'exécution minimum est sélectionnée. En fonction de ce temps minimum, la tâche est ordonnancée sur la machine correspondante. Le temps de disponibilité de cette ressource est mis à jour pour toutes les autres tâches, en ajoutant le temps d'exécution de la tâche assignée au temps d'exécution des autres tâches sur cette machine. La tâche planifiée est ensuite détachée de la liste des tâches et la procédure est répétée jusqu'à ce que toutes les tâches non planifiées soient épuisées.

3.4.1.2 Max-Min

L'algorithme Max-Min suit le même principe que l'algorithme Min-Min sauf qu'après avoir déterminé le temps d'exécution minimum, la valeur maximale est sélectionnée et en fonction de

cette valeur, la tâche est planifiée sur la machine correspondante. Puis le temps de disponibilité de la machine qui a acquise la tâche sélectionnée est mis à jour et la tâche assignée est supprimée de la liste des tâches. Le processus est répété jusqu'à ce que toutes les tâches soient planifiées.

3.4.1.3 Round robin

Cet algorithme suit une stratégie simple qui consiste à distribuer de manière équitable les tâches sur les machines virtuelles disponibles, c'est-à-dire que le nombre de tâches pour chaque machine virtuelle est le même. Cet algorithme est implémenté dans le simulateur CloudSim.

L'algorithme Round robin se concentre principalement sur la distribution de la charge de manière égale à toutes les ressources.

3.4.2 Évaluation Multi-objectif

Dans ce cas, nous évaluons trois critères d'optimisation qui sont : le makespan, le coût et la fiabilité.

Makespan : considéré comme l'un des mesures les plus utilisées pour l'évaluation des algorithmes d'ordonnancement. Le makespan représente la différence entre le temps de soumission de la première tâche et le temps de réception des résultats. Autrement dit, c'est le temps de fin d'exécution de la dernière tâche.

Coût : la plupart des fournisseurs de Cloud fixe les prix pour l'utilisation de leurs services. Dans notre cas, le coût total de l'exécution correspond à la somme des coûts liés à l'utilisation des VMs pour l'ensemble des tâches.

Le coût d'exécution d'une tâche dépend, à la fois, de sa durée d'exécution sur la machine ainsi que du prix unitaire de l'exécution des tâches sur cette machine.

Fiabilité : la fiabilité représente la probabilité qu'une tâche soit exécutée avec succès, sans aucune faute de ressources. Notre modèle pour mesurer la fiabilité est basé sur un taux d'échec λ qui est une propriété intrinsèque de la ressource. La fiabilité est calculée selon l'équation 3.12 [35].

$$Fiabilité = \exp^{-\sum_{i=1}^{n} TE(T_i)*\lambda_j} \tag{3.12}$$

Où, $TE(T_i)$ est le temps d'exécution de la tâche T_i et λ_i est le taux d'échec de la machine qui exécute la tâche.

Les 3 métriques sont classées en deux catégories : les métriques de QoS à maximiser (fiabilité) et celles à minimiser (makespan et coût).

Dans le cadre d'une optimisation multi-objectif, la fonction objectif qui résulte de l'agrégation des trois fonctions objectifs définies précédemment est comme suit :

$$F = w_1 * Makespan + w_2 * Coût + w_3 * \left(\frac{1}{Fiabilité} \right) \tag{3.13}$$

Où $w = \{w_1, w_2, w_3\}$ désigne le vecteur des poids qui traduit l'importance ou l'exigence de l'utilisateur par rapport à chaque critère. Pour cela, nous avons utilisé deux vecteurs différents de poids $\{0.5, 0.2, 0.3\}$ et $\{0.2, 0.5, 0.3\}$ correspondant respectivement aux 3 critères $\{Makespan, Coût, Fiabilité\}$. Le premier vecteur favorise le makespan, le deuxième vecteur favorise le cout.

Comme la fonction F est à minimiser et comme le troisième paramètre (Fiabilité) est à maximiser, donc nous prendrons son inverse (1/Fiabilité) pour minimiser toutes les fonctions.

En seconde lieu, l'implémentation de la méthode TOPSIS nécessite des poids pour chaque critère. Pour cela, nous avons utilisé les mêmes poids définis précédemment $\{0.5, 0.2, 0.3\}$ et $\{0.2, 0.5, 0.3\}$ correspondant respectivement aux 3 critères $\{Makespan, Coût, Fiabilité\}$.

3.4.3 Adaptation de la métaheuristique FPA

Le pseudo code de la métaheuristique FPA qui a été adaptée dans le cadre de ce PFE pour l'affectation des tâches dans les différentes VMs est donné comme suit.

Définir une probabilité de commutation $p \in [0,1]$

Pour chaque Tâche T_i de la liste des cloudlets **faire**

Initialiser la population initiale avec des VMs aléatoires

Trouver la meilleure machine virtuelle V_{best} pour la tâche T_i

Tant que critère d'arrêt **faire**

Pour chaque machine virtuelle V_i dans la population **faire**

Si $rand() < p$ **alors**

Tirer une taille de pas L qui obéit à une distribution Levy

$V_i' = V_i + L\,(V_i - V_{best})$

Sinon

Tirer ε d'une distribution uniforme dans $[0,1]$

Tirer au hasard deux machine virtuelles V_j et V_k

$V_i' = V_i + \varepsilon\,(V_j - V_k)$

Fin Si

Evaluer la machine virtuelle V_i'

Si la nouvelle machine générée est meilleure, remplacez V_i par V_i'

Fin Pour

Mettre à jour V_{best} la meilleure machine dans la population

Fin tant que

Fin pour

Une phase de pré-simulation en utilisant le FPA adapté est nécessaire afin de trouver les meilleures affectations des cloudlets aux VMs avant de les envoyer au Broker.

3.5 IHM développée

CloudSim s'exécute uniquement en mode console et n'a pas d'interface graphique, par conséquent nous avons créé notre IHM pour faciliter l'accès et la manipulation de la simulation.

Afin d'examiner le comportement de notre algorithme, nous devons d'abort passer par l'étape de configuration pour déterminer les paramètres de simulation.

Les interfaces suivantes présentent les différentes étapes pour paramétrer la simulation.

Interface Principale :

Dès le lancement de notre l'application, la fenêtre d'accueil illustrée dans la figure 3.2 apparait, elle contient les informations de notre PFE.

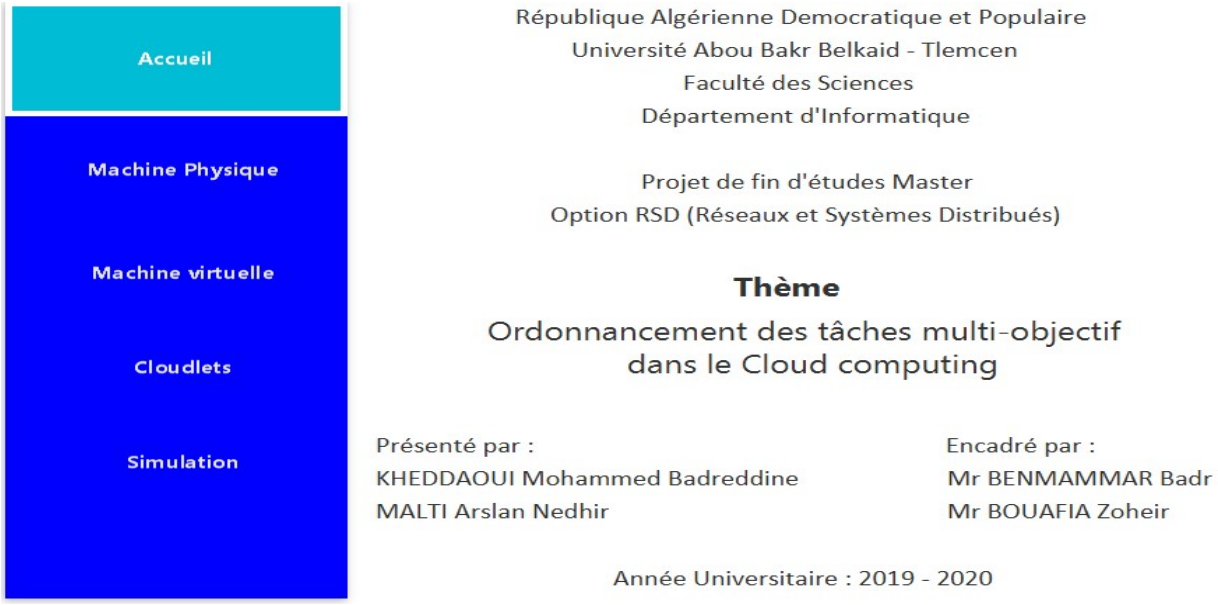


Figure 3.2 : Interface principale.

Configuration des machines physiques :

Pour configurer le Cloud, nous commençons par saisir les paramètres nécessaires aux machines physiques comme le nombre des DataCenters, le nombre des hôtes qui se trouvent dans chaque DataCenter, le nombre de processeur (PE) dans chaque hôte, la vitesse de chaque processeur (MIPS), la RAM, la bande passante et aussi la capacité de stockage.

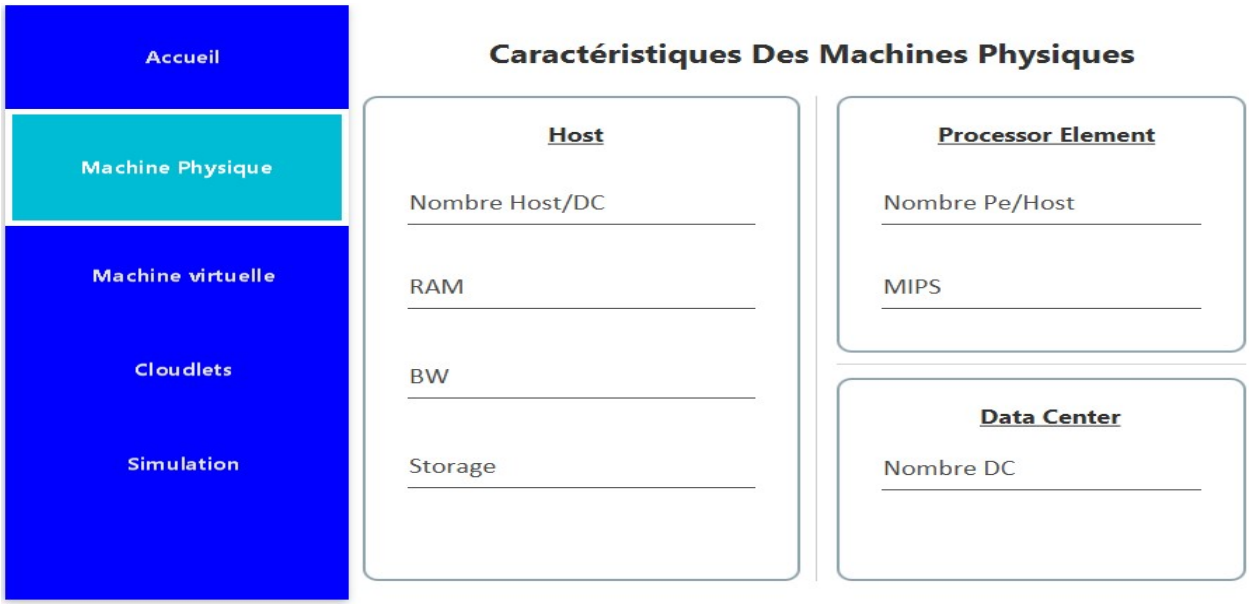


Figure 3.3 : Création des machines physiques.

Configuration des machines virtuelles :

Une fois nous avons configurés les différentes caractéristiques des machines physiques, nous devrons configurer les machines virtuelles VMs, de telle sorte qu'un hôte peut être alloué à un ensemble de machines virtuelles.

Une machine virtuelle est caractérisée par sa vitesse de traitement (MIPS), son coût d'utilisation, son taux d'échec, sa capacité de RAM, sa bande passante ainsi que son stockage comme indiquer dans la figure 3.4.

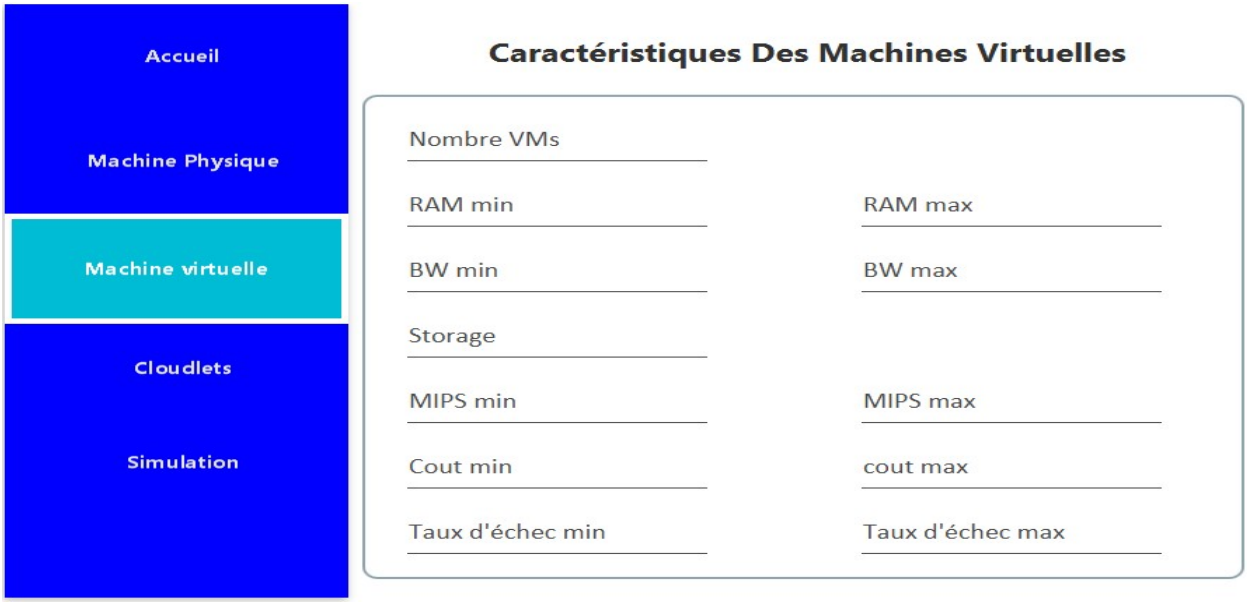


Figure 3.4 : Création des machines virtuelles.

Configuration des Cloudlets :

Cette IHM permet de définir les caractéristiques des cloudlets telle que leurs nombres, leurs longueurs et leur nombre de dépendances. Ce dernier est utilisé afin de généraliser l'ordonnancement des tâches à l'ordonnancement des clusters (groupes de tâches) au sein d'une application.

Exemple : supposons qu'une application contient trois clusters (T1, T2, T3), (T4) et (T5, T6, T7) respectivement:

- T1, T2 et T3 se déroulent en premier temps dans n'importe quel ordre.
- T4 doit se dérouler après la terminaison de T1, T2 et T3.
- Une fois T4 se termine, T5, T6 et T7 peuvent se dérouler dans n'importe quel ordre.

Dans ce cas, le nombre de dépendances pour les 7 tâches est égal à 3.

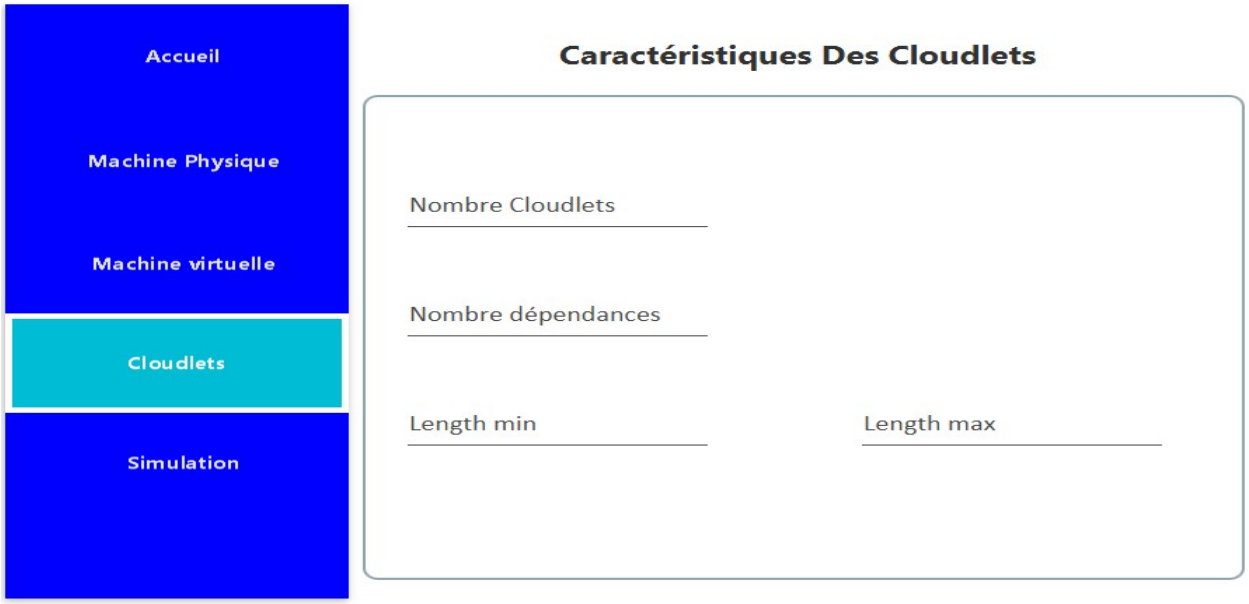


Figure 3.5 : Création des Cloudlets.

Simulation :

A partir de l'interface présentée dans la figure 3.6, l'exécution de la simulation peut être lancée, mais nous devons d'abord choisir le type d'ordonnancement à utiliser pendant la simulation (ordonnancement mono-objectif par rapport au makespan ou ordonnancement multi-objectif par rapport aux 3 métriques (makespan, cout et fiabilité)). La simulation est lancée en cliquant sur le bouton Démarrer.

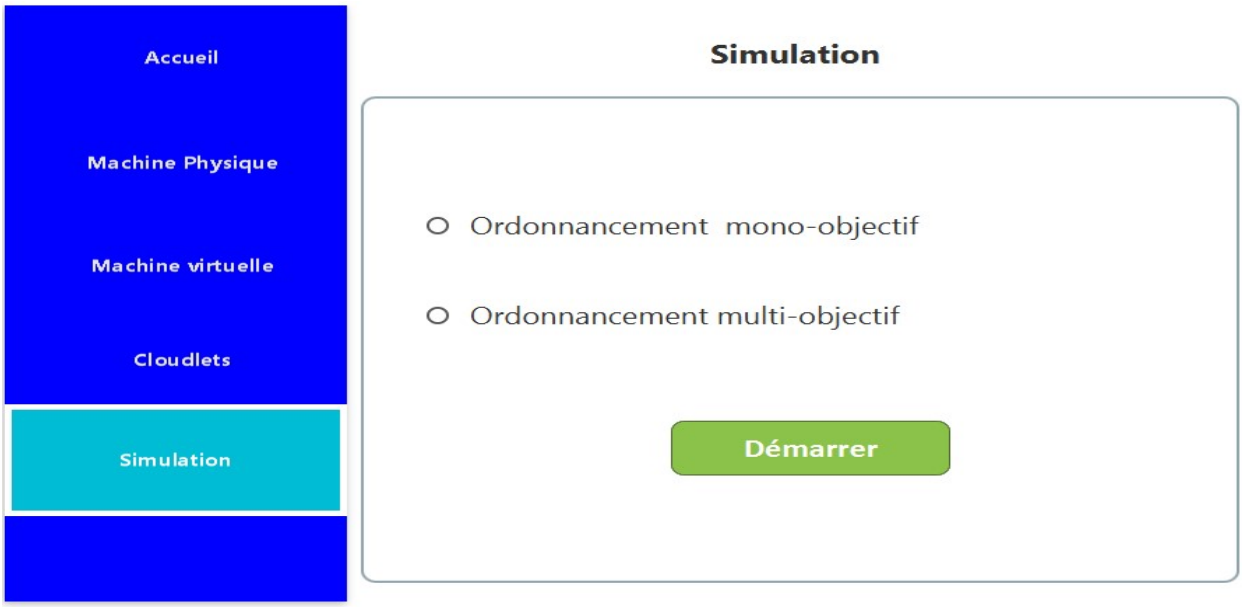


Figure 3.6 : Interface de la simulation.

Après avoir terminé les deux phases de configuration et de pré-simulation, la planification des tâches dans les différentes machines virtuelles est envoyée au broker afin de lancer la simulation et afficher les résultats obtenus.

3.6 Résultats obtenus et étude comparative

Cette section explique la configuration des différentes simulations ainsi que les résultats obtenus après avoir exécuté l'algorithme FPA pour l'ordonnancement des tâches.

Les différentes simulations ont été réalisées sous une machine sous Windows 10 de 64 bits, avec un processeur Intel(R) Core (TM) i5-3320M CPU @ 2.60GHz, une vitesse de 2.60 Ghz et une capacité mémoire de 8 GB.

Concernant FPA, les paramètres à définir sont : la taille de la population et la probabilité de commutation p qui détermine le pourcentage de diversification et d'intensification au cours de la recherche. Pour toutes les simulations, nous avons utilisé une taille de population égale à 20% des machines virtuelles créées (c'est ce pourcentage qui nous a fourni de meilleures performances). Concernant la probabilité de commutation, nous avons pris $p = 0.8$ puisque cette valeur convient à la plupart des applications [22].

Pour le critère d'arrêt, nous avons pris en considération la convergence de l'algorithme FPA.

Les simulations ont été réalisées dans un environnement hétérogène comme indiquer dans le tableau 3.1 définissant la valeur associée à chaque paramètre.

Data center	Nombre	4
Host	Nombre	8
	PES	4
	MIPS	6 000
	RAM	20 GB
	Bande passante	10 GB
	Stockage	1 TB
Machine virtuelle	Nombre	10 – 30 – 50 – 70
	MIPS	1000 jusqu'a 5000
	Coût	1.0 jusqu'a 30.0
	Taux d'échec	10^{-5} jusqu'à 10^{-8}
	RAM	1 GB jusqu'à 5GB
	Bande passante	100 MB jusqu'à 500 MB
	Stockage	10 GB
Cloudlet	Nombre	500 – 1000 – 1500
	Longueur	3000 jusqu'a 10000
	Nombre de dépendances	15
	Type	Hétérogène
	Temps de soumission	Loi de poisson de paramètre λ

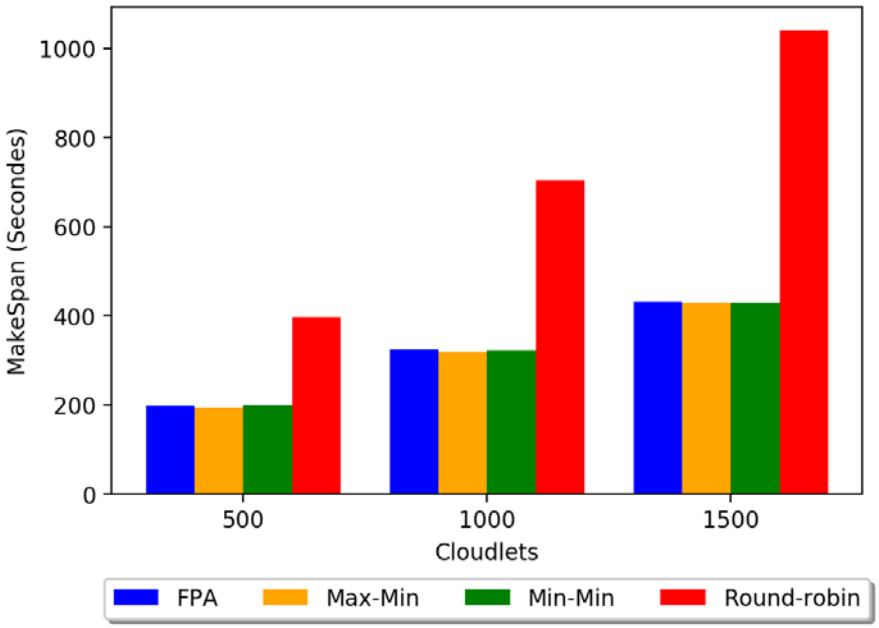
Tableau 3.1 : Les paramètres de simulation de CloudSim.

Etant donné que les algorithmes utilisés ne sont pas déterministes, une seule exécution ne suffit pas pour tirer des conclusions, nous nous sommes donc basés sur le résultat obtenu après une moyenne de 10 simulations pour chaque type d'ordonnancements, pour que les résultats soient plus pertinents.

3.6.1 Evaluation de l'ordonnancement mono-objectif

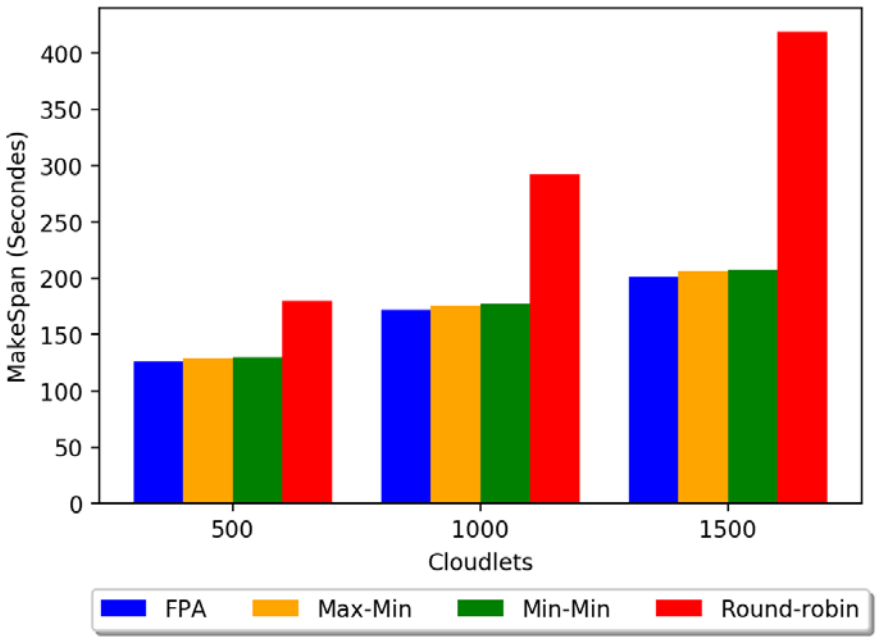
Pour la validation de l'ordonnancement mono-objectif, une comparaison a été réalisée entre FPA (pondération {1, 0, 0} pour évaluer le makespan) et les heuristiques citées précédemment.

Les 4 figures suivantes représentent les résultats de comparaison entre FPA et les heuristiques Max-Min, Min-Min et Round robin par rapport au makespan (mesuré en secondes) pour un nombre de VMs égale à 10, 30, 50 et 70 respectivement.



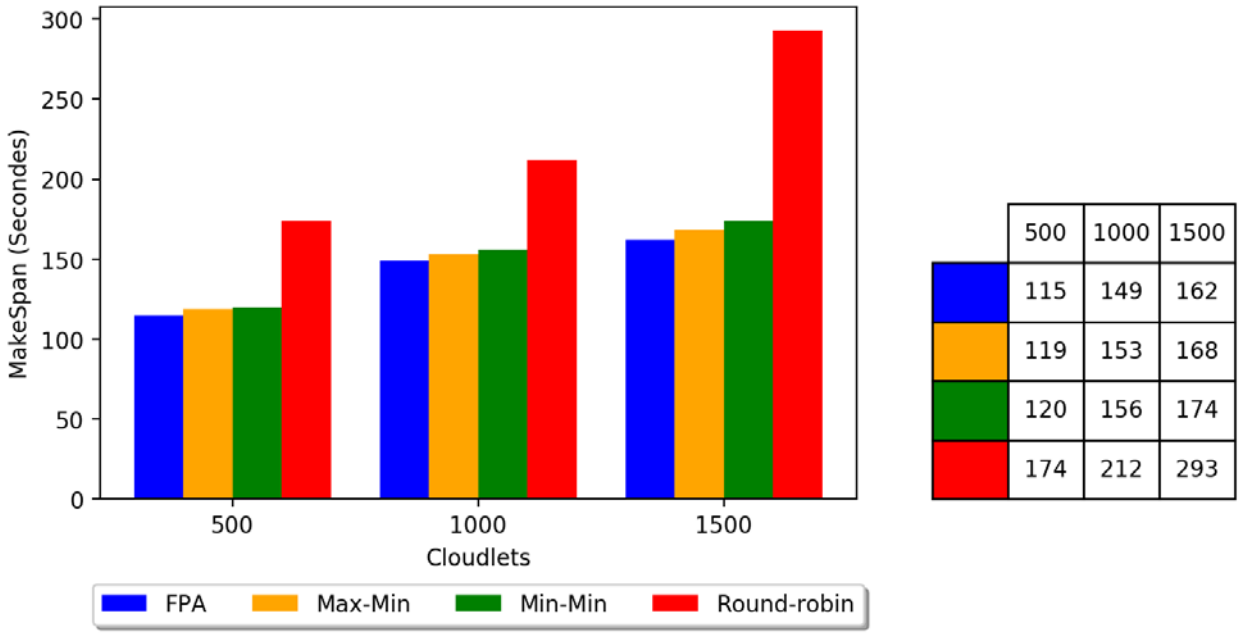
	500	1000	1500
FPA	197	324	432
Max-Min	195	319	429
Min-Min	199	322	430
Round-robin	397	705	1041

Figure 3.7 : Comparaison entre FPA et les heuristiques en termes de makespan pour 10 VMs.



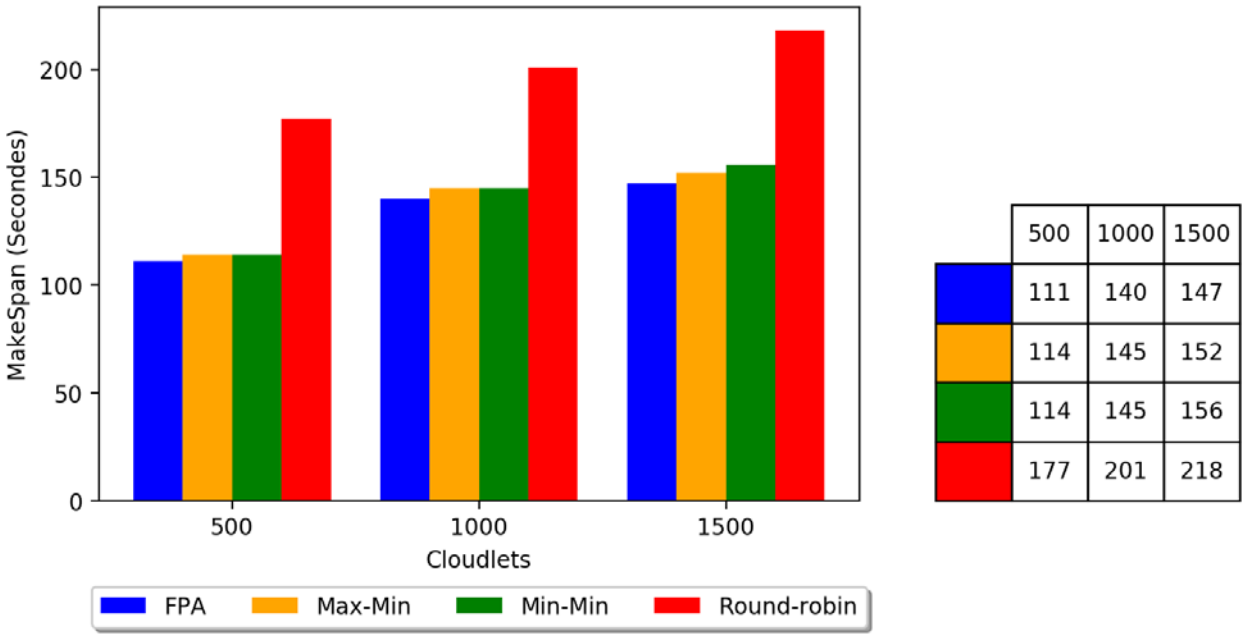
	500	1000	1500
FPA	126	172	201
Max-Min	129	176	206
Min-Min	130	177	208
Round-robin	180	292	419

Figure 3.8 : Comparaison entre FPA et les heuristiques en termes de makespan pour 30 VMs.



	500	1000	1500
FPA	115	149	162
Max-Min	119	153	168
Min-Min	120	156	174
Round-robin	174	212	293

Figure 3.9 : Comparaison entre FPA et les heuristiques en termes de makespan pour 50 VMs.



	500	1000	1500
FPA	111	140	147
Max-Min	114	145	152
Min-Min	114	145	156
Round-robin	177	201	218

Figure 3.10 : Comparaison entre FPA et les heuristiques en termes de makespan pour 70 VMs.

Résultat : Max-Min et Min-Min sont meilleurs légèrement par rapport à FPA pour un nombre de VMs égal à 10. Lors du passage à l'échelle (nombre de Vms et de tâches plus important), le FPA a surpassé tous les autres algorithmes (Max-Min, Min-Min et Round robin). A titre d'exemple, pour 70 Vms et 1500 tâches, le FPA a surpassé de 9 secondes l'algorithme Min-Min. Pour 50 Vms et 1500 tâches, le FPA a surpassé de 12 secondes l'algorithme Min-Min. Le Round robin quant à lui est connu pour ne pas être scalable. Nous pouvons affirmer donc que FPA a réussi à garantir une

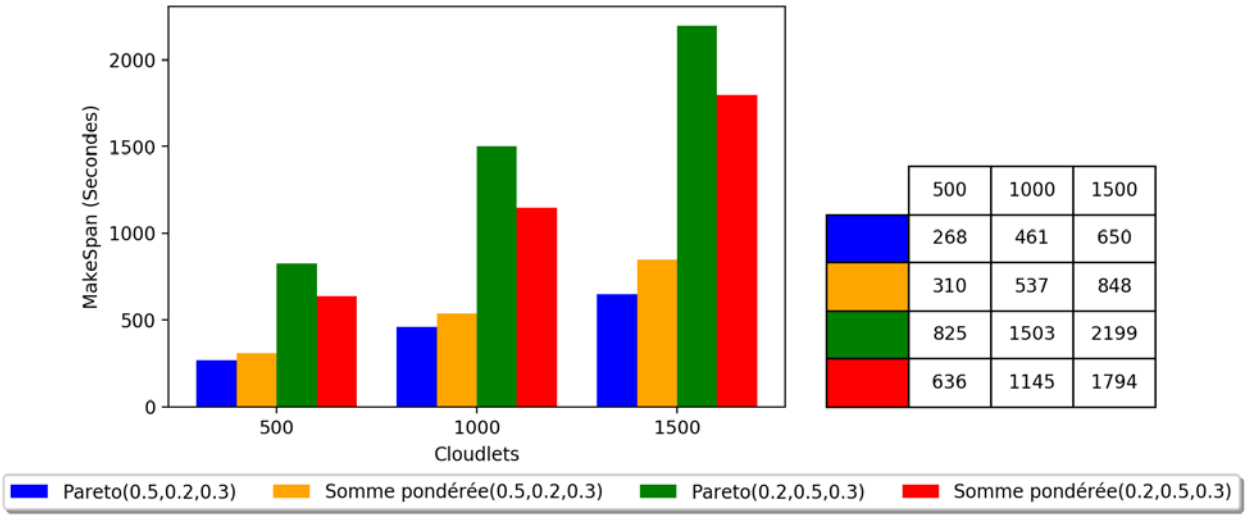
meilleure qualité de service en termes de makespan peu importe l'échelle en se comparant par rapport aux autres heuristiques.

3.6.2 Evaluation de l'ordonnancement multi-objectif

Pour la validation de l'ordonnancement multi-objectif, une autre comparaison a été réalisée en utilisant deux versions pour l'évaluation des solutions dans l'algorithme FPA, à savoir le principe de Pareto basé sur la méthode TOPSIS et la méthode de la somme pondérée en agrégeant les différentes métriques de QoS selon la fonction objectif définie précédemment (équation 3.13).

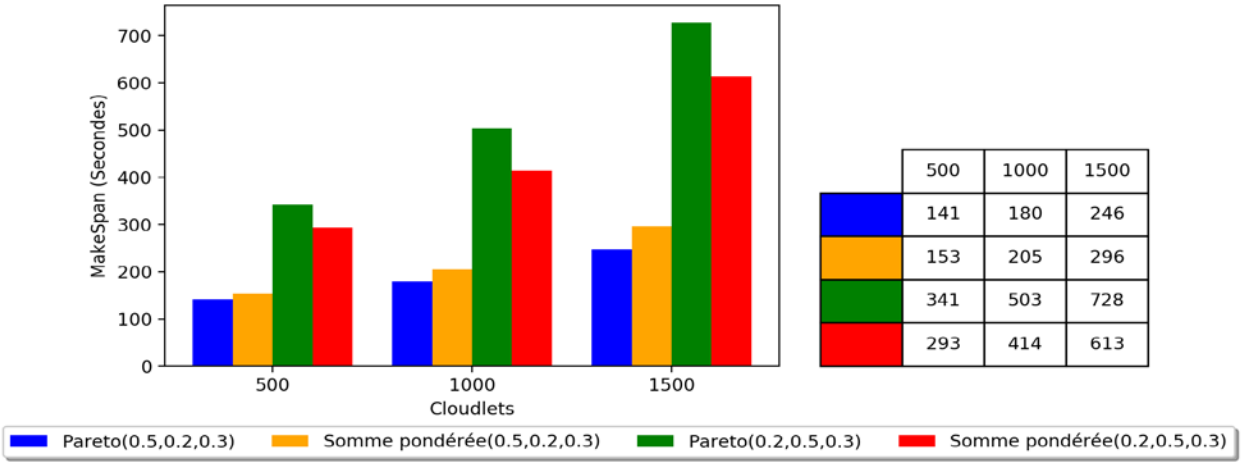
Pour les deux versions, nous avons utilisé deux vecteurs de poids différents $\{0.5, 0.2, 0.3\}$ et $\{0.2, 0.5, 0.3\}$ correspondant respectivement aux critères $\{Makespan, Coût, Fiabilité\}$. Le premier vecteur est utilisé pour favoriser le makespan, le second vecteur est utilisé pour favoriser le cout.

3.6.2.1 Comparaison par rapport au makespan



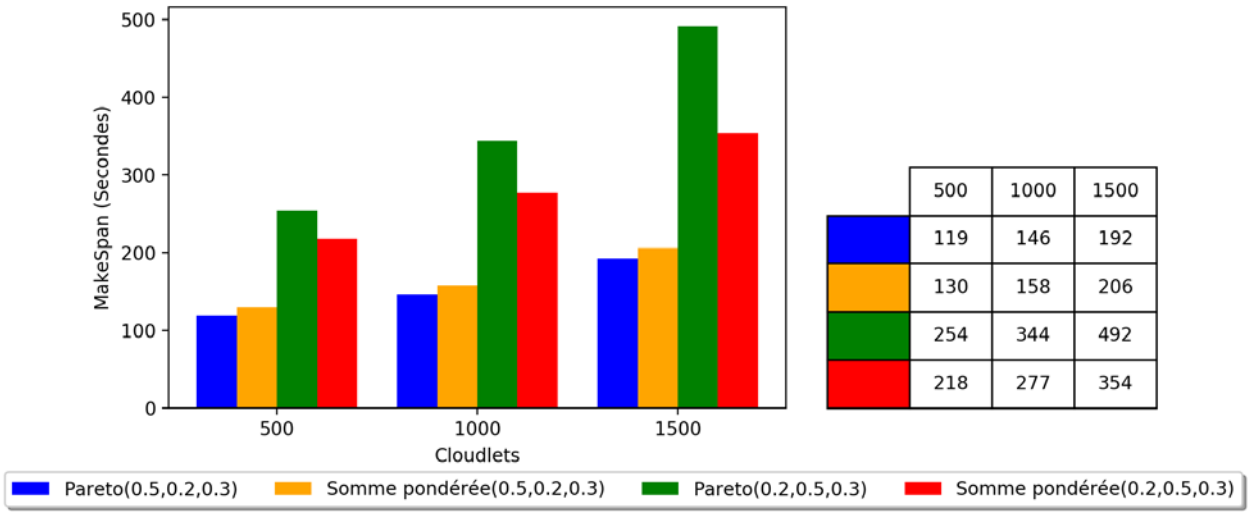
	500	1000	1500
Pareto(0.5,0.2,0.3)	268	461	650
Somme pondérée(0.5,0.2,0.3)	310	537	848
Pareto(0.2,0.5,0.3)	825	1503	2199
Somme pondérée(0.2,0.5,0.3)	636	1145	1794

Figure 3.11 : Comparaison en termes de makespan pour 10 VMs.



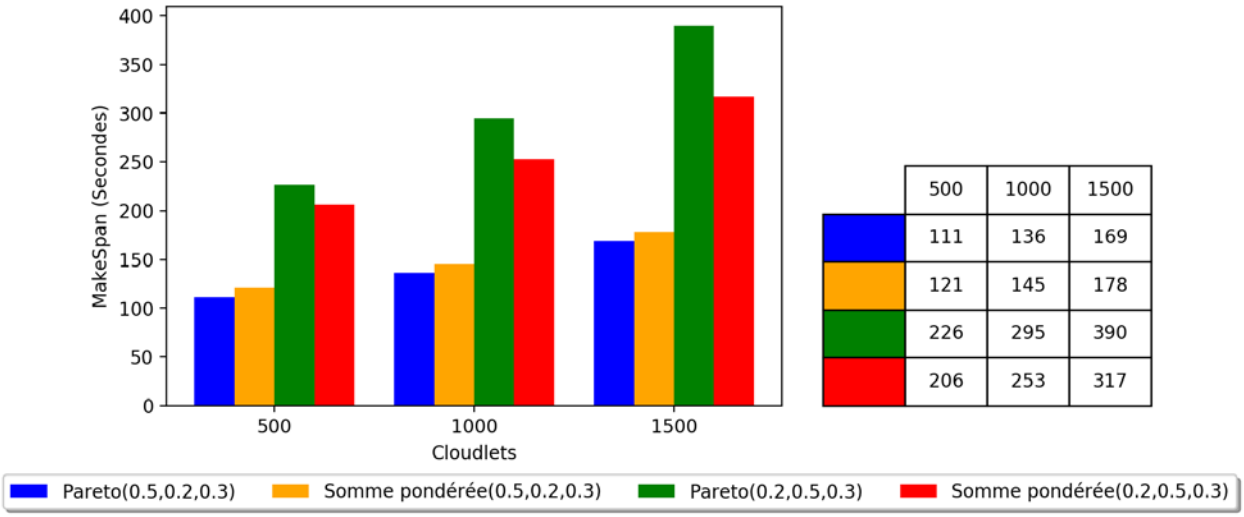
	500	1000	1500
Pareto(0.5,0.2,0.3)	141	180	246
Somme pondérée(0.5,0.2,0.3)	153	205	296
Pareto(0.2,0.5,0.3)	341	503	728
Somme pondérée(0.2,0.5,0.3)	293	414	613

Figure 3.12 : Comparaison en termes de makespan pour 30 VMs.



	500	1000	1500
Pareto(0.5,0.2,0.3)	119	146	192
Somme pondérée(0.5,0.2,0.3)	130	158	206
Pareto(0.2,0.5,0.3)	254	344	492
Somme pondérée(0.2,0.5,0.3)	218	277	354

Figure 3.13 : Comparaison en termes de makespan pour 50 VMs.

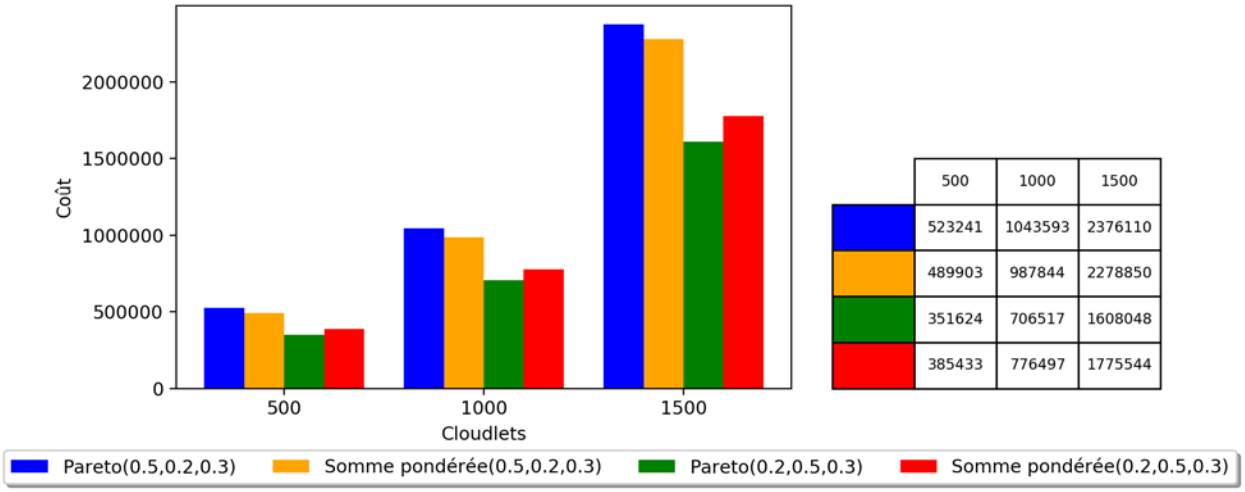


	500	1000	1500
Pareto(0.5,0.2,0.3)	111	136	169
Somme pondérée(0.5,0.2,0.3)	121	145	178
Pareto(0.2,0.5,0.3)	226	295	390
Somme pondérée(0.2,0.5,0.3)	206	253	317

Figure 3.14 : Comparaison en termes de makespan pour 70 VMs.

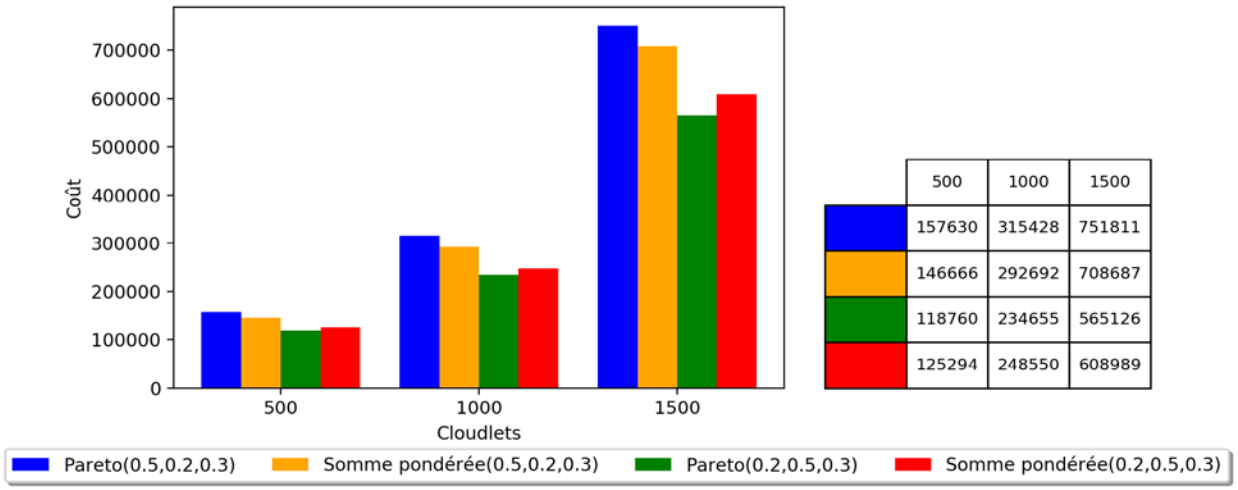
Résultat : peu importe le nombre de VMs et le nombre de tâches**,** la meilleure façon d'optimiser le makespan est d'utiliser le Pareto/TOPSIS avec le vecteur de poids $\{0.5, 0.2, 0.3\}$ qui favorise le makespan. Cette dernière configuration surpasse largement le Pareto/TOPSIS et la somme pondérée avec la pondération $\{0.2, 0.5, 0.3\}$ qui favorise le cout mais aussi a surpassé la somme pondérée avec la pondération $\{0.5, 0.2, 0.3\}$ qui favorise pourtant le makespan.

3.6.2.2 Comparaison par rapport au coût



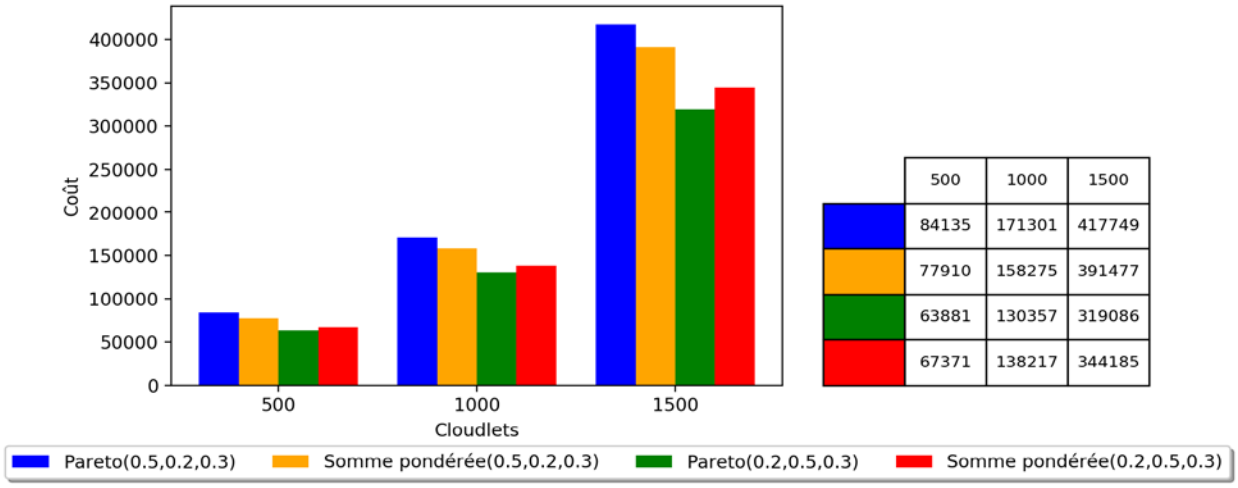
	500	1000	1500
Pareto(0.5,0.2,0.3)	523241	1043593	2376110
Somme pondérée(0.5,0.2,0.3)	489903	987844	2278850
Pareto(0.2,0.5,0.3)	351624	706517	1608048
Somme pondérée(0.2,0.5,0.3)	385433	776497	1775544

Figure 3.15 : Comparaison en termes de coût pour 10 VMs.



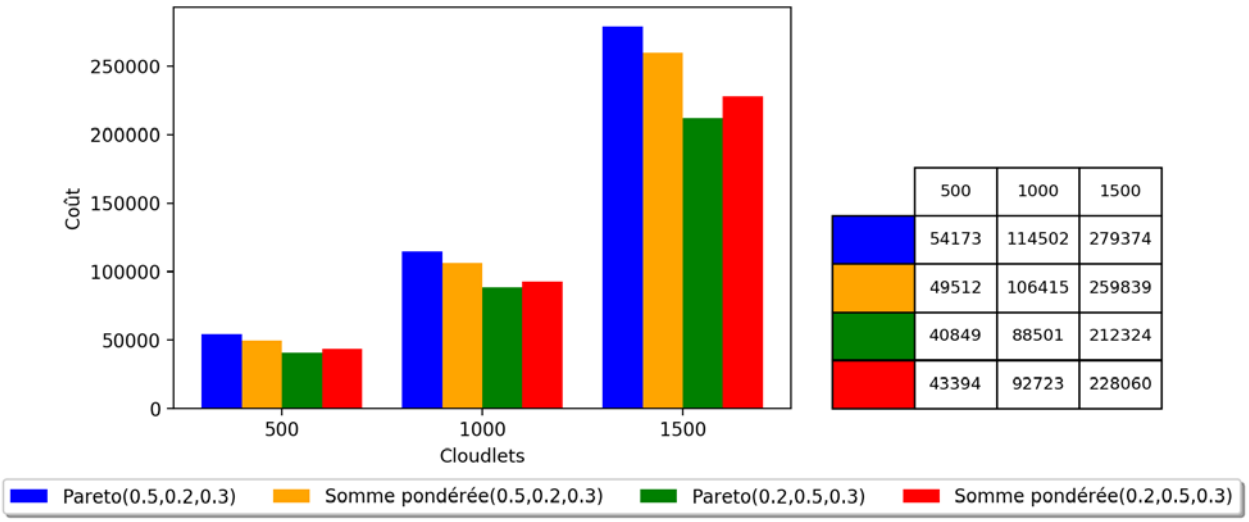
	500	1000	1500
Pareto(0.5,0.2,0.3)	157630	315428	751811
Somme pondérée(0.5,0.2,0.3)	146666	292692	708687
Pareto(0.2,0.5,0.3)	118760	234655	565126
Somme pondérée(0.2,0.5,0.3)	125294	248550	608989

Figure 3.16 : Comparaison en termes de coût pour 30 VMs.



	500	1000	1500
Pareto(0.5,0.2,0.3)	84135	171301	417749
Somme pondérée(0.5,0.2,0.3)	77910	158275	391477
Pareto(0.2,0.5,0.3)	63881	130357	319086
Somme pondérée(0.2,0.5,0.3)	67371	138217	344185

Figure 3.17 : Comparaison en termes de coût pour 50 VMs.



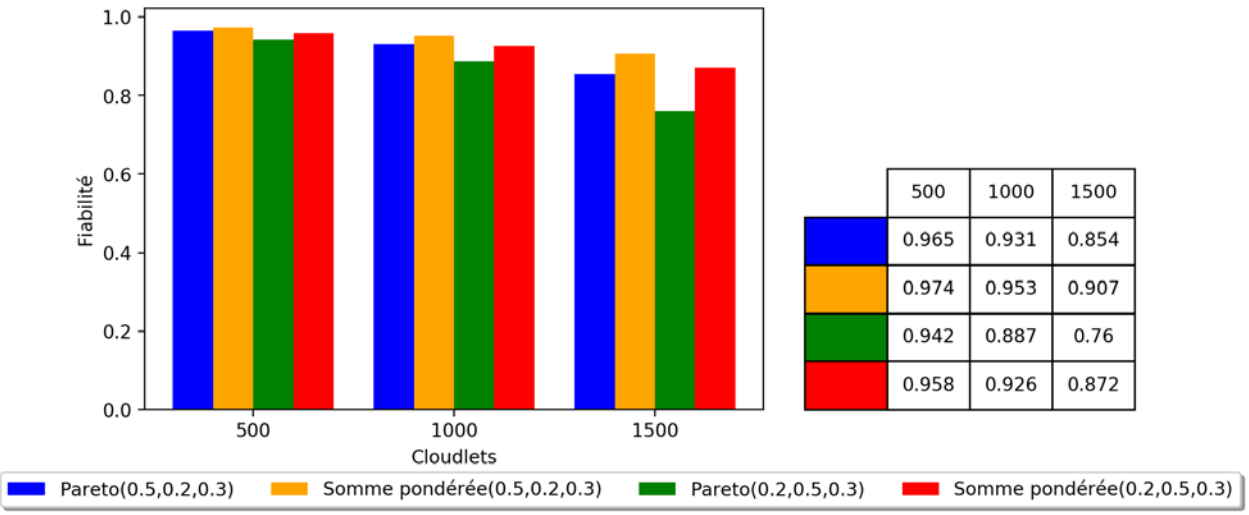
	500	1000	1500
Pareto(0.5,0.2,0.3)	54173	114502	279374
Somme pondérée(0.5,0.2,0.3)	49512	106415	259839
Pareto(0.2,0.5,0.3)	40849	88501	212324
Somme pondérée(0.2,0.5,0.3)	43394	92723	228060

Figure 3.18 : Comparaison en termes de coût pour 70 VMs.

Résultat : peu importe le nombre de VMs et le nombre de tâches**,** la meilleure façon d'optimiser le cout est d'utiliser le Pareto/TOPSIS avec le vecteur de poids $\{0.2, 0.5, 0.3\}$ qui favorise le cout. Cette dernière configuration surpasse largement le Pareto/TOPSIS et la somme pondérée avec la pondération $\{0.5, 0.2, 0.3\}$ qui favorise le makespan mais aussi a surpassé la somme pondérée avec la pondération $\{0.2, 0.5, 0.3\}$ qui favorise pourtant le cout.

Nous remarquons aussi que le cout d'exécution s'accroît lorsque le makespan diminue, une chose qui est logique vu que le coût d'exécution est inversement proportionnel au temps d'exécution.

3.6.2.3 Comparaison par rapport à la fiabilité



	500	1000	1500
Pareto(0.5,0.2,0.3)	0.965	0.931	0.854
Somme pondérée(0.5,0.2,0.3)	0.974	0.953	0.907
Pareto(0.2,0.5,0.3)	0.942	0.887	0.76
Somme pondérée(0.2,0.5,0.3)	0.958	0.926	0.872

Figure 3.19 : Comparaison en termes de fiabilité pour 10 VMs.

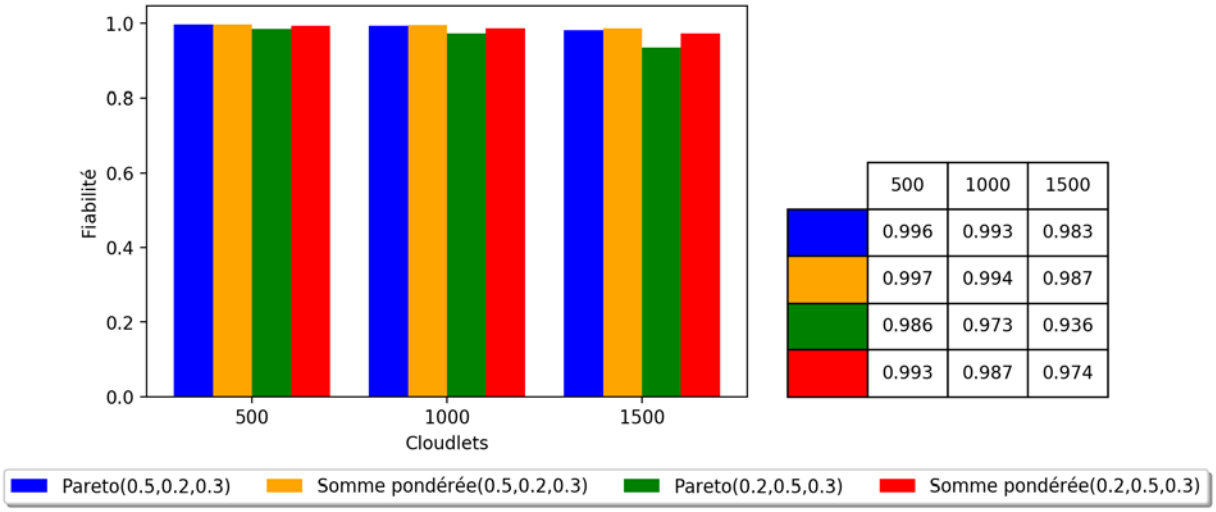


Figure 3.20 : Comparaison en termes de fiabilité pour 30 VMs.

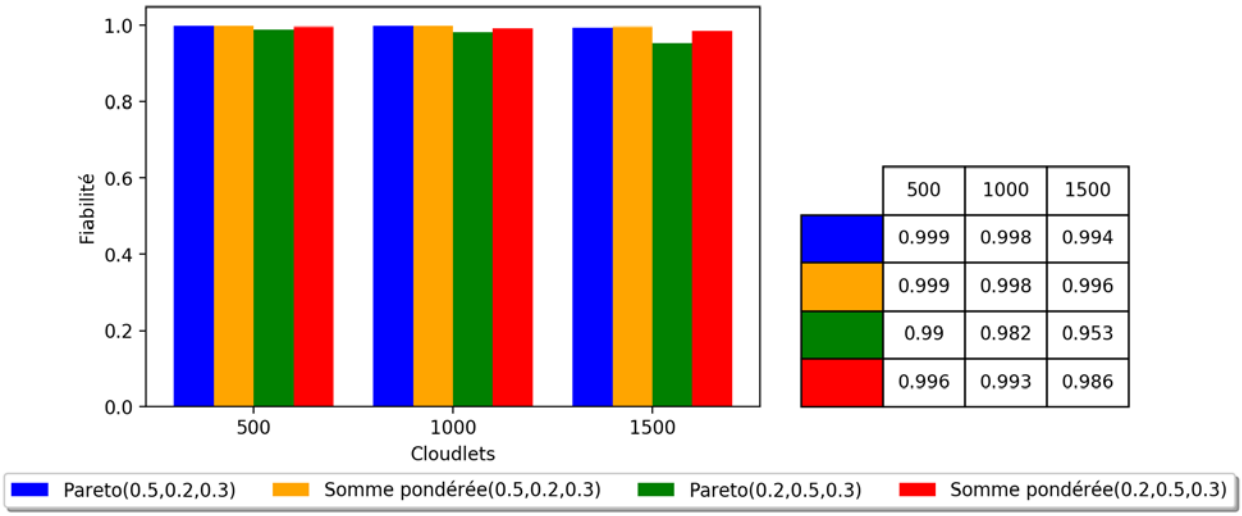


Figure 3.21 : Comparaison en termes de fiabilité pour 50 VMs.

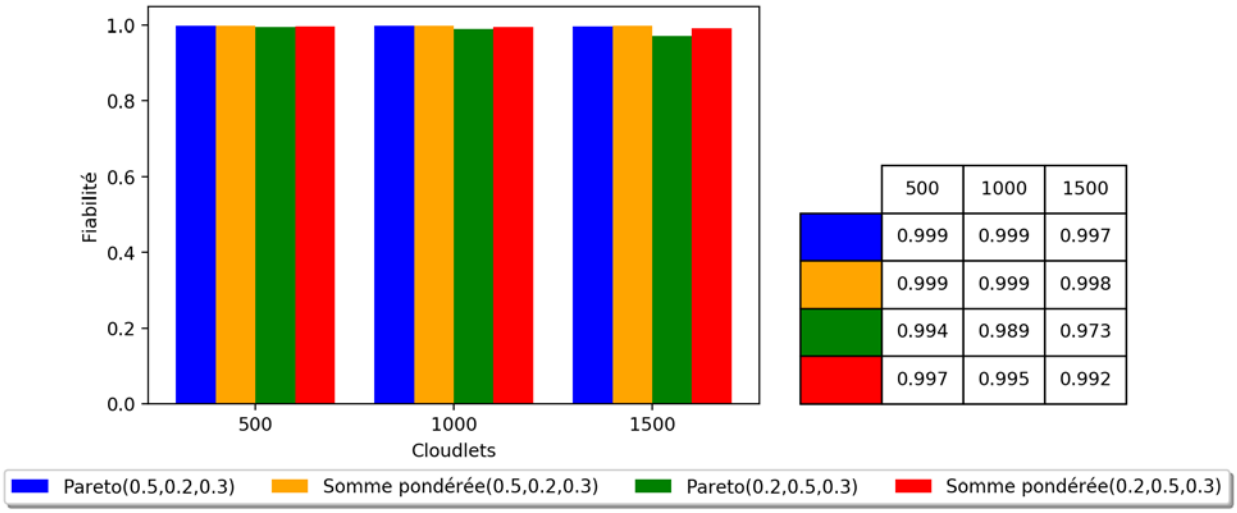


Figure 3.22 : Comparaison en termes de fiabilité pour 70 VMs.

Résultat : peu importe le nombre de VMs et le nombre de tâches**,** nous nous apercevons que la différence entre les deux méthodes en termes de fiabilité requise n'est pas flagrante ou presque similaire.

Résultat global : concernant les deux métriques makespan et cout, le mieux est d'utiliser Pareto/TOPSIS avec le vecteur de poids adapté. En effet, la somme pondérée est une simple projection du multi-objectif vers le mono-objectif. Une approche purement multi-objectif comme le Pareto/TOPSIS est plus recommandée dans ce contexte. Pour la fiabilité, il s'agit d'une métrique un peu particulière car dans le Cloud, les machines virtuelles sont considérées comme fiables puisque les fournisseurs Cloud existants proposent des services avec un taux d'échec trop petit qui peut aller jusqu'à 10^{-7} ou 10^{-8}. C'est pour cette raison que la méthode utilisée n'a pas beaucoup d'impact sur cette métrique.

3.7 Conclusion

Ce chapitre a été consacré à la présentation de notre contribution dans le cadre de ce PFE. Nous avons présenté les outils de simulations utilisés, à savoir, le langage JAVA, l'IDE NetBeans, l'outil Jfreechart ainsi que le simulateur CloudSim. Nous avons présenté également quelques notions fondamentales pour l'ordonnancement multi-objectif.

Pour le mono-objectif, les résultats obtenus des différentes simulations effectuées montrent que notre algorithme FPA surpasse les heuristiques classiques comme Max-Min, Min-Min et Round robin notamment lors du passage à l'échelle et ceci, peu importe le nombre de cloudlets.

La comparaison entre les deux versions de la fonction objectif pour l'ordonnancement multi-objectif a mis en lumière tant de points positifs. L'évaluation par la méthode TOPSIS selon le principe de Pareto est plus crédible par rapport à l'agrégation en particulier pour le critère favorisé (le makespan ou le coût dans notre cas). Bien que chaque évaluation soit différente, on retrouve néanmoins une similitude entre-elles en ce qui concerne la fiabilité qui a des performances approximativement similaires.

Conclusion générale

Le Cloud computing est une avancée récente caractérisée par l'utilisation d'un grand pool de ressources informatiques accessibles à la demande via Internet. La plupart de ces services sont sous forme de tâches dépendantes. Par conséquent, cela nécessite des techniques d'ordonnancement plus élaborées pour ordonnancer les tâches et de les mapper sur les machines virtuelles appropriées, afin de répondre aux exigences de QoS spécifiées dans les contrats de service. D'autre part, les utilisateurs sont confrontés au problème du choix des meilleures ressources à utiliser pour répondre à leurs exigences. Ce problème est considéré comme étant un problème d'optimisation multi-objectif en raison de la nature conflictuelle des métriques de QoS à prendre en considération.

Dans notre étude, nous avons adapté une métaheuristique pour l'optimisation multi-objectif du problème de l'ordonnancement dans le domaine de Cloud computing. Il s'agit de la métaheuristique de pollinisation des fleurs (FPA) permettant de trouver la solution optimale en un temps raisonnable en fonction de trois métriques de QoS qui sont le makespan, le coût et la fiabilité.

Pour valider notre contribution, nous avons traité deux scénarios en fonction du nombre de paramètres impliqués pour mesurer la QoS. Dans le premier scénario, nous avons traité le problème de l'ordonnancement mono-objectif par rapport à un seul critère, qui est le makespan. Les résultats obtenus ont été comparés avec un algorithme classique déjà implémenté dans CloudSim qui est le Round Robin et deux autres heuristiques qui sont Max-Min et Min-Min.

Dans le second scénario, nous avons étudié l'ordonnancement multi-objectif par rapport à 3 critères qui sont le makespan, le coût et la fiabilité. Nous avons opté également pour deux approches pour mesurer la fonction objectif, à savoir l'agrégation et le Pareto basé sur TOPSIS pour sélectionner la solution optimale du front de Pareto.

Dans l'approche par agrégation, nous avons mappé le problème d'ordonnancement multi-objectif en un problème mono-objectif en agrégeant les différentes métriques de QoS avec des préférences définies à priori. Tandis que, dans l'approche Pareto, nous avons traité le problème d'ordonnancement multi-objectif en utilisant le principe de Pareto afin de satisfait au mieux toutes les métriques de QoS selon les résultats obtenus par la méthode TOPSIS.

A travers les différentes simulations réalisées avec CloudSim, nous avons constaté la supériorité de FPA en mono-objectif par rapport à Round Robin, Max-Min et Min-Min en particulier lors du passage à l'échelle. En multi-objectif, les résultats obtenus sont plus intéressants pour TOPSIS selon le principe de Pareto en les comparant avec l'agrégation notamment pour le critère favorisé, afin d'offrir une meilleure QoS aux utilisateurs selon leurs besoins.

Suite à la réalisation de notre étude, il serait intéressant d'étendre le problème d'optimisation à d'autres métriques de QoS (énergie, équilibrage de charge, tolérance aux pannes, …). Introduire une nouvelle notion d'optimalité à part Pareto et l'agrégation. Appliquer d'autres métaheuristiques comme les algorithmes génétiques, l'optimisation par essaims particulaires ou cuckoo search afin de comparer avec les résultats obtenus par le FPA.

Bibliographie

[1] Bourmaki Imane, Mohammedi Meriem. Algorithme d'optimisation multi-objectifs dans le Cloud computing. Mémoire de Master en Informatique. Université Abou Bakr Belkaid Tlemcen. 2019.

[2] ARIF, Mohamed. A history of cloud computing. Disponible sur: www.computerweekly.com/feature/A-history-of-cloud-computing. Consulté le 29/04/2020.

[3] Benghelima Mohamed Amine, Inal Mohammed Taha. Développement d'un algorithme de collecte d'informations de charge basé sur la technique de calcul par sphère. Mémoire de Master en Informatique. Université Abou Bakr Belkaid Tlemcen. 2016.

[4] Les inconvénient du Cloud computing. Disponible sur: https://missarte.wordpress.com/lesinconvenients-du-cloud-computing. Consulté le 30/04/2020.

[5] Qu'est-ce que la virtualisation?. Disponible sur: https://www.supinfo.com/articles/single/5969-qu-est-ce-que-virtualisation. Consulté le 03/05/2020.

[6] VIRTUALISATION – Definition –. Disponible sur: https://abtec.fr/virtualisation-definition/. Consulté le 03/05/2020.

[7] Architecture traditionnelle vs Architecture virtuelle. Disponible sur : https://www.redswitches.com/blog/different-types-virtualization-cloud-computing-explained/. Consulté le 03/05/2020.

[8] Définition Data Center : qu'est-ce qu'un centre de données ? Disponible sur: https://www.lebigdata.fr/definition-data-center-centre-donnees. Consulté le 02/05/2020.

[9] Esma Insaf Djebbar. Optimisation d'ordonnancement et d'allocation de ressources dans les clouds computing. Thèse de doctorat en informatique. Université d'Oran 1 Ahmed Ben Bella. 2016.

[10] Fred van der Molen. Get ready for Cloud Computing: A comprehensive guide to Virtualization and Cloud Computing. December 13, 2010. Van Haren Publishing; (December 13, 2010). 182 pages.

[11] Meslem yamina, Debbas soumia ismahèn. Ordonnancement et réplication de données dans le Cloud Computing. Mémoire de Master en informatique. Université Dr Tahar Moulay SAIDA. 2017.

[12] Qu'est-ce que le SaaS (Software as a Service) ? Disponible sur : http://www.interoute.fr/what- saas. Consulté le 28/04/2020.

[13] Les trois couches du Cloud Computing. Disponible sur : https://www.uniprint.net/fr/7-types-cloud-computing-structures/. Consulté le 28/04/2020

[14] Naziha Ali Saoucha. Exploitation des techniques de l'intelligence artificielle pour l'optimisation de la QoS et l'efficacité spectrale dans les réseaux de radio cognitive. Thèse de doctorat en Informatique. Université de Tlemcen. 2020.

[15] Gherboudj, Amira. "Méthodes de résolution de problèmes difficiles académiques." *Université de Constantine 2*. Rapport de Doctorat 3ème cycle LMD en Informatique. 2013.

[16] Autin, Baptiste. "Les métaheuristiques en optimisation combinatoire." Mémoire examen probatoire en informatique. Conservatoire Nationales des Arts et des Métiers Paris. 2006.

[17] Sidi Mohamed Douiri, Souad Elbernoussi, Halima Lakhbab. "Cours des méthodes de résolution exactes heuristiques et métaheuristiques." *Université Mohamed V, Faculté des sciences de Rabat* (2009): 5-87.

[18] Les métaheuristiques. Disponible sur le lien : http://www.gerad.ca/~alainh/Metaheuristiques.pdf . Consulté le 01/05/2020.

[19] Ferhat, Halima. *Optimisation de la QoS dans un réseau de radio cognitive en utilisant l'algorithme de la recherche par harmonie*. Rapport de Master en Informatique. Université de Tlemcen. 2015.

[20] Yang, Xin-She, and Suash Deb. "Cuckoo search: state-of-the-art and opportunities." *2017 IEEE 4th International Conference on Soft Computing & Machine Intelligence (ISCMI 2017)*. IEEE publisher, 22-24 Nov 2017. Republic of Mauritius. p. 55-59.

[21] Jayabarathi, T., T. Raghunathan, and A. H. Gandomi. "The bat algorithm, variants and some practical engineering applications: A review." *Nature-Inspired Algorithms and Applied Optimization*. Studies in Computational Intelligence, Springer, Cham, Vol. 744. 313-330. 2018

[22] Yang, Xin-She. "Flower pollination algorithm for global optimization." *International conference on unconventional computing and natural computation*. Springer, Berlin, Heidelberg, 2012.

[23] Abdel-Basset, Mohamed, and Laila A. Shawky. "Flower pollination algorithm: a comprehensive review." *Artificial Intelligence Review*. Vol. 52, No 4. 2533-2557. 2019

[24] Chittka, Lars, James D. Thomson, and Nickolas M. Waser. "Flower constancy, insect psychology, and plant evolution." *Naturwissenschaften.* Vol. 86, No 8. 361-377. 1999.

[25] Waser, Nickolas M. "Flower constancy: definition, cause, and measurement." *American Naturalist*. Vol. 127, No. 5. 593-603. 1986.

[26] Pavlyukevich, Ilya. "Lévy flights, non-local search and simulated annealing." *Journal of Computational Physics*. Vol. 226. No. 2.1830-1844. 2007.

[27] Tong, Zhijun, Richard Y. Kain, Wei-Tek Tsai. Alow overhead checkpointing and rollback recovery scheme for distributed systems. In Proceedings of the 8th Symposium on Reliable Distributed Systems, SRDS, pages 12–20, Seattle, Washington, USA, October 10-12, 1989.

[28] Gosling, J., Joy, B., Steele, G., Bracha, G., Buckley, A. The Java® Language Specification-Java SE 8 Edition (2015). *Retrieved August*, 2015, vol. 8, p. 2017.

[29] Documentation netbeans. Disponible sur le lien : http://doc.ubuntu-fr.org/netbeans. Consulté le 16/05/2020.

[30] The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne. CloudSim : A Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. Disponible sur le lien : http://www.cloudbus.org/cloudsim/. Consulté le 16/05/2020.

[31] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., Buyya, R. CloudSim : a Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, Software : Practice and Experience, vol. 41, no. 1, p. 23-50.

[32] Benmammar, Badr, Youcef Benmouna, and Francine Krief. "A Pareto optimal multi-objective optimisation for parallel dynamic programming algorithm applied in cognitive radio ad hoc networks." *International Journal of Computer Applications in Technology* 59.2 (2019): 152-164.

[33] Newman, T. R., Barker, B. A., Wyglinski, A. M., Agah, A., Evans, J. B., Minden, G. J. "Cognitive engine implementation for wireless multicarrier transceivers." *Wireless communications and mobile computing* 7.9 (2007): 1129-1142.

[34] Hwang, Ching-Lai, and Kwangsun Yoon. "Methods for multiple attribute decision making." *Multiple attribute decision making*. Springer, Berlin, Heidelberg, 1981. 58-191.

[35] Yassa Sonia. Allocation optimale multicontraintes des workflows aux ressources d'un environnement Cloud Computing. Thèse de doctorat en Sciences et Technologie de l'Information et de la Communication STIC. Université de Cergy-Pontoise. 2014.

Résumé :

L’ordonnancement de tâches est un enjeu important qui influence considérablement les performances de l'environnement de Cloud computing. Les fournisseurs et les utilisateurs des services Cloud ont des objectifs et des exigences conflictuels. Un bon ordonnanceur doit offrir un compromis acceptable entre ces objectifs. Ainsi, l’ordonnancement de tâches dans le Cloud devient un problème d’optimisation multi objectif. Notre contribution dans le cadre de ce travail consiste à traiter le problème de l'ordonnancement des tâches dans le Cloud computing afin de trouver la meilleure affectation des tâches à l’ensemble des machines virtuelles. La métaheuristique FPA a été utilisée en évaluant sa fonction objectif avec trois métriques de QoS qui sont, le makespan, le cout et la fiabilité. Les simulations et les évaluations ont été réalisées à l’aide de CloudSim et les résultats obtenus sont très satisfaisants.

Mots clés: Cloud computing, Optimisation multi-objectif, QoS, FPA, CloudSim.

Abstract:

Task scheduling is an important factor that greatly influences the performance of the Cloud computing environment. Cloud service providers and users have conflicting objectives and requirements. A good scheduler must offer an acceptable compromise between these objectives. Thus, the Cloud tasks scheduling becomes a multi-objective optimization problem. Our contribution through this work consists in dealing with the problem of task scheduling in Cloud computing in order to find the best assignment of tasks to all virtual machines. The FPA metaheuristics was used by evaluating its objective function with three QoS metrics which are, the makespan, the cost and the reliability. The simulations and evaluations were carried out using CloudSim and the results obtained are very satisfactory.

Key words: Cloud computing, Multi-objective optimization, QoS, FPA, CloudSim.

ملخص:

جدولة المهام هي عامل مهم تؤثر بشكل كبير على أداء بيئة الحوسبة السحابية. لدى مقدمي الخدمات السحابية والمستخدمين أهداف ومتطلبات متضاربة. يجب على المجدول الجيد تنفيذ حل وسط مقبول بين هذه الأهداف. وبالتالي، تصبح جدولة المهام في السحابة مشكلة تحسين متعدد الأهداف. الهدف من هذا العمل هو معالجة مشكلة جدولة المهام في الحوسبة السحابية من أجل العثور على أفضل تعيين مهام للأجهزة الافتراضية باستخدام FPA métaheuristique المطبق على مقاييس جودة الخدمة الثلاثة التالية: وقت التنفيذ والتكلفة والموثوقية. نفذت عمليات المحاكاة والتقييم باستخدام CloudSim والنتائج التي تم الحصول عليها مرضية للغاية.

الكلمات المفتاحية: الحوسبة السحابية، التحسين متعدد الأهداف، مقاييس جودة الخدمة، FPA، CloudSim.