**République Algérienne Démocratique et Populaire**

**Ministère de l'Enseignement Supérieur et de la**

**Recherche Scientifique Université 20 Aout 1955-Skikda**

**Faculté de Sciences - Département Informatique**

**Mémoire de fin d'études pour l'obtention du diplôme de Master en Informatique**

**Option : Systèmes Informatiques**

# Thème

*L'algorithme d'optimisation de fourrage des Raies Manta pour le placement des machines virtuelles des data center des cloud*

**Réalisé par :**

BELMEGUENAI Yasmine

**Encadré par :**

Mr. LAOUAR Walid

Mr. BOUAITA Riad

**Session : 2022**

# Remerciement

*On remercie Dieu le tout puissant de nous avoir donne la sante et la volonté d'entamer et de terminer ce mémoire*

*Tout d'abord, ce travail ne serait pas aussi riche et n'aurait pas pu avoir le jour sans l'aide et l'encadrement de Mr W. LAOUAR et Mr R. BOUAITA, on le remercie pour la qualité de leur encadrements exceptionnel, pour leur patience, leurs rigueurs, et leur disponibilité durant notre préparation de ce mémoire.*

*Nous remerciement s'adresse également à tout nos professeurs pour leurs générosités et la grande patience dont ils ont sa faire preuve malgré leurs charges académiques et professionnelles.*

*Enfin, nos remerciements aux membres de jury qui ont fait l'honneur d'accepter de juger ce modeste travail.*

# Dédicace

*Je tiens c'est avec grande plaisir que je dédie ce modeste travail :*

*A l'être le plus cher de ma vie, ma mère.*

*A celui qui m'a fait de moi une femme, mon père.*

*A mon chers frère et sœur.*

*A la petite princesse Amani*

*A tous mes amis de promotion de 2éme année Master Systèmes Informatiques, toute personne qui occupe une place dans mon cœur.*

*A tous les membres de ma famille, je dédie ce travail a tous ceux qui ont participe à ma réussie.*

# Résumé

Le Cloud Computing est une technologie qui fournit aux utilisateurs un accès aux capacités de traitement et de stockage depuis leurs ordinateurs. Ces capacités constituent des grands serveurs placés dans des dépôts à distance et accessibles via le réseau Internet. Ces centres des données sont constitués de nombreux serveurs et consomment une énorme quantité d'énergie, ce qui exige aux fournisseurs Cloud d'avoir des mécanismes pour optimiser le placement des machines virtuelles dans ces serveurs. La virtualisation est une technique qui permet d'installer plusieurs systèmes d'exploitation dans un seul serveur. Cette opération réduit le nombre de serveurs en marche, et permet donc de minimiser la consommation d'énergie. L'optimisation de l'énergie consommée est très importante vu son impact significatif sur les dépenses et donc sur les revenues des fournisseurs d'une part, et sur les performances d'utilisation des ressources pour les clients d'autre part. Sachant que la minimisation de l'énergie consommés dans les centres de données Cloud est un problème d'optimisation de complexité NP-Complet, plusieurs techniques ont été adopté dans la littérature. Dans ce travail, nous proposons une nouvelle technique basée sur un nouvel algorithme bio-inspiré appelé "Optimisation de l'alimentation des Raies Manta" (MRFO : Manta Ray Foraging Optimization). Cet algorithme a été proposé récemment afin de résoudre les problèmes d'optimisation pour les applications d'ingénierie. Cependant, à notre connaissance, il n'est jamais utilisé pour résoudre le problème de placement des machines virtuelles dans le Cloud. A cet effet, le présent travail utilisera l'algorithme MRFO pour la première fois afin d'optimiser le placement des Virtuelles Machines (VMs) dans les centres de données Cloud en minimisant particulièrement l'énergie consommée. La stratégie proposée a été implémenté sur le simulateur CloudSim plus, un simulateur Cloud le plus reconnus. Les résultats de simulation montrent l'efficacité de l'algorithme proposé par rapport à d'autres approches notamment Random et Round Robin.

**Mots clé :** Cloud Computing, placement de machines virtuelles, méthode d'optimisation, Optimisation de l'alimentation des Raies Manta.

# Abstract

Cloud Computing is a technology that provides users with access to processing and storage capacities from their computers. These capacities constitute large servers placed in remote depots and accessible via the Internet network. These data centers consist of many servers and consume a huge amount of energy, which requires cloud providers to have mechanisms to optimize the placement of virtual machines in these servers. Virtualization is a technique that allows multiple operating systems to be installed in a single server. This operation reduces the number of servers running, and therefore minimizes energy consumption. The optimization of the energy consumed is very important given its significant impact on the expenses and therefore on the revenues of the suppliers on the one hand, and on the performance of the use of resources for the customers on the other hand. Knowing that minimizing the energy consumed in Cloud data centers is an NP-Complete complexity optimization problem, several techniques have been adopted in the literature. In this work, we propose a new technique based on a new bio-inspired algorithm called "Manta Ray Foraging Optimization" (MRFO: Manta Ray Foraging Optimization). This algorithm was proposed recently in order to solve optimization problems for engineering applications. However, to our knowledge, it is never used to solve the problem of placing virtual machines in the Cloud. To this end, the present work will use the MRFO algorithm for the first time in order to optimize the placement of virtual machines (VMs) in Cloud data centers by particularly minimizing the energy consumed. The proposed strategy has been implemented on the CloudSim plus simulator, a most recognized Cloud simulator. The simulation results show the efficiency of the proposed algorithm compared to other approaches including Random and Round Robin.

**Keywords:** Cloud Computing, placement of virtual machines, optimization method, Manta Ray Foraging optimization.

# ملخص

الحوسبة السحابية هي تقنية توفر للمستخدمين إمكانية الوصول إلى سعات المعالجة والتخزين من أجهزة الكمبيوتر الخاصة بهم. تشكل هذه القدرات خوادم كبيرة موضوعة في مستودعات بعيدة ويمكن الوصول إليها عبر شبكة الإنترنت. تتكون مراكز البيانات هذه من العديد من الخوادم وتستهلك قدرًا هائلاً من الطاقة ، مما يتطلب من موفري السحابة امتلاك آليات لتحسين وضع الأجهزة الافتراضية في هذه الخوادم. المحاكاة الافتراضية هي تقنية تسمح بتثبيت أنظمة تشغيل متعددة في خادم واحد. تقلل هذه العملية من عدد الخوادم قيد التشغيل ، وبالتالي تقلل من استهلاك الطاقة. يعد تحسين الطاقة المستهلكة أمرًا مهمًا للغاية نظرًا لتأثيره الكبير على النفقات وبالتالي على إيرادات الموردين من ناحية ، وعلى أداء استخدام الموارد للعملاء من ناحية أخرى. مع العلم أن تقليل الطاقة المستهلكة في مراكز البيانات السحابية يمثل مشكلة تحسين التعقيد NP-Complete ، فقد تم اعتماد العديد من التقنيات في الأدبيات. في هذا العمل ، نقترح تقنية جديدة تعتمد على خوارزمية جديدة مستوحاة من الأحياء تسمى " مانتا راي العلف الأمثل ". تم اقتراح هذه الخوارزمية مؤخرًا لحل مشكلات التحسين للتطبيقات الهندسية. ومع ذلك ، على حد علمنا ، لا يتم استخدامه مطلقًا لحل مشكلة وضع الأجهزة الافتراضية في السحابة. ولهذه الغاية ، سيستخدم العمل الحالي خوارزمية مانتا راي العلف الأمثل لأول مرة من أجل تحسين وضع الأجهزة الظاهرية في مراكز البيانات السحابية عن طريق تقليل الطاقة المستهلكة بشكل خاص. تم تنفيذ الاستراتيجية المقترحة على CloudSim plus simulator ، وهو أكثر محاكي السحابة شهرة. أظهرت نتائج المحاكاة كفاءة الخوارزمية المقترحة مقارنة مع الخصائص الوصفية الأخرى بما في ذلك عشوائي ومستدير روبن.

**الكلمات الرئيسية:** الحوسبة السحابية ، وضع الأجهزة الافتراضية ، طريقة التحسين ، مانتا راي العلف الأمثل.

# Table des matières

## Chapitre 4 : Implémentation

# Liste des Figures

# Introduction Générale

Dans le monde de l'informatique, le concept de Cloud Computing s'est répandu ces dernières années et s'est imposé comme un modèle primordial d'utilisation des ressources informatiques. Ces ressources fournies par le fournisseur Cloud sont accessibles sur un réseau informatique sous forme de services.

Avec le développement du Cloud Computing, la portée des centres de données continue de s'étendre et le centre de données Cloud utilise généralement des milliers de machines physiques. La technologie de virtualisation permet à une seule machine physique de prendre en charge plusieurs machines virtuelles, et chaque machine virtuelle exécute différents types de services et d'applications qui représentent les demandes de ressources provenant des utilisateurs.

Avec la diffusion du Cloud Computing, la consommation d'énergie des centres de données Cloud augmente de plus en plus. Selon les estimations d'Amazon, les coûts énergétiques des centres de données Cloud représente 42 % des coûts d'exploitation totaux. De plus, l'augmentation de la consommation d'énergie a entraîné une augmentation significative des émissions de dioxyde de carbone et a directement affecté notre environnement. Par conséquent, la réduction de la consommation d'énergie est une préoccupation impérieuse qui doit être résolue car elle réduira non seulement les coûts énergétiques, mais également la durabilité environnementale.

Une des solutions à ce problème est le placement optimal des machines virtuelles dans ces serveurs. Cette optimisation est très importante pour la réduction du nombre de serveurs actifs afin de réaliser une efficacité énergétique et d'augmenter les performances d'utilisation des ressources.

Dans ce travaille, nous utiliseront un nouvel algorithme bio-inspiré appelé "Optimisation de l'alimentation des Raies Manta" (MRFO : Manta Ray Foraging Optimization). Cet algorithme a été proposé récemment afin de résoudre les problèmes d'optimisation pour les applications d'ingénierie. Ce travaille utilisera l'algorithme MRFO pour la première fois afin de d'optimiser le placement des VMs dans les centres de données Cloud en minimisant la consommation d'énergie.

Sachant que l'algorithme MRFO d'origine traite l'optimisation des positions des Raies Manta dans un espace réel, et vu la nature discrète du problème de placement des machines virtuelles dans le Cloud, une version discrète de cet algorithme est proposée et utilisé pour résoudre ce problème.

Le présent mémoire est structuré autour de quatre chapitres et une conclusion générale comme suit :

**Dans le chapitre 1** nous définissons le concept de Cloud Computing et en présentant particulièrement ces caractéristiques, ses modèles de service et ses modèles de déploiement.

**Le chapitre 2** est consacré aux méthodes d'optimisation combinatoire et plus particulièrement aux méta-heuristiques, étant donné que le problème de placement de machines virtuelles dans le Cloud est un problème d'optimisation de complexité NP-Complet, ce qui rend ce type de méthodes très approprié pour ce type de problèmes.

**Dans le chapitre 3** nous présentons notre contribution qui consiste à la proposition d'une nouvelle stratégie basée sur l'algorithme MRFO afin de résoudre le problème de placement des machines virtuelles dans le Cloud.

**Le chapitre 4** permet au lecteur de voir le procédé de réalisation d'une implémentation de la stratégie proposée, à savoir l'environnement de développement, les outils utilisés et quelques interfaces utilisateur ainsi que les résultats de simulation.

Nous terminons ce mémoire par une conclusion générale, où nous résumons ce que nous avons fait, et ce que nous traçons comme perspectives pour des travaux futurs.

# Chapitre 1 : Le Cloud Computing – Concepts et terminologies

## 1. Introduction

Actuellement, le Cloud Computing représente une révolution dans le monde informatique. En effet, le Cloud Computing a émergé dans les dernières années comme une solution universelle utilisée par différents types d'utilisateurs.

Le Cloud Computing est une nouvelle technologie qui permet aux entreprises d'externaliser une partie ou l'intégralité de leur infrastructure informatique ce qui leur permet d'avoir des puissances de calcul et de stockage supplémentaires pour le traitement de grosses quantités d'informations.

## 2. Définitions

### Définition 1

Le Cloud Computing est un modèle permettant un accès réseau omniprésent, pratique et à la demande à un pool partagé de ressources informatiques configurables (par exemple, réseaux, serveurs, stockage, applications et services) qui peuvent être rapidement provisionnées et libérées avec un minimum d'efforts de gestion ou d'interactions avec le fournisseur de services [1].

### Définition 2

Le Cloud Computing peut également être vu comme un réseau informatique distribué composé de serveurs virtualisés, dont les ressources peuvent être approvisionnées de manière dynamique et reposant sur un contrat de service entre un fournisseur et un client [2].

### Définition 3

Selon Gartner [3] : Le Cloud Computing regroupe « *l'ensemble des disciplines, technologies et modèles d'entreprise utilisés pour fournir des capacités informatiques (logiciels, plateformes, matériels) à la manière d'un service à la demande, évolutif et élastique* ».

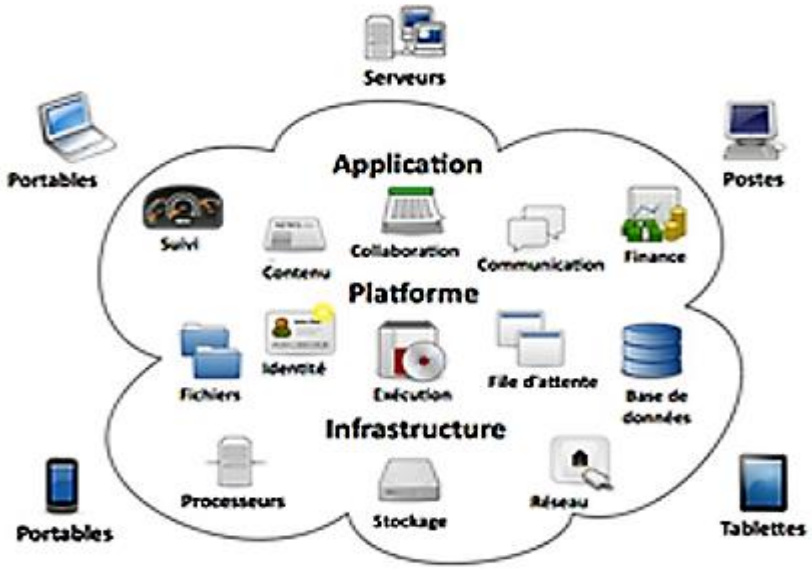



Figure 1.1. Cloud Computing.

## 3. Technologies connexes

Le Cloud Computing utilise des technologies telles que la virtualisation, l'architecture orientée services et les services web. Le Cloud Computing a connu son évolution en se basant sur plusieurs paradigmes informatiques, essentiellement : le Grid Computing, l'Utility Computing et l'Autonomic Computing, dont chacun partage certains aspects.

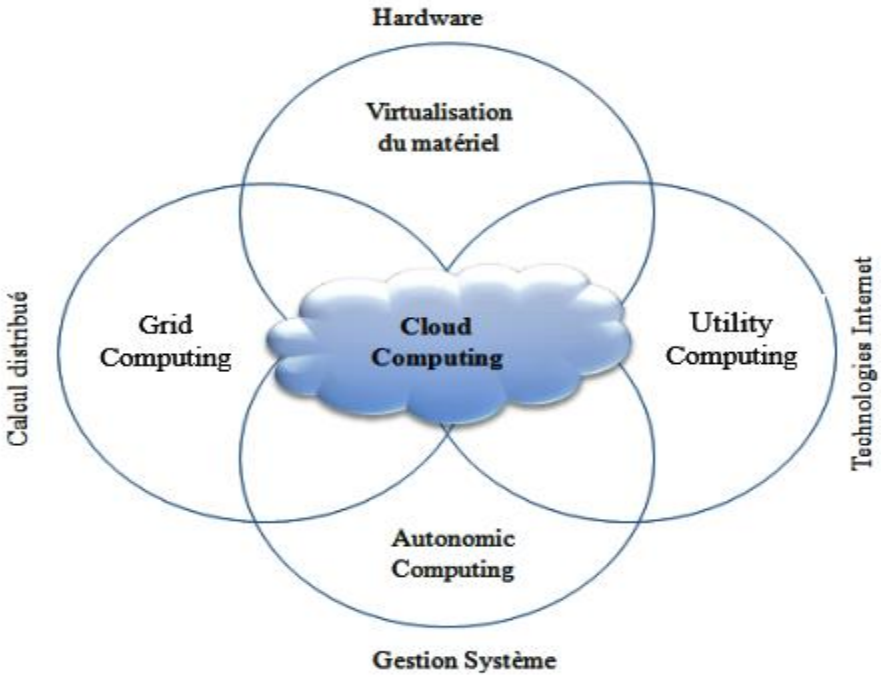



Figure 1.2. Technologies connexes [4].

### 3.1. La Grille Informatique « Grid Computing »

L'informatique en grille est un modèle informatique distribué qui coordonne des ressources indépendantes et géographiquement distribuées pour atteindre un objectif informatique commun. Le Grid s'appuie sur le partage dynamique de ressources entre participants, organisations et entreprises afin de pouvoir les agréger, et ainsi réaliser des applications scientifiques, qui sont généralement des calculs poussés requérant le traitement de très grandes quantités de données. Le Cloud Computing est similaire au Grid Computing : les deux adoptent le concept de fourniture de ressources en tant que services. Cependant, leurs objectifs sont différents : le Grid est principalement destiné à permettre à différents groupes de partager un accès en libre-service à leurs ressources, alors que le Cloud est destiné à fournir aux utilisateurs des services "à la demande" en se basant sur le principe du "*pay per use*". De plus, le Cloud exploite les technologies de virtualisation à plusieurs niveaux (plates-formes matérielles et applications) pour réaliser un partage et un provisionnement dynamiques des ressources [5].

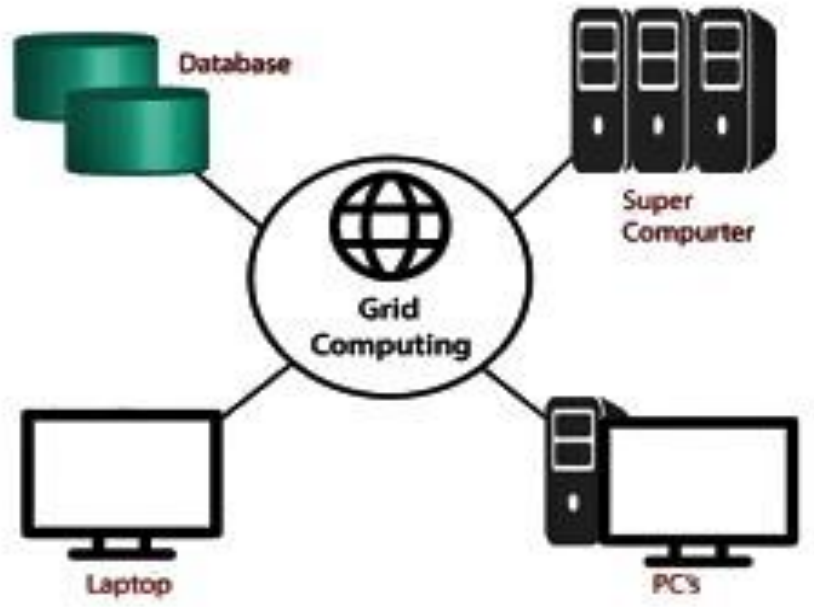



Figure 1.3. Grid Computing [6].

### 3.2. L'Informatique Utilitaire « Utility Computing »

L'utility Computing est simplement la délocalisation d'un système informatique ou de stockage. Il propose un modèle d'allocation des ressources à la demande et de facturation des utilisateurs en fonction de leur utilisation. Le Cloud Computing peut être considéré comme une réalisation de l'informatique utilitaire. Il adopte un système de tarification basé sur les services publics entièrement pour des raisons économiques. En fournissant des ressources à la demande

et payantes, les fournisseurs de services peuvent optimiser l'utilisation des ressources et réduire les coûts d'exploitation [5].

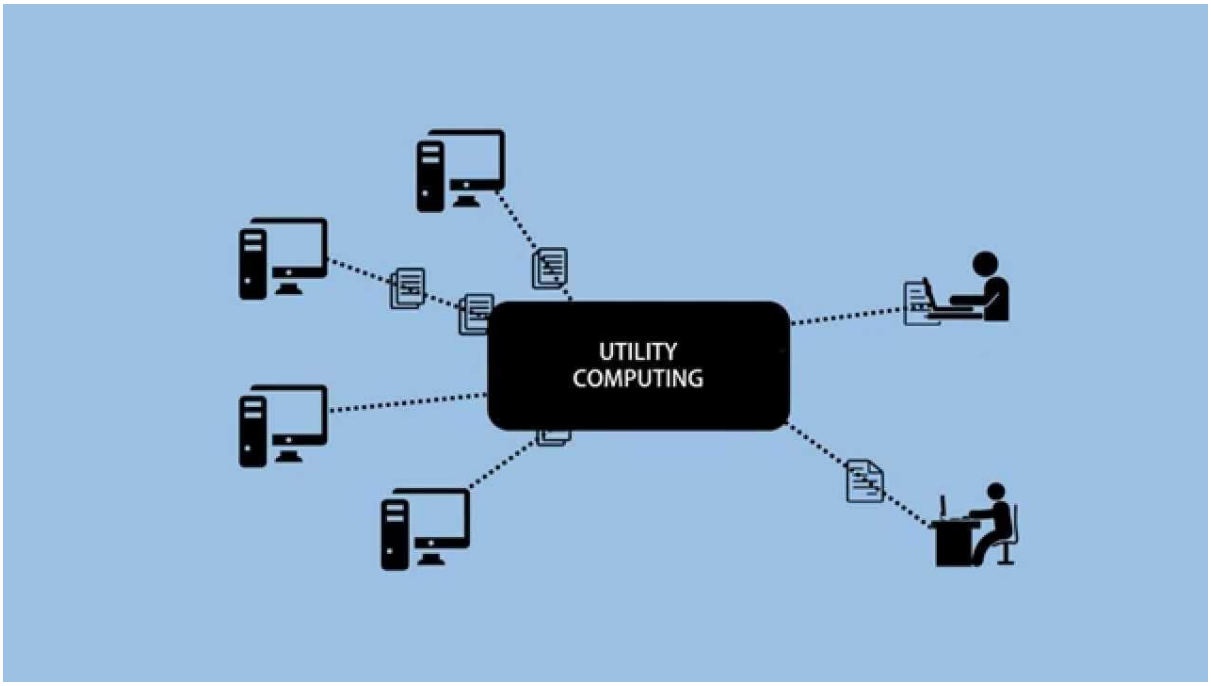



Figure 1.4. Utility Computing [7].

### 3.3. La Virtualisation

La virtualisation est une technologie qui permet d'abstraire les détails du matériel physique et de fournir des ressources virtuelles pour les applications de haut niveau. En effet, la virtualisation regroupe toutes les technologies matérielles ou logiciels qui permettent de faire fonctionner, sur une même machine physique, plusieurs configurations informatiques (systèmes d'exploitation, conteneurs, etc.), pour former de nombreuses machines virtuelles, qui reproduisent le comportement des machines physiques. La virtualisation est la base du Cloud Computing car elle offre la possibilité de mutualiser les ressources informatiques des fermes de serveurs, et ainsi d'allouer ou de réallouer dynamiquement des machines virtuelles à des applications à la demande. Un moniteur de machines virtuelles (VMM : *Virtual Machine Manager*), également appelé hyperviseur, divise la ressource physique du serveur physique principal en plusieurs machines virtuelles différentes, chacune exécutant un système d'exploitation et une suite d'utilisateurs distincts [5].

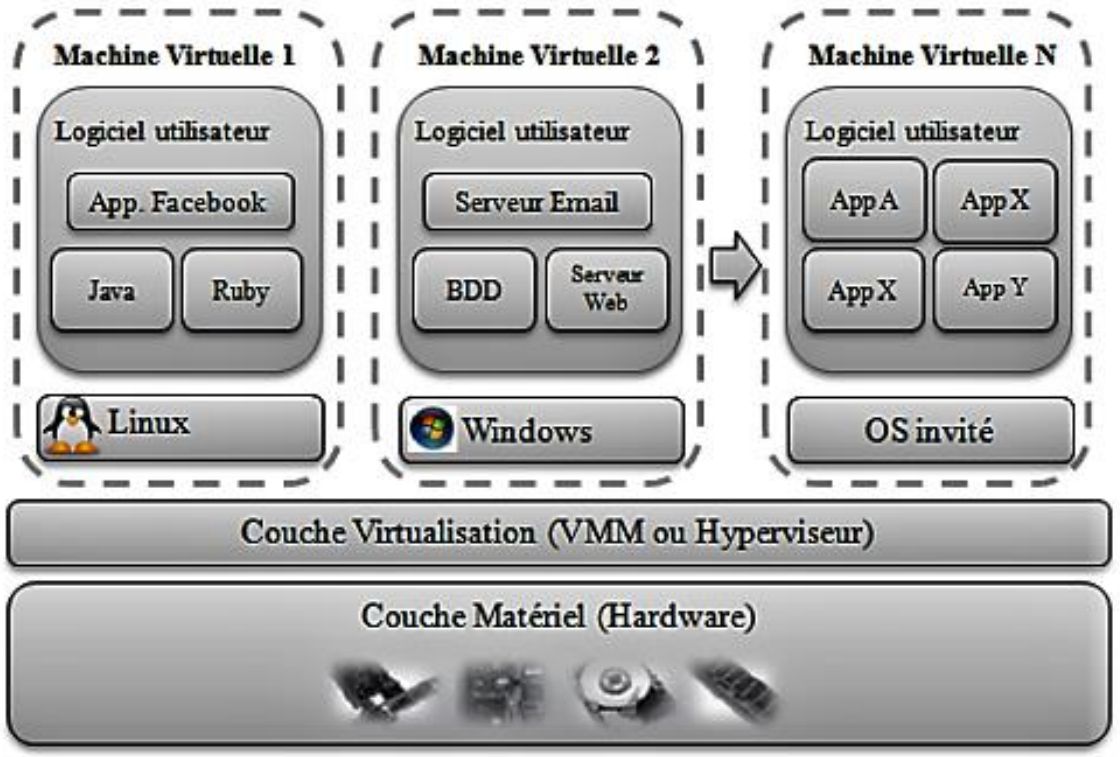



Figure 1.5. Virtualisation [8].

### 3.4. L'informatique autonome « Autonomic Computing »

L'informatique autonome vise à construire des systèmes informatiques capables de s'autogérer, c'est-à-dire capables de fonctionner selon des politiques et des règles générales définies sans intervention humaine. L'objectif de l'informatique autonome est de surmonter la complexité croissante de la gestion des systèmes informatiques, tout en étant capable de continuer à augmenter l'interconnectivité et l'intégration sans relâche. Bien que le Cloud Computing présente certaines similitudes avec l'informatique automatique dans la manière dont il interconnecte et intègre des centres de données distribués sur les continents, son objectif est en quelque sorte de réduire le coût des ressources plutôt que de réduire la complexité du système [5].

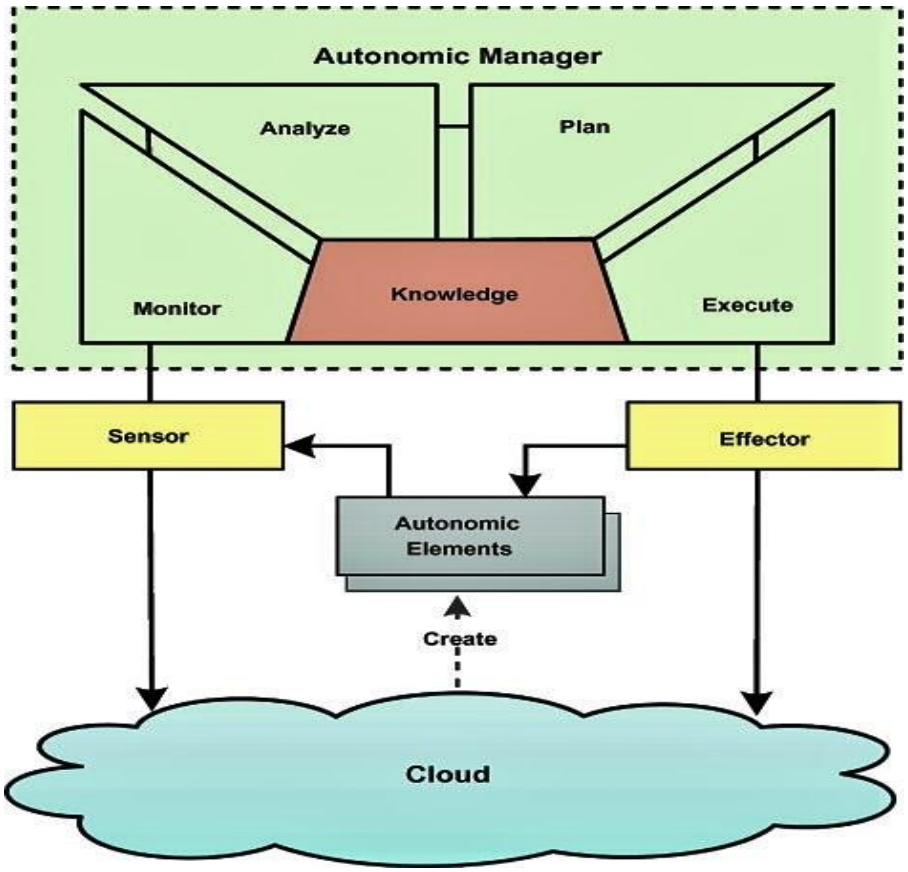



Figure 1.6. Autonomic Computing [9].

## 4. Caractéristiques du Cloud Computing

Le Cloud Computing est composé par cinq caractéristiques [1] :

- **Libre-service à la demande :** Un consommateur peut fournir unilatéralement des capacités informatiques, telles que le temps du serveur et le stockage réseau, selon les besoins, automatiquement, sans nécessiter d'interaction humaine avec chaque fournisseur de services.

- **Large accès au réseau :** les services sont accessibles en ligne et sur tout type de support (ordinateur de bureau, portable, smartphone, tablette). Tout se passe dans le navigateur internet.

- **Mise en commun des ressources :** Les ressources informatiques du fournisseur sont mises en commun pour servir plusieurs consommateurs à l'aide d'un modèle multi-tenant, avec différentes ressources physiques et virtuelles affectées et réaffectées dynamiquement en fonction de la demande des consommateurs. Il existe un sentiment d'indépendance géographique dans la mesure où le client n'a généralement aucun contrôle ou connaissance de l'emplacement exact des ressources fournies, mais peut

être en mesure de spécifier l'emplacement à un niveau d'abstraction plus élevé (par exemple, pays, état ou centre de données). Des exemples de ressources incluent le stockage, le traitement, la mémoire et la bande passante du réseau.

- **Élasticité rapide :** les capacités peuvent être provisionnées et libérées de manière élastique, dans certains cas automatiquement, pour évoluer rapidement vers l'extérieur et vers l'intérieur en fonction de la demande. Pour le consommateur, les capacités disponibles pour l'approvisionnement semblent souvent illimitées et peuvent être appropriées en n'importe quelle quantité à tout moment.
- **Service mesuré :** les systèmes Cloud surveillent et optimisent automatiquement l'utilisation des ressources en exploitant une capacité de mesure à un certain niveau d'abstraction approprié au type de service (par exemple, stockage, traitement, bande passante, comptes d'utilisateurs actifs). L'utilisation des ressources peut être surveillée, contrôlée, auditée et signalée, offrant une transparence à la fois au fournisseur et au consommateur du service utilisé.

## 5. Acteurs du Cloud

L'écosystème du Cloud Computing est composé principalement par cinq acteurs majeurs (Cloud Provider, Cloud Consumer, Cloud Carrier, Cloud Broker, Cloud Auditor) [10] :

### 5.1. Fournisseur de Cloud (Cloud Provider)

Le fournisseur des ressources Cloud Computing. Il est responsable de fournir un service Cloud Computing qui satisfait les caractéristiques définies dans la précédente section, tout en respectant les Service Level Agreements (SLAs) établies avec les autres acteurs (en particulier le Cloud Consumer). Le Cloud Provider a comme activité l'allocation, l'orchestration et la gestion des ressources qu'il offre tout en assurant le bon niveau de sécurité.

### 5.2. Consommateur Cloud (Cloud Consumer)

L'utilisateur des ressources Cloud Computing. Cet utilisateur peut être un utilisateur final ou un développeur selon le type du service Cloud alloué. Cet utilisateur peut être une personne, un groupe de personnes, les petites et moyennes entreprises, les multinationales ou les gouvernements.

### 5.3. Opérateur Cloud ou Fournisseur de réseau (Cloud Carrier or Network Provider)

Le fournisseur de réseau est l'intermédiaire qui assure principalement la connectivité entre les ressources Cloud Computing et la liaison entre les acteurs de l'écosystème

Cloud Computing (en particulier entre le Cloud Provider et le Cloud Consumer). Cet utilisateur peut jouer un simple rôle d'acheminements des paquets, comme il peut jouer un rôle plus important en offrant des fonctionnalités avancées dans le réseau. Ces fonctionnalités sont basées sur des SLAs établies avec les autres acteurs de l'écosystème.

### 5.4. Courtier en nuage (Cloud Broker)

Le courtier Cloud est un intermédiaire qui négocie la relation entre les Cloud Providers et les Cloud Consumers. Il peut offrir de nouveaux services qui simplifient les tâches de gestion du Cloud Consumer. Ce dernier peut demander les ressources Cloud Computing auprès du Cloud Broker au lieu du Cloud Provider directement.

### 5.5. Auditeur Cloud (Cloud Auditor)

L'auditeur Cloud s'occupe de la vérification et l'audition des services Cloud Computing. Il évalue les services offerts par les Cloud Providers, Cloud Carriers et Cloud Brokers du point de vue performances et sécuritaires. Le but principal est de vérifier que les fournisseurs respectent bien les SLAs qu'ils proposent.

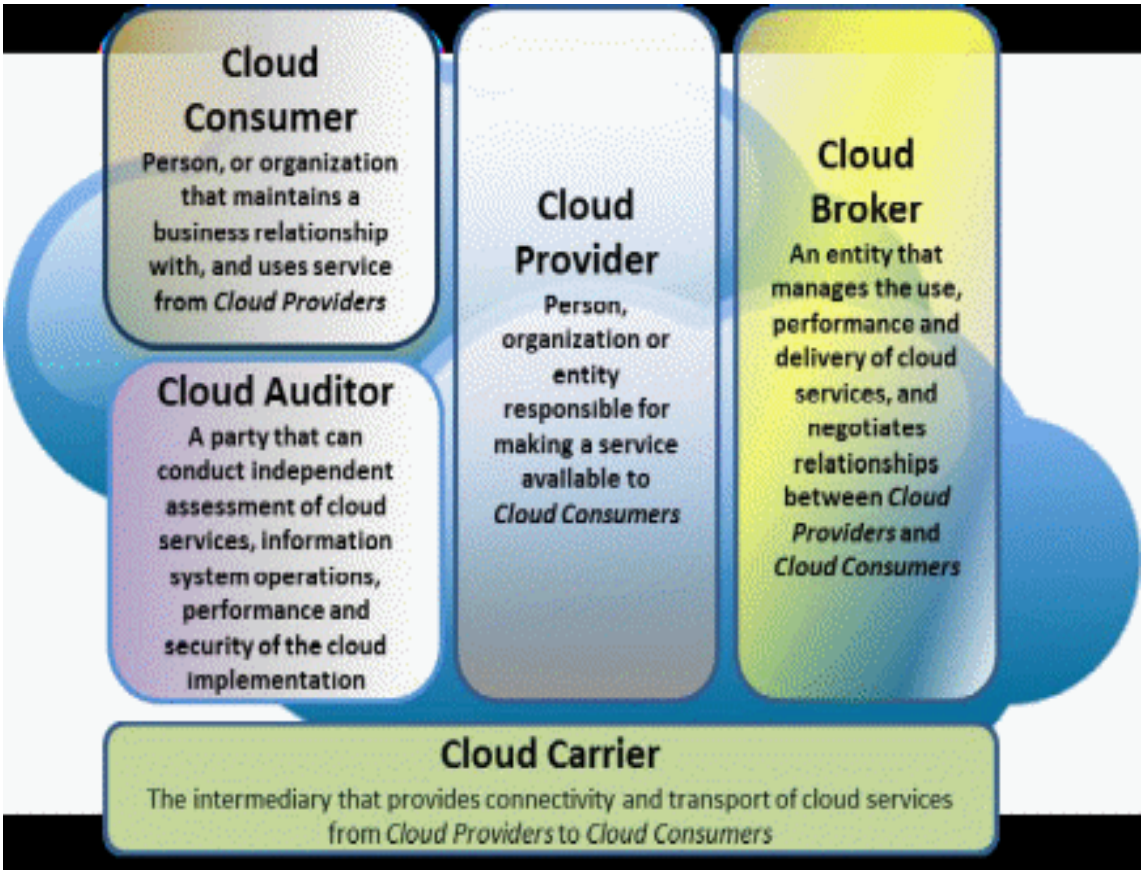



Figure 1.7. Acteurs du Cloud [11].

## 6. Modèles de services du Cloud Computing

### 6.1. Infrastructure as a Service (IaaS)

Les IP gèrent un large ensemble de ressources informatiques, telles que la capacité de stockage et de traitement. Grâce à la virtualisation, ils sont en mesure de diviser, d'affecter et de redimensionner dynamiquement ces ressources pour créer des systèmes ad hoc comme l'exigent les clients, les SP. Ils déploient les piles logicielles qui exécutent leurs services. Il s'agit du scénario Infrastructure en tant que service (IaaS) [12].

### 6.2. Platform as a Service (PaaS)

Les systèmes cloud peuvent offrir un niveau d'abstraction supplémentaire : au lieu de fournir une infrastructure virtualisée, ils peuvent fournir la plate-forme logicielle sur laquelle les systèmes fonctionnent. Le dimensionnement des ressources matérielles demandées par l'exécution des services se fait de manière transparente [12].

### 6.3. Software as a Service (SaaS)

Le fournisseur fournit des services d'intérêt potentiel à une grande variété de clients hébergés dans son infrastructure Cloud. Les services sont accessibles à partir de divers appareils clients via une interface client léger telle qu'un navigateur Web. Le client n'a pas besoin de gérer les ressources Cloud ni même les capacités des applications individuelles. Le client peut éventuellement se voir accorder des paramètres de configuration d'application limités et spécifiques à l'utilisateur. Une variété de services, fonctionnant comme un logiciel en tant que service, sont disponibles sur Internet [5].

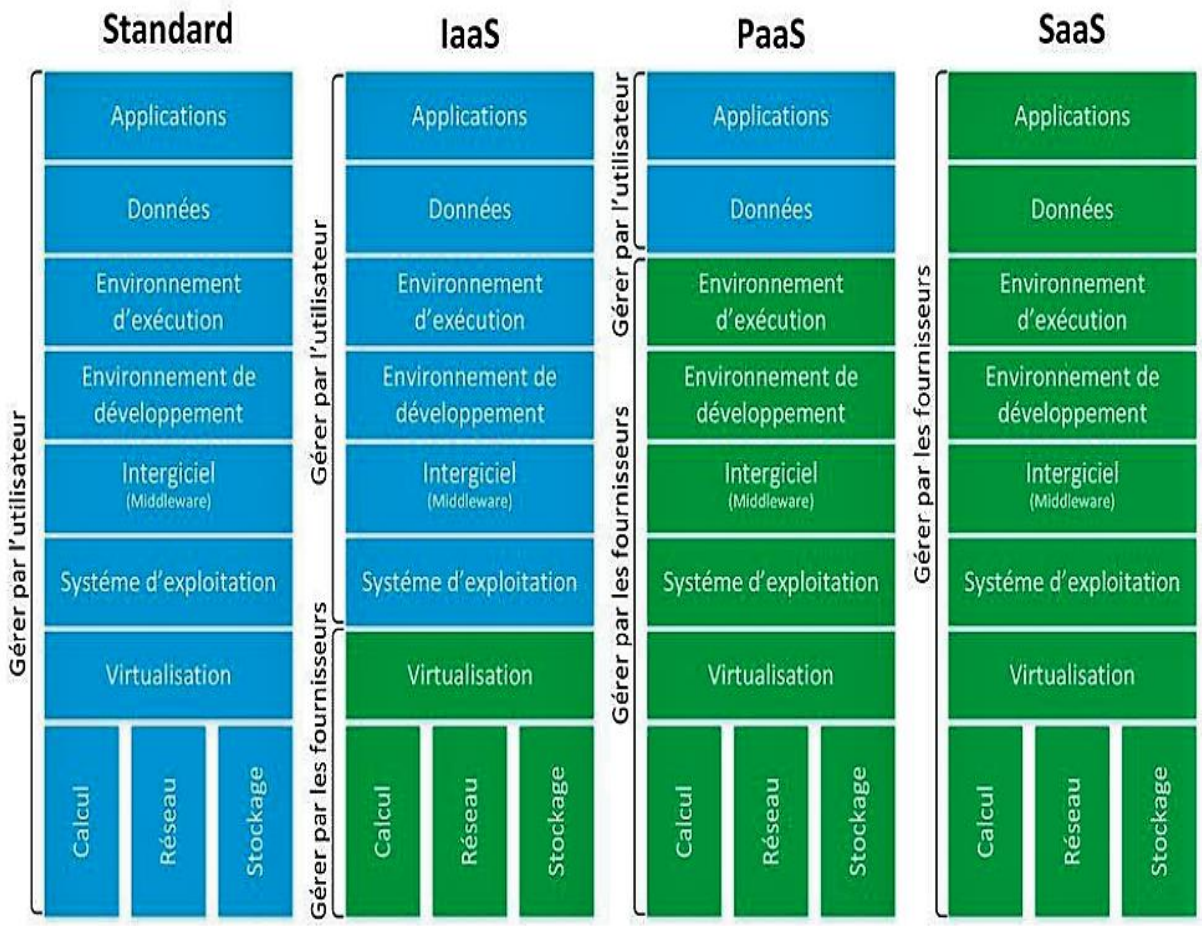



Figure 1.8. Modèles de services du Cloud Computing [13].

## 7. Modèles de déploiement

Le Cloud Computing est composé par quatre modèles de déploiement [1] :

### 7.1. Cloud public

L'infrastructure d'un Cloud public est provisionnée pour une utilisation ouverte par le grand public. Il peut être détenu, géré et exploité par une entreprise, une université ou une organisation gouvernementale, ou une combinaison de ceux-ci. Il existe dans les locaux du fournisseur de Cloud.

### 7.2. Cloud privé

Toutes les ressources de Cloud privé sont fournies exclusivement à une entreprise ou organisation. Le Cloud privé peut être géré par l'entreprise elle-même (Cloud privé interne) ou par un tiers (Cloud privé externe). Les ressources de Cloud privé sont généralement situées dans les locaux de l'entreprise ou chez un fournisseur de services. Dans ce dernier, l'infrastructure est entièrement dédiée à l'entreprise et accessible via un réseau sécurisé (type VPN).

### 7.3. Cloud Communautaire

L'infrastructure d'un Cloud communautaire est partagée par plusieurs organisations indépendantes ayant des intérêts communs. L'infrastructure peut être gérée par les organisations membres ou par un tiers. L'infrastructure peut être située, soit au sein des dites organisations, soit chez un fournisseur de services.

### 7.4. Cloud hybride

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

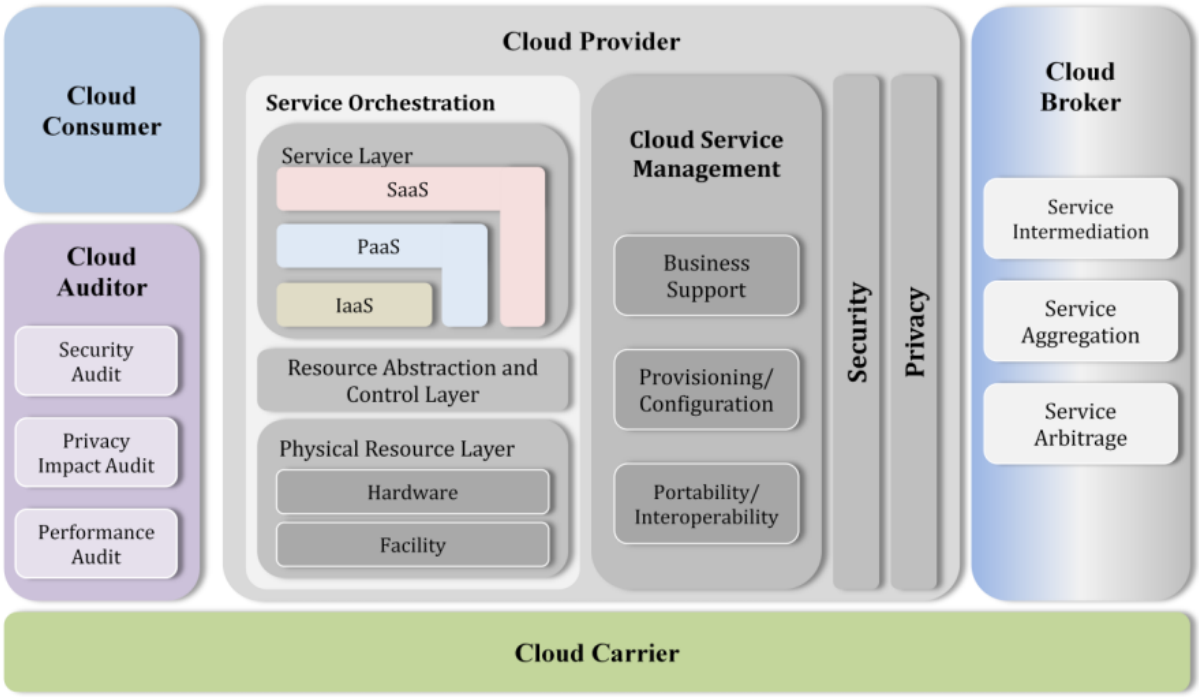



Figure 1.9. Architecture du Cloud Computing [1].

## 8. Avantages et inconvénients d'une solution Cloud

### Les avantages [10]

- Réduction des coûts d'infrastructure.
- Réduction des coûts de développement.
- Réduction des coûts des logiciels.

- Des ressources et services plus rapide à allouer et plus simple à utiliser.
- Accès aux ressources plus flexible.
- Meilleure utilisation, plus efficace, des ressources.
- Augmentation de la puissance de calcul.
- Grande capacité de stockage (quasi illimitée).
- Moins de problèmes d'entretien.
- Gestion des mises à jour plus simple et rapide.
- Pas de perte de données.
- Tout est considéré comme un service défini par un SLA.
- Infrastructure allouée et disponible juste à temps.
- Réduction du temps de mise sur le marché.

### Les inconvénients [10]

- Données Lock-in.
- Confidentialité des données.
- Chiffrement des données.
- Nécessité d'un accès réseau constant.
- Mauvais fonctionnement avec les connexions à basse vitesse.
- Faible niveau de la qualité de service dans le réseau.
- Risque d'engorgements lors des transferts de données.
- Problème d'interopérabilité.
- Problème de portabilité.
- Faible contrôlabilité.
- Manque de fonctionnalités d'audit.
- Des contrats de service SLAs non normalisés.

## 9. Conclusion

Au cours de ce chapitre nous avons présenté quelques généralités sur le Cloud Computing, à savoir sa définition, ses caractéristiques, ses acteurs, ses modèles de services qui l'offre, ses modèles de déploiement et enfin nous terminons avec ses avantages et inconvénients. Cela nous a conduits à déterminer notre besoin pour mettre en place une optimisation de placement des machines virtuelles dans un environnement Cloud.

# Chapitre 2 : Optimisation du problème de placement des machines virtuelles dans Cloud Computing

# 1. Introduction

Un des problèmes qui permet la rationalisation des ressources dans le Cloud est le placement des machines virtuelles (*VM : Virtual Machine*). Ceci consiste à allouer les serveurs pour héberger les machines virtuelles qui sont créés au sein du Cloud. Un placement optimal permet d'optimiser l'utilisation de ressources matérielles et de minimiser leur gaspillage le plus possible. Ces ressources constituent généralement les capacités en mémoire, la puissance de calcul et les bandes passantes de communication. Ce chapitre est consacré à une présentation du problème de placement des machines virtuelles dans les centres de données Cloud en présentant d'abord les problèmes d'optimisation d'une manière générale.

# 2. Problèmes d'optimisation

Un problème d'optimisation, noté $(X, f)$, est caractérisé par un ensemble réalisable ou admissible $X$ non-vide et une fonction objective $f$ qui associe un scalaire dans $R$ à chaque élément x de l'ensemble X. Les éléments de X sont dits solutions réalisables. Résoudre le problème $(X, f)$ revient à trouver parmi les solutions réalisables, une qui minimise ou maximise$f$, par exemple dans le cas d'un problème de minimisation, trouver une solution $x*$ $\in X$ telle que $(x) \geq f(x*)$ pour tout élément $x$ dans $X$. Une telle solution est dite optimale et sera notée $(X, f)$.

On peut dire aussi qu'un problème d'optimisation se définit comme la recherche du minimum ou du maximum (l'optimum) d'une fonction donnée. On peut aussi trouver des problèmes d'optimisation pour lesquels les variables de la fonction à optimiser sont des contraintes à évoluer dans une certaine partie de l'espace de recherche [14].

## 2.1. La complexité d'un problème d'optimisation

La complexité d'un problème d'optimisation est la complexité du meilleur algorithme qui permet de le résoudre. Si cet algorithme est polynomial, le problème est dit facile, autrement le problème est difficile. Nous présentons ici la complexité en temps (complexité temporelle). Il existe aussi une mesure de la performance des algorithmes en termes d'espace mémoire dite complexité en espace (complexité spatiale). Les classes de complexité d'un tel problème d'optimisation sont :

- **Un problème de la classe P :** un problème est dit polynomial s'il existe un algorithme de complexité polynomiale permettant de répondre à la question posée dans ce problème, quel que soit la donnée de celui-ci. La classe P est l'ensemble de tous les problèmes de reconnaissances polynomiaux.
- **Un problème de la classe NP :** un problème de reconnaissance est dans la classe NP si, pour toute instance de ce problème, on peut vérifier, en un temps polynomial par rapport à la taille de l'instance, qu'une solution proposée ou devinée permet d'affirmer que la présence est « oui » pour cette instance.
- **Un problème de la classe NP-hard :** on dit qu'un problème est dans la classe NP-hard si chaque autre problème dans NP est réductible d'une manière polynomiale dans ce dernier.

## 2.2. La classification des problèmes d'optimisation

On peut classer les différents problèmes d'optimisation que l'on rencontre dans la vie courante en fonction de leurs caractéristiques [14] :

1. **Nombre de variables de décision**
   - Une ⇒monovariable.
   - Plusieurs ⇒multivariable.
2. **Type de la variable de décision**
   - Nombre réel continu ⇒ continu.
   - Nombre entier ⇒ entier ou discret.
   - Permutation sur un ensemble fini de nombres ⇒ combinatoire.
3. **Type de la fonction objectif**
   - Fonction linéaire des variables de décision ⇒ linéaire.
   - Fonction quadratique des variables de décision ⇒ quadratique.
   - Fonction non linéaire des variables de décision ⇒ non linéaire.
4. **Formulation du problème**
   - Avec des contraintes ⇒ contraint.
   - Sans contraintes ⇒ non contraint.

### 2.3. Classification des méthodes d'optimisation

Les méthodes d'optimisation peuvent être classées, selon la méthode de recherche de l'optimum, en deux grandes catégories : les méthodes exactes et les méthodes stochastique.

#### 2.3.1 Méthodes exactes

Les méthodes exactes sont garanties pour trouver une solution optimale et prouver son optimalité pour chaque instance d'un problème d'optimisation. Cependant, le temps d'exécution augmente souvent considérablement avec la taille de l'instance, et souvent seules les instances de taille petite ou modérée peuvent être pratiquement résolu à l'optimalité démontrable. On peut citer entre autres :

#### 2.3.2. Méthodes stochastiques

Les méthodes stochastiques ne sont pas garanties pour trouver une solution optimale. Elles se basent sur des mécanismes de transitions aléatoires et probabilistes.

Dans les sections ultérieures, nous décrivons en détail ces deux classes de méthodes en présentant ainsi des exemples de méthodes les plus utilisées pour chacune d'eux.

## 3. Méthodes d'optimisation du problème de placement des machines virtuelles dans le Cloud

### 3.1. Placement statique vs placement dynamique

Les méthodes de placement de machines virtuelles peuvent être classées en deux catégories (figure 2.1) :

#### 3.1.1. Placement statique de MV

Le placement statique des machines virtuelles est effectué lorsque le système est en mode hors ligne ou lors du démarrage du système. Il s'agit du placement initial des VM dans l'environnement Cloud. Aucune correspondance préalable des machines virtuelles n'est trouvée. Un algorithme de placement statique peut être classé en deux types :

a. **Approche basée sur la puissance :** vise à obtenir une cartographie VM-PM qui se traduit par un système économe en énergie avec une utilisation maximale des ressources.

b. **Approche basée sur la QoS :** Vise à obtenir une cartographie VM-PM pour assurer une satisfaction maximale des exigences de qualité de service.

### 3.1.2 Placement dynamique de MV

Permet d'obtenir des solutions optimales à partir du mappage déjà présent des machines virtuelles à un coût minimal. Les algorithmes de placement dynamique de VM peuventt être classés en 2 types :

- **Proactif :** modifie la machine physique de la machine virtuelle d'un placement initial avant que le système n'atteigne une certaine condition ou ne satisfasse à un critère spécifique.
- **Réactif :** passe à un placement initial après que le système a atteint un certain état indésirable. La modification de cet emplacement d'origine peut être due à des problèmes de performances, de maintenance, d'alimentation ou de charge ou à des violations des SLAs.

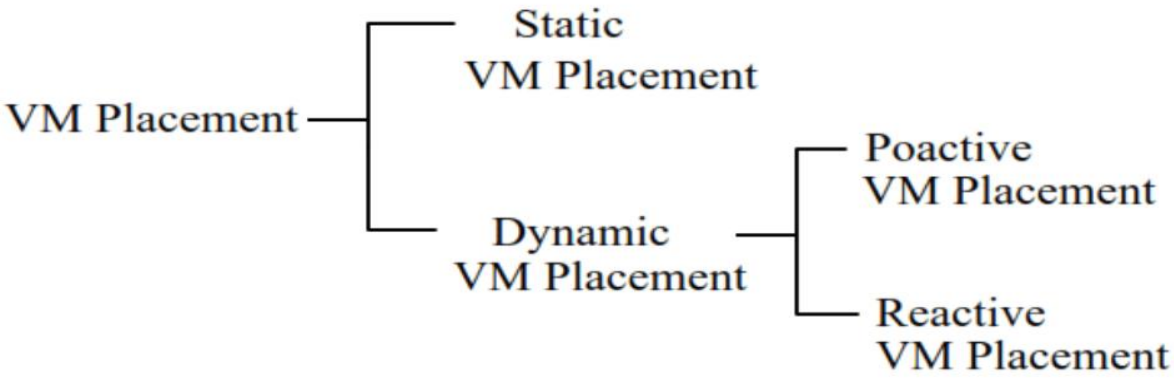



Figure 2.1. Classifications de placement des virtuelles machines.

## 3.2. Méthodes d'optimisation des problèmes de placement des machines virtuelles

### 3.2.1. Méthodes exactes de résolution des modèles mathématiques

Plusieurs techniques exactes existent pour obtenir des solutions optimales aux problèmes NP-difficiles, mais elles sont plus efficaces pour les petits ensembles de problèmes. Cependant, ces techniques exactes sont inefficaces pour les grands ensembles de problèmes car leur temps de calcul varie exponentiellement avec leur taille. Parmi les méthodes exactes, on trouve des méthodes d'énumération de toutes les solutions possibles ; nous avons également identifié des techniques de division intelligentes de l'espace des solutions possibles afin de trouver la meilleure solution. Ces solutions répondent aux objectifs à atteindre et aux contraintes à respecter. Un ensemble de contraintes à respecter peut restreindre un ensemble de solutions possibles pour trouver une solution qui maximise ou minimise une fonction objective qui respecte les contraintes. Dans cette classe des méthodes, on peut citer les algorithmes classiques suivants : la programmation linéaire, la programmation dynamique et les méthodes de recherche arborescente (Branch & Bound) [16].

- **La programmation linéaire :** C'est une méthode d'optimisation permettant de résoudre de nombreux problèmes économiques et industriels et se posent lorsque l'on cherche à rendre optimale (minimum ou maximum) une fonction linéaire de plusieurs variables. Ces variables étant assujetties à des contraintes linéaires, c'est à dire, du premier degré [17].
- **La programmation dynamique :** Basée sur une décomposition du problème en sous-problèmes plus simples. A chaque sous-problème correspond un ensemble d'options, représentant chacune un coût en termes de fonction objectif. Les algorithmes basés sur la programmation dynamique sont généralement faciles à implémenter et très efficaces pour résoudre les problèmes de petites et moyennes tailles [17].
- **Les méthodes de recherche arborescente (Branch & Bound) :** C'est l'une des méthodes qui propose un mécanisme de recherche très intelligent, grâce auquel elle permet une bonne exploitation de l'espace de recherche et l'aboutissement à la solution optimale plus rapidement que d'autres méthodes exactes en combinant deux principes primordiaux : la séparation et l'évaluation [18].

### 3.2.2. Méthodes stochastiques

#### 3.2.2.1. Heuristique

Une heuristique est un algorithme approché qui fournit rapidement une solution réalisable, sans garanti d'optimalité, pour un problème d'optimisation spécifique. C'est une règle d'estimation, une stratégie, une méthode ou astuce utilisée pour améliorer l'efficacité d'un système qui tente de découvrir les solutions des problèmes complexes [19].

#### 3.2.2.2. Métaheuristique

Les métaheuristiques sont un ensemble d'algorithmes d'optimisation visant à résoudre les problèmes d'optimisation difficiles. Elles sont souvent inspirées par des systèmes naturels, qu'ils soient pris en physique (cas du recuit simulé), en biologie de l'évolution (cas des algorithmes génétiques) ou encore en éthologie (cas des algorithmes de colonies de fourmis ou de l'optimisation par essaims particulaires) [20].

Les métaheuristiques peuvent être classées en deux catégories :

##### A) Les métaheuristiques à solution unique

Les métaheuristiques à base de solution unique, aussi appelées méthodes de trajectoire. Contrairement aux métaheuristiques à base de population, ces méthodes débutent la recherche avec une seule solution initiale Elles se basent sur la notion du voisinage pour améliorer la qualité de la solution courante. En fait, la solution initiale subit une série de modifications en fonction de son voisinage. De nombreuses méthodes ont été développées dans cette catégorie, parmi lesquelles nous citons les plus connues : la méthode de descente, la méthode du recuit simulé et la recherche taboue [21].

###### a) Recuit Simulé

Cette méthode de recherche a été proposée par des chercheurs d'IBM qui étudiaient les verres de spin. Ici, on utilise un processus métallurgique (le recuit) pour trouver un minimum. En effet, pour qu'un métal retrouve une structure proche du cristal parfait (l'état cristallin correspond au minimum d'énergie de la structure atomique du métal), on porte celui-ci à une température élevée, puis on le laisse refroidir lentement de manière à ce que les atomes aient le temps de s'ordonner régulièrement [22].

Ce processus métallurgique a été transposé à l'optimisation et a donné une méthode simple et efficace. Le pseudo code du recuit simulé est représenté dans l'algorithme 1.

Le fonctionnement de cet algorithme est le suivant :

- On commence par choisir un point de départ au hasard :
- On calcule un voisin de ce point (γ = Voisin (x)).
- On évalue ce point voisin et on calcule l'écart par rapport au point d'origine (ΔC = C(γ ) - C(x)).
- Si cet écart est négatif, on prend le point γ comme nouveau point de départ, s'il est positif on peut quand même accepter le point γ comme nouveau point de départ, mais avec une probabilité $e^{-\frac{\Delta c}{T}}$ (qui varie en sens inverse de la température T).
- Au fur et à mesure du déroulement de l'algorithme, on diminue la température T (T = α (T)), souvent par paliers.
- On répète toutes ces étapes tant que le système n'est figé (par exemple, tant que la température n'a pas atteint un seuil minimal).

Une recherche de recuit simulé acceptera n'importe quelles nouvelles solutions qui sont évaluées comme des solutions supérieures, mais elle acceptera aussi les changements négatifs de qualité avec une probabilité qui dépend de la taille de diminution dans la qualité et la valeur courante de la température. La température commence à une valeur élevée - idéalement suffisamment élevée pour que toute solution inférieure ait presque 100 % de chances d'être acceptée, et diminue au fur et à mesure que le processus suit son cours. La probabilité qu'une solution négative soit acceptée diminue, jusqu'à ce qu'il n'y ait aucune chance qu'un changement de qualité négatif soit autorisé [23].

Le pseudo-code de cette méthode est représenté dans l'algorithme 1.

**Algorithme 1** Recuit simulé.

Init T (température initial)

Init $x$ (point de départ)

Init $\Delta T$ (temperature)

While (not (end))

$y$ =V*oisin* ($x$)

$\Delta C = C\ (y)$-C ($x$)

if $\Delta C < 0$ then $y = x$

else if alea (0,1) < $e^{-\frac{\Delta C}{T}}$ then $y=x$

$T=\alpha\ (T)$

if $T < \Delta T$ then end (while)

Repeat (while)

Quelques applications du recuit simulé [24] :

- ✓ Traitement d'images.
- ✓ Problèmes d'ordonnancement.
- ✓ Conception des circuits électroniques (Problème de placement et de routage).
- ✓ Organisation du réseau informatique du Loto (France).
- ✓ Collecte des ordures ménagères.
- ✓ Problème du voyageur de commerce (10000 villes). La longueur de la tournée excède de moins de 2% celle de la tournée optimale.

Avantages de la méthode [24] :

- ✓ Solution de bonne qualité.
- ✓ Méthode générale et facile à programmer.
- ✓ Souplesse d'emploi : De nouvelles contraintes peuvent être facilement incorporées.

Inconvénients [24] :

- ✓ Nombre important de paramètres.
- ✓ Temps de calcul : excessif dans certaines applications.

### b) La Recherche Tabou

Cette méthode, mise au point par F. Glover [25] est conçue en vue de surmonter les optimums locaux de la fonction objective. C'est une technique d'optimisation combinatoire que certains présentent comme une alternative au recuit simulé.

A partir d'une configuration initiale quelconque, la méthode engendre une succession de configurations qui doivent aboutir à la configuration optimale. A chaque itération, le mécanisme de passage d'une configuration, soit $X_n$, à la suivante, soit $X_{n+1}$, est le suivant :

- On construit l'ensemble des « voisins » de $X_n$, c'est-à-dire l'ensemble des configurations accessibles en un seul « mouvement » élémentaire à partir de $X_n$ (si cet ensemble est trop vaste, on en extrait aléatoirement un sous-ensemble de taille fixée) : soit Voisinage ($X_n$) l'ensemble (ou le sous-ensemble) envisagé ;
- On évalue la fonction objective f du problème de chacune des configurations appartenant à Voisinage ($X_n$). La configuration $X_{n+1}$, qui succède la configuration $X_n$ dans la chaîne de Markov construite par Tabou, est la configuration de Voisinage ($X_n$) en laquelle f prend sa valeur minimale.

Notons que la configuration $X_{n+1}$ est adoptée même si $f(X_{n+1}) > f(X_n)$ : c'est grâce à cette particularité que Tabou permet d'éviter les optimums locaux de *f*.

Cependant, telle quelle la procédure ne fonctionne généralement pas, car il y a un risque important de retourner à une configuration déjà retenue lors d'une itération précédente la procédure ne fonctionne généralement pas, car il y a un risque important de retourner à une configuration déjà retenue lors d'une itération précédente, ce qui provoque l'apparition d'un cycle. Pour éviter ce phénomène, on tient à jour, à chaque itération, une « liste Tabou » de mouvements interdits ; cette liste -qui a donné son nom à la méthode- contient les mouvements inverses ($X_{n+1} \rightarrow X_n$) des m derniers mouvements ($X_n \rightarrow X_{n+1}$) effectués (typiquement m=7). La recherche du successeur de la configuration courante $X_n$est alors restreinte aux voisins de $X_n$qui peuvent être atteints sans utiliser un mouvement de la liste tabou. La procédure peut être stoppée dès que l'on effectué un nombre donné d'itérations, sans améliorer la meilleure solution atteinte jusqu'ici.

L'algorithme ainsi décrit et dit « Tabou simple ». Selon [26], il serait plus efficace que le recuit simulé pour le problème modèle du « coloration d'un graphe ». Cependant, le mode de

construction de la liste tabou – qui, pour une simple raison d'économie de place mémoire, contient des mouvements interdits, et non des configurations interdites – peut bloquer l'accès à certaines solutions, pourtant non encore visitées. Pour éviter cet inconvénient, on peut employer la méthode plus complexe dite « tabou généralisée », qui prévoit la possibilité d'annuler le statut tabou d'un mouvement, lorsque le bénéfice escompté est « suffisant » (cette circonstance est appréciée à l'aide de la notion de « niveau d'aspiration ».

Le pseudo-code de cette méthode est représenté dans l'algorithme 2.

**Algorithme** Recherche Tabou.

Init L=Ø (liste des points tabous)
Init $x$ (point de départ)
Init $k_{fin}$ (nb d'itérations max total)
Init $k$=0 et $l$=0
While (not (end))
  $y$ =V*oisinage* ($x$)–L (on prend un voisinage sans points tabous)
  $\Delta C = C\ (y)$-C ($x$)
  $x = y$ avec $y \in$ *Voisinage* $(x) - L$ tel que min $C(y) - C(x)$
  $L = L + \{x\}$
  $k = k + 1$
  Update L (on retire des mouvements tabous de L suivant des critères d'aspiration)
  if $k = k_{fin}$ then end (while)
Repeat

### c) La recherche par descente

La méthode de descente est une des métaheuristiques à base de solution unique, la plus simple et la plus utilisée. Elle est facile à mettre en place et donne de bons résultats [27]. La figure 2.2 schématise son principe général, qui est aussi détaillé par l'algorithme. Partant d'une solution initiale donnée, à chaque itération, l'algorithme remplace la solution courante par une solution voisine qui améliore la fonction objective. La descente s'arrête lorsqu'il n'y a plus de solution voisine permettant d'améliorer la solution courante, ainsi un optimum local est atteint.

Il existe plusieurs stratégies pour la sélection d'une meilleure solution voisine : choisir la première solution voisine améliorante (*first improvement*), choisir la meilleure parmi toutes les

solutions voisines (*best improvement*), ou encore en choisir une aléatoirement, parmi celles qui améliorent la solution courante (*random selection*).

Le pseudo-code de cette méthode est représenté dans l'algorithme 3.

**Algorithme** Méthode de la descente

Sol = $S_0$ // *Génération d'une solution initiale*
**Tant que** ∃ solution voisine (sol) **faire**
- Génération des solutions voisines candidates V(Sol) // *S' dans le voisinage*

**S'il** n'y a pas de meilleure solution voisine **alors**
Stop
**Fin si**
Sol= S' //*Sélection d'une meilleure solution voisine S' ∈ V(Sol)*
**Fin tant que**
**Retour** Solution finale trouvée (optimum local)

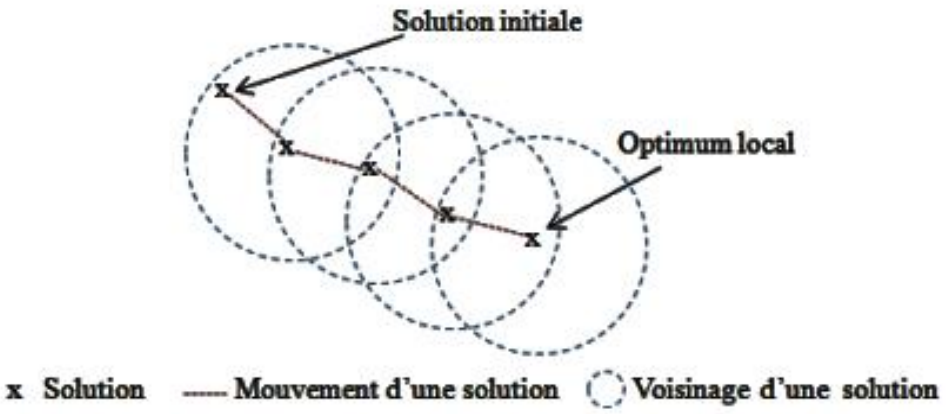



Figure 2.2. Processus d'une recherche par descente.

Une des principales limites de la recherche locale par descente est qu'elle converge vers un optimum local. Pour éviter de rester bloqué sur un optimum local, des mécanismes supplémentaires sont proposés, menant à la création de nouveaux algorithmes (recuit simulé, recherche tabou), ou à l'intégration de la recherche locale dans un processus itératif (ILS).

## B) Les métaheuristiques à base de population

Les métaheuristiques sont utilisées pour optimiser les problèmes complexes de grandes tailles. Contrairement aux méthodes exactes, les métaheuristiques à base de population de solutions améliorent une population de solutions, au fur et à mesure des itérations. De nombreuses méthodes ont été développées dans cette catégorie, parmi lesquelles nous citons les plus connues, à savoir : l'algorithme de colonies de fourmis, l'algorithme génétique, Optimisation par Essaim Particulaire [28].

### a) L'algorithme de colonies de fourmis

Les algorithmes de colonies de fourmis ont été proposés par Marco Dorigo dans sa thèse de doctorat en en 1992. Cet algorithme reproduit le comportement des fourmis dans l'exploration de l'espace à la recherche de sources de nourriture. Une fourmi seule n'est pas capable de trouver et de transporter suffisamment de nourriture pour nourrir la ruche. En explorant les chemins d'une source de nourriture, les fourmis libèrent un composant chimique appelé phéromone. De cette façon, les fourmis peuvent retrouver leur chemin vers la ruche et vers la source de nourriture. D'autres fourmis peuvent trouver cette piste de phéromones et pourraient la suivre. Il est plus probable qu'une fourmi suive un chemin si celui-ci contient une forte concentration de phéromones. Les fourmis suivant le chemin le plus court font un plus grand nombre d'allers-retours, donc plus de phéromones sont déposées dans le sentier. Après un certain temps, le chemin le plus court contient la plus grande quantité de phéromones. Par conséquent, la plupart des fourmis suivent ce chemin, illustré dans la figure 2.3 [28].

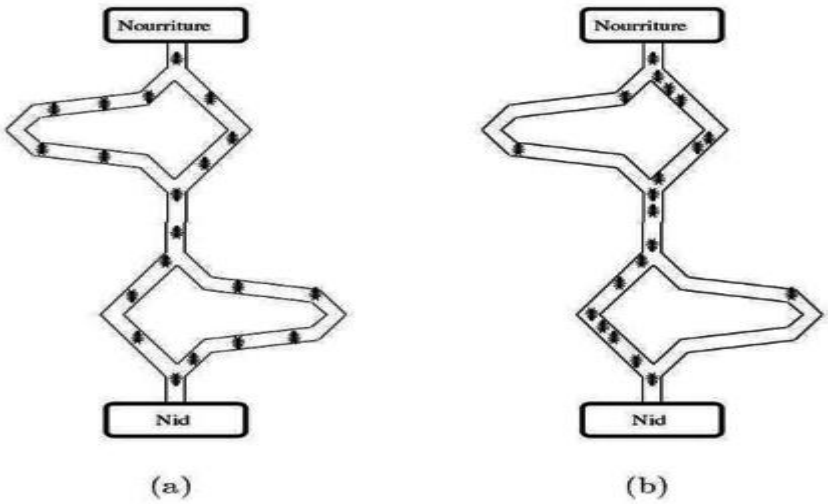



Figure 2.3. Trouver le chemin le plus court entre la nourriture et le nid.

### b) L'algorithme génétique

Les algorithmes génétiques sont des algorithmes évolutionnaires inspirés du domaine de la biologie. Les techniques utilisées reproduisent le schéma d'évolution des espèces et construisent leur vocabulaire. Les solutions sont appelées individus et l'algorithme traite plusieurs individus simultanément ; Un exemple d'individu utilisé dans notre simulation de problème est présenté ci-dessous (Figure 2.4) :

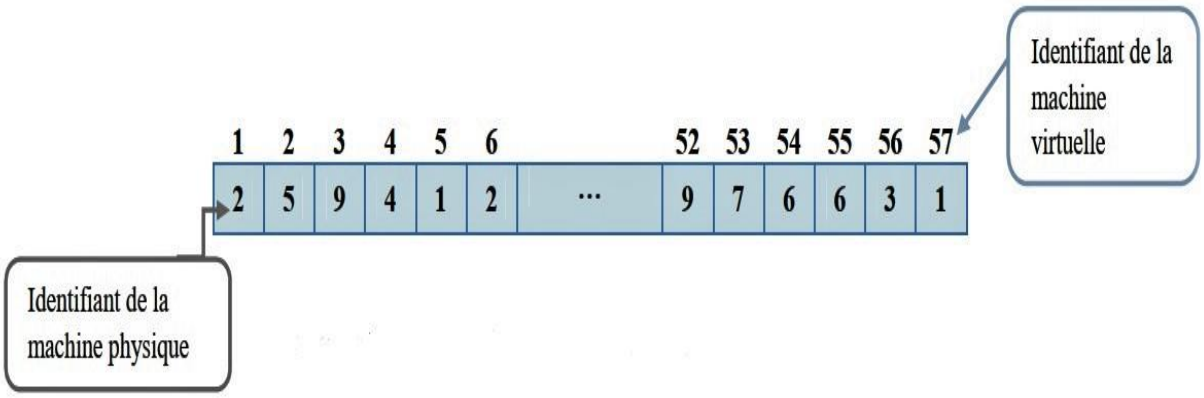



Figure 2.4. Exemple d'une représentation d'un individu.

L'ensemble de ces individus est appelé population et évolue à chaque itération de l'algorithme. La population relative à une itération donnée s'appelle génération, on obtient donc une nouvelle génération à chaque itération. Les individus qui servent à produire la nouvelle génération sont appelés parents et les individus résultants sont appelés enfants ou fils.

Le principe des algorithmes génétiques est simple, l'idée est de faire évoluer une population initiale grâce à des mécanismes de reproduction et de sélection de manière à obtenir des individus de plus en plus performants ; la figure 2.5 ci-dessous illustre cela.

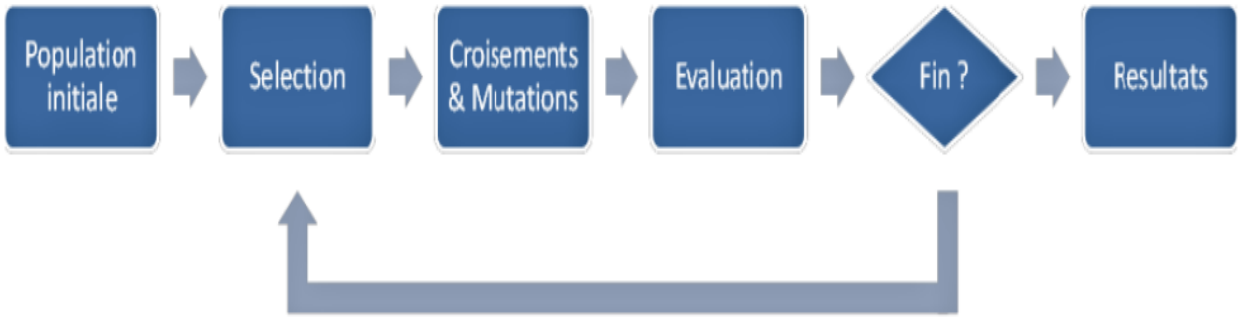



Figure 2.5. Fonctionnement des algorithmes génétiques.

- **Population initiale**

Initialement, on génère un nombre aléatoire d'individus afin de construire la population initiale.

- **Sélection**

La sélection consiste à choisir les individus les mieux adaptés afin d'avoir une population de solution la plus proche de converger vers l'optimum global.

Il existe plusieurs techniques de sélection. Voici les principales utilisées :

**Sélection par rang :** Cette technique de sélection choisit toujours les individus possédant les meilleurs scores d'adaptation.

**Probabilité de sélection proportionnelle à l'adaptation :** Technique de la roulette ou roue de la fortune, pour chaque individu, la probabilité d'être sélectionné est proportionnelle à son adaptation au problème.

**Sélection par tournoi :** Cette technique utilise la sélection proportionnelle sur des paires d'individus, puis choisit parmi ces paires l'individu qui a le meilleur score d'adaptation.

**Sélection uniforme :** La sélection se fait aléatoirement, uniformément et sans intervention de la valeur d'adaptation.

Voici un exemple des individus une fois la sélection effectuée dans la figure 2.6 ci-dessous :

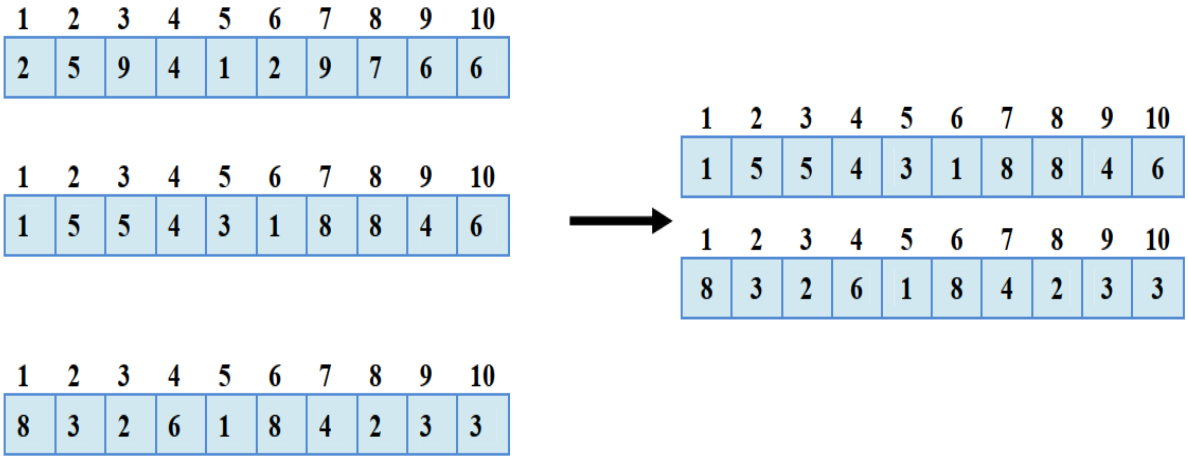



Figure 2.6. Exemple de sélection.

On réalise ensuite un croisement entre deux chromosomes parmi la population restante.

- **Croisement**

Les opérateurs de croisement produisent un ou deux enfants à partir de deux parents, c'est le résultat obtenu lorsque deux chromosomes partagent leurs particularités.

Il existe deux méthodes de croissement : simple ou double croisement.

Le simple croisement consiste à fusionner les particularités de deux individus à partir d'un pivot, afin d'obtenir un ou deux enfants dans la figure 2.7 ci-dessous :

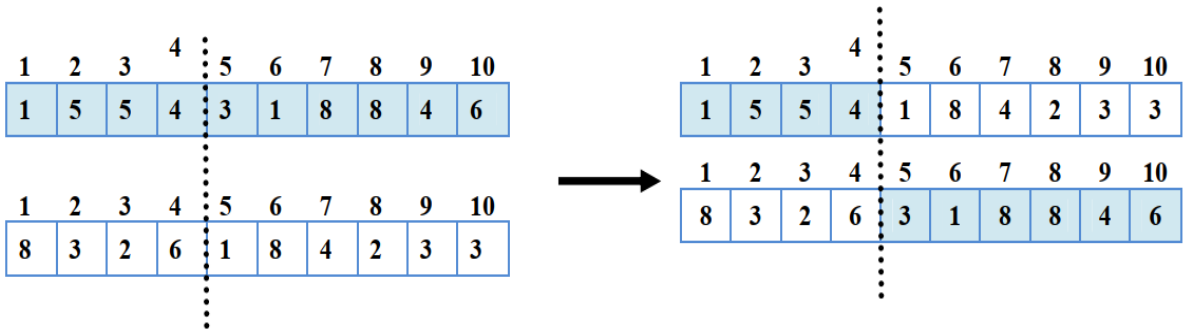



Figure 2.7. Exemple de croisement simple.

Le double croisement repose sur le même principe, sauf qu'il y a deux pivots dans la figure 2.8 ci-dessous :

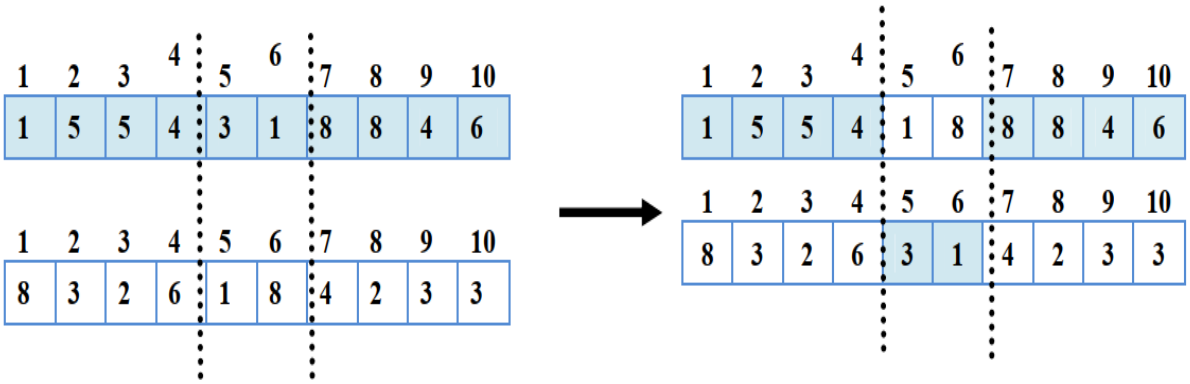



Figure 2.8. Exemple de croisement double.

On réalise ensuite une mutation sur les enfants obtenues lors du croisement.

- **Mutation**

Les opérateurs de mutation produisent un enfant à partir d'un unique individu, en altérant un gène dans un individu selon un facteur de mutation. Ce facteur est la probabilité qu'une mutation soit effectuée sur un individu ; la figure 2.9 ci-dessous illustre cela.

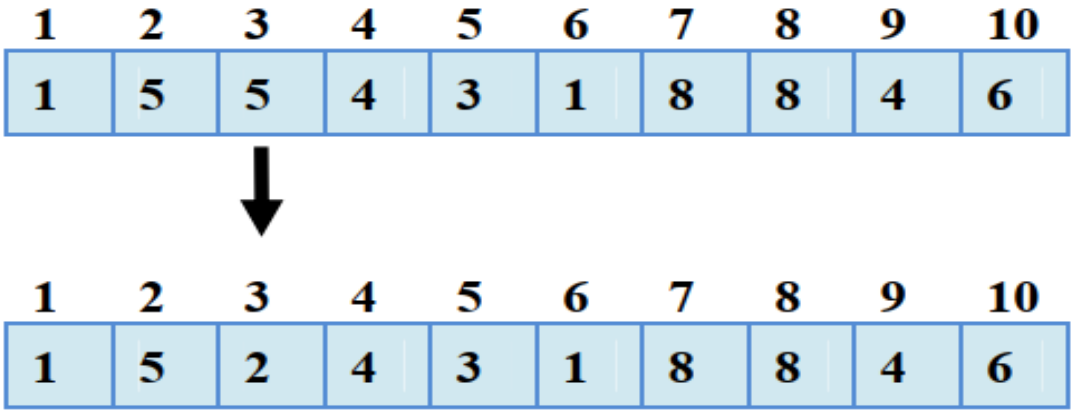



Figure 2.9. Exemple de mutation.

- **Evaluation**

Chaque individu se voit attribuer un score (valeur de fitness), suite au calcul d'une fonction objectif ; chaque individu est donc évalué en fonction de cette valeur, et seuls les chromosomes ayant obtenus une valeur de fitness estimée être assez bonne pour faire partie de la génération sont retenus pour constituer la population de travail de la génération suivante. Ainsi, à chaque

génération le jeu des opérateurs créant de nouveaux chromosomes tend à ce que la population contienne des individus représentant de meilleures solutions [29].

### c) Optimisation par Essaim Particulaire

L'Optimisation par Essaim Particulaire (OEP), ou Particle Swarm Optimization (PSO) en anglais, est une méthode d'optimisation stochastique basée sur une population de solutions. Elle a été développée par Kennedy et Eberhart en 1995. Elle est inspirée du comportement social des colonies d'insectes, des nuées d'oiseaux, des bancs de poissons et bien d'autres sociétés animales évoluant en essaim. En effet, on peut observer chez ces animaux des dynamiques de déplacement relativement complexes, alors qu'individuellement chaque individu a une « intelligence » limitée, et ne dispose que d'une connaissance locale de sa situation dans l'essaim. Des règles simples, telles que « aller à la même vitesse que les autres », « se déplacer dans la même direction », ou encore « rester proche de ses voisins », suffisent pour maintenir la cohésion de l'essaim, permettant ainsi de mettre on œuvre des comportements collectifs complexes et adaptés. L'intelligence globale de l'essaim est la conséquence directe des interactions locales entre les différentes particules de l'essaim [30].

L'essaim de particules correspond à une population d'agents simples, appelés particules. Les particules représentent des solutions potentielles au problème d'optimisation. Elles survolent l'espace de recherche, afin de trouver l'optimum global. Chaque particule possède une position (le vecteur solution) et une vitesse. De plus, elle possède une mémoire lui permettant de se souvenir de sa meilleure performance (en position et en valeur) et de la meilleure performance atteinte par les particules « voisines » (informatrices).

Le déplacement d'une particule est influencé par une combinaison linéaire des trois composantes suivantes :

- Une composante d'inertie : la particule tend à suivre sa direction courante de déplacement ;
- Une composante cognitive : la particule tend à se diriger vers le meilleur site par lequel elle est déjà passée ;
- Une composante sociale : la particule tend à se fier à l'expérience de ses congénères et, ainsi, à se diriger vers le meilleur site déjà atteint par ses voisins.

Dans le cas d'un problème d'optimisation, la qualité d'un site de l'espace de recherche est déterminée par la valeur de la fonction objectif en ce point. La stratégie de déplacement d'une particule est illustrée dans la figure 2.10.

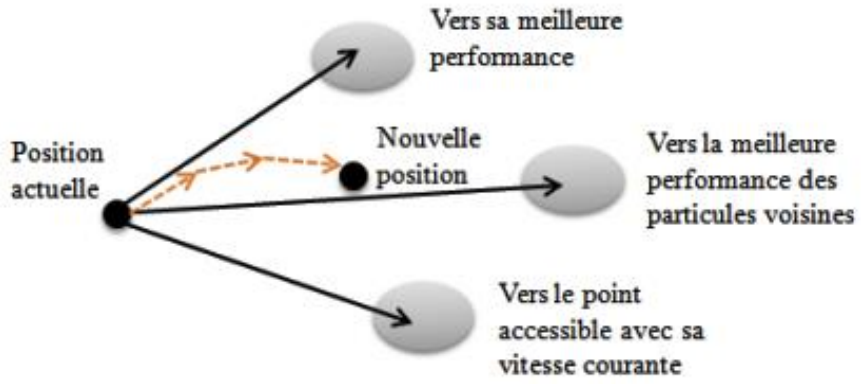



Figure 2.10. La stratégie de déplacement d'une particule.

Dans un espace de recherche de dimension D, la particule *i* de l'essaim est modélisée par son
vecteur position $X_i = (x_{i1};\ x_{i2};\ \dots;\ x_{iD})$ et par son vecteur vitesse $V_i = (v_{i1};\ v_{i2};\ \dots;\ v_{iD})$. Cette particule garde en mémoire la meilleure position par laquelle elle est déjà passée, que l'on note $Pbest_i = (Pbest_{i1};\ Pbest_{i2};\ \dots;\ Pbest_{iD})$. La meilleure position atteinte par toutes les particules de l'essaim est $Gbest_i = (Gbest_{i1};\ Gbest_{i2};\ \dots;\ Gbest_{iD})$.

A l'itération *k+1*, le vecteur vitesse et le vecteur position sont mis à jour à partir de l'équation
(2.1) et de l'équation (2.2), respectivement.

$$V_i^{k+1} = \omega V_i^k + c_1 r_1 \left(Pbest_i^k - X_i^k\right) + c_2 r_2 \left(gGbest^k - X_i^k\right) \qquad (2.1)$$

$$X_i^{k+1} = X_i^k + V_i^{k+1} \qquad (2.2)$$

Où $\omega$ est une constante, nommée coefficient d'inertie, $c_1$ et, $c_2$ sont deux constantes, appelées coefficients d'accélération, $r_1$ et $r_2$ sont deux nombres aléatoires tirés uniformément dans [0,1] à chaque itération et pour chaque dimension.

**$\omega V_i^k$ :** correspond à la composante d'inertie du déplacement, où le paramètre $\omega$ contrôle l'influence de la direction de déplacement sur le déplacement futur. Il est à noter que, dans certaines applications, le paramètre $\omega$ peut être variable ;

$\boldsymbol{c_1 r_1 (Pbest_i^k - X_i^k)}$ **:** correspond à la composante cognitive du déplacement, où le paramètre $c_1$ contrôle le comportement cognitif de la particule ;

$\boldsymbol{c_2 r_2 (gGbest^k - X_i^k)}$ **:** correspond à la composante sociale du déplacement, où le paramètre $c_2$ contrôle l'aptitude sociale de la particule.

La combinaison des paramètres $\omega$, $c_1$ et $c_2$ permet de régler l'équilibre entre les phases de diversification et d'intensification du processus de recherche [31].

L'optimisation par essaim particulaire est un algorithme à population. Il commence par une initialisation aléatoire de l'essaim dans l'espace de recherche. A chaque itération de l'algorithme, chaque particule est déplacée suivant les équations 2.1 et 2.2. Une fois le déplacement des particules effectué, les nouvelles positions sont évaluées.

Les ainsi que sont alors mis à jour. Les différentes étapes du PSO sont présentées dans l'algorithme.

Concernant le critère d'arrêt, il est commun de fixer un nombre maximum d'évaluations de la fonction objectif ou un nombre maximum d'itérations. Cependant, selon le problème posé et les exigences de l'utilisateur, d'autres critères d'arrêt peuvent être utilisés.

Le pseudo-code de cette méthode est représenté dans l'algorithme 4.

**Algorithme** Algorithme d'optimisation par essaim particulaire

**Initialiser** aléatoirement les particules : position et vitesse.
**Evaluer** les positions des particules
**Pour** chaque particule $i$, $Pbest_i = X_i$
**Calculer** *Gbest*
**Tant que** le critère d'arrêt n'est pas satisfait **faire**
**Pour** chaque particule **faire**
**Déplacer** les particules selon les équations 2.4 et 2.5
**Evaluer** les positions des particules
**Mettre** à jour $Pbest_i$
**Fin pour**
**Mettre à jour** *Gbest*
**Fin**

# 4. Conclusion

A travers ce chapitre, nous avons passé en revue le problème de placement des machines virtuelles dans les centres de données Cloud après avoir introduit quelques concepts liés aux problèmes d'optimisation d'une façon générale. Nous avons présenté également un état de l'art sur les différentes méthodes d'optimisation du problème de placement des machines virtuelles en mettant l'accent sur les métaheuristiques. Dans le chapitre suivant, nous présenterons une nouvelle métaheuristique appelée : « Optimisation de l'alimentation des Raies Manta » pour résoudre le problème de placement des machines virtuelles dans le Cloud. Il s'agit d'une méthode bio-inspirée proposée récemment pour résoudre les problèmes d'optimisation des applications d'ingénierie.

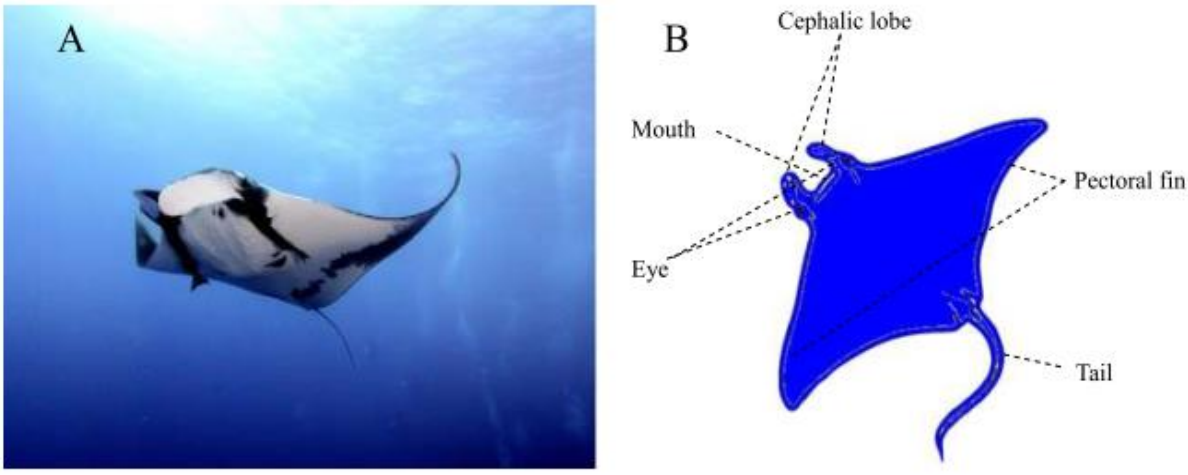
# Chapitre 3 : l’algorithme MRFO pour résoudre placement des machines virtuelles

# 1. Introduction

Comme vu dans le chapitre précèdent, pour résoudre le problème de placement des VMs dans le Cloud, plusieurs métaheuristiques ont été utilisées. Dans ce chapitre, nous utiliseront un nouvel algorithme bio-inspiré appelé "Optimisation de l'alimentation des Raies Manta" (*MRFO : Manta Ray Foraging Optimization*) [32]. Cet algorithme a été proposé récemment afin de résoudre les problèmes d'optimisation pour les applications d'ingénierie. Ce chapitre utilisera l'algorithme MRFO pour la première fois afin d'optimiser le placement des VMs dans les centres de données Cloud en minimisant la consommation d'énergie. Comme l'algorithme MRFO a été proposé pour obtenir le meilleur emplacement pour la nourriture des Raies Manta dans un espace de recherche continu, on a développé une version discrète de cet algorithme vu la nature discrète du problème de placement des machines virtuelles.

# 2. Optimisation de Recherche de nourriture des Raies Manta

Les Raies Manta font partie des créatures marines. Ils ont un corps plat et deux nageoires pectorales. Ils nagent librement comme des oiseaux. En outre, ils ont une paire de lobes verticaux étendus devant leur énorme bouche terminale, comme le montre la figure 3.1. Ils n'ont donc pas de dents pointues ; ils se nourrissent de plancton (animaux microscopiques dans l'eau).




A) Une Raie Manta en quête de nourriture B) la structure d'une Raie Manta

Figure 3.1. Les Raie Manta [32].

# 3. Inspiration de l'algorithme MRFO

Lorsqu'ils se nourrissent, les Raies Manta aspirent de l'eau contenant du plancton dans leur bouche à l'aide de leurs lobes verticaux en forme de corne. Le plancton est alors extrait de l'eau par leurs branchiospines. Les Raies Manta peuvent être classées en deux espèces différentes. L'un est les Raies Manta de récif (Manta alfredi) qui vivent également dans l'ouest et le sud du Pacifique, dans l'océan Indien. Ce type peut atteindre 5,5 m de largeur. L'autre type est les Raies Manta géantes appelées Manta birostris, qui existaient il y a 5 millions d'années. On le trouve dans les océans tempérés chauds, les eaux tropicales et subtropicales. Il peut atteindre 7 m de large. Les Raies Manta se nourrissent de plancton qui n'est pas uniformément ou régulièrement réparti. Le plancton se forme à la suite du reflux et des marées ou du changement de saisons. Ces créatures ont développé diverses stratégies de recherche de nourriture intelligentes. Le premier est la recherche de nourriture en chaîne. Un groupe de Raies Manta se compose de 50 ou plus qui commencent à se nourrir les unes avec les autres. Ils s'alignent les uns derrière les autres, construisent une ligne cohérente. Le plus petit mâle s'appuyait sur les nageoires pectorales de la femelle. Le plancton manqué sera attrapé par ceux qui sont derrière eux. De cette façon, ils peuvent transmettre la majeure partie du plancton à leurs branchies. Une autre stratégie est le cyclone. Cette méthode est utilisée à forte concentration de plancton. Des dizaines de Raies Manta combinées ensemble. Ils génèrent un sommet en spirale en reliant les extrémités des queues aux têtes. Le plancton est tiré vers le bas dans leur bouche ouverte. La stratégie finale est la recherche de nourriture en culbute. Cependant, ce comportement de recherche de nourriture est rare dans la nature, il a été modélisé et développe l'algorithme d'optimisation de la recherche de nourriture des Raies Manta (MRFO). Lorsque les Raies Manta découvrent une source de nourriture (plancton), elles font des culbutes en arrière et tournent autour du plancton en le retirant vers elles. Le mouvement de culbute a des caractéristiques uniques, il est aléatoire, répété, local et cyclique [32].

# 4. Modèle mathématique de l'algorithme MRFO

L'algorithme MRFO s'inspire de trois comportements de recherche de nourriture, à savoir la recherche de nourriture en chaîne, la recherche de nourriture par cyclone et la recherche de nourriture par culbute. Les modèles mathématiques sont décrits ci-dessous.

## 4.1. La recherche de nourriture en chaîne (*Chain foraging*)

Les Raies Manta recherchent le plancton et, lorsqu'elles déterminent sa position, elles nagent vers lui. La meilleure position est celle qui contient une concentration plus élevée de plancton. Les Raies Manta font la queue tête-bêche et forment une chaîne de recherche de nourriture. Chaque mouvement individuel n'est pas seulement vers la nourriture mais aussi vers un autre devant. Chaque individu a mis à jour sa position en fonction de la meilleure solution obtenue jusqu'à présent et de la solution de celle en tête. Le comportement de la chaîne est illustré à la Figure 3.2. Le modèle mathématique de recherche de nourriture en chaîne peut être représenté comme suit [32] :

$$x_i^d(t+1) = \begin{cases} x_i^d(t) + r\,.\,\left(x_{best}^d(t) - x_i^d(t)\right) + \alpha\,.\left(x_{best}^d(t) - x_i^d(t)\right) & i = 1 \\ x_i^d(t) + r\,.\,\left(x_{i-1}^d(t) - x_i^d(t)\right) + \alpha\,.\left(x_{best}^d(t) - x_i^d(t)\right) & i = 2, \dots, N \end{cases} \quad (1)$$

$$\alpha = 2\,.r\,.\sqrt{|\log(r)|} \quad (2)$$

$x_i^d(t)$ est la position du $i$ ème individu au temps t dans la $d$ ème dimension.

$r$ est un nombre aléatoire dans la plage de [0, 1].

$\alpha$ est un coefficient de poids.

$x_{best}^d(t)$ est le plancton à forte concentration.

La figure 3.2 illustre ce comportement de recherche de nourriture dans un espace 2-D. La mise à jour de la position du $i$ ème individu est déterminée par la position $x_{i-1}(t)$ du (i-1) ème individu actuel et la position $x_{best}(t)$ de l'aliment.

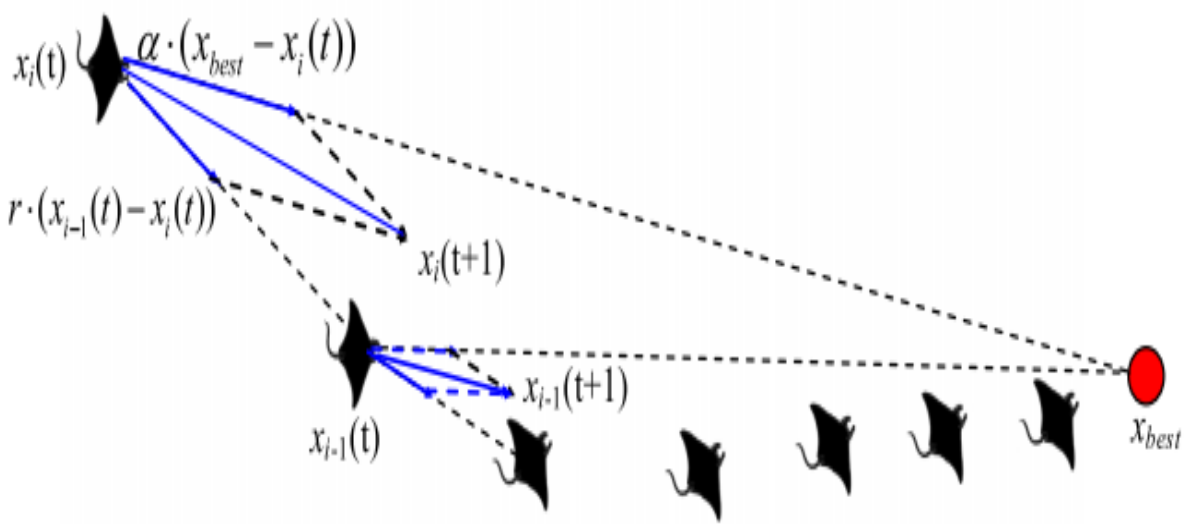



Figure 3.2. Comportement de recherche de nourriture en chaîne dans un espace 2-D.

## 4.2. La recherche de nourriture par cyclone (*Cyclone foraging*)

Lorsqu'un banc de Raies Manta reconnaît une parcelle de plancton en eau profonde, elle forme une longue chaîne de recherche de nourriture et nage vers la nourriture par une spirale. La figure 3.3 illustre le comportement de recherche de nourriture du cyclone dans un espace 2D. Un individu non seulement suit celui qui le précède, mais se déplace uniquement vers la nourriture le long d'un chemin en spirale. L'équation mathématique modélisant le mouvement en forme de spirale des Raies Manta dans un espace 2D peut être définie comme :

$$\begin{cases} X_i(t+1) = X_{best} + r \,.\, (X_{i-1}(t) - X_i(t)) + e^{b\omega} \,.\, \cos(2\pi\omega) \,.\, (X_{best} - X_i(t)) \\ Y_i(t+1) = Y_{best} + r \,.\, (Y_{i-1}(t) - Y_i(t)) + e^{b\omega} \,.\, \sin(2\pi\omega) \,.\, (Y_{best} - Y_i(t)) \end{cases} \quad (3)$$

$\omega$ est un nombre aléatoire dans [0, 1].

Ce comportement de mouvement peut être étendu à un espace n-D. Pour plus de simplicité, ce modèle mathématique de recherche de nourriture cyclonique peut être défini comme

$$x_i^d(t+1) = \begin{cases} x_{best}^d(t) + r \,.\, \left(x_{best}^d(t) - x_i^d(t)\right) + \beta \,.\, \left(x_{best}^d(t) - x_i^d(t)\right) & i = 1 \\ x_{best}^d(t) + r \,.\, \left(x_{i-1}^d(t) - x_i^d(t)\right) + \beta \,.\, \left(x_{best}^d(t) - x_i^d(t)\right) & i = 2, \dots, N \end{cases} \quad (4)$$

$$\beta = 2e^{r_1} \frac{T - t + 1}{T} \,.\, \sin(2\pi r_1) \quad (5)$$

$\beta$ est le coefficient de poids.

$T$ est le nombre maximal d'itérations.

$r_1$ est le nombre rand dans [0, 1].

Tous les individus effectuent la recherche au hasard par rapport à la nourriture comme position de référence, de sorte que la recherche de nourriture par le cyclone a une bonne exploitation pour la région avec la meilleure solution trouvée jusqu'à présent. Ce comportement est également utilisé pour améliorer considérablement l'exploration. Nous pouvons forcer chaque individu à rechercher une nouvelle position loin de la meilleure actuelle en attribuant une nouvelle position aléatoire dans tout l'espace de recherche comme position de référence. Ce mécanisme se concentre principalement sur l'exploration et permet à MRFO de réaliser une recherche globale étendue, son équation mathématique est présentée ci-dessous

$$x_{rand}^{d} = Lb^{d} + r\,.\,(Ub^{d} - Lb^{d}) \tag{6}$$

$$x_i^d(t+1) = \begin{cases} x_{rand}^{d} + r\,.\,\left(x_{rand}^{d} - x_i^d(t)\right) + \beta\,.\left(x_{rand}^{d} - x_i^d(t)\right) & i = 1 \\ x_{rand}^{d} + r\,.\,\left(x_{i-1}^{d}(t) - x_i^d(t)\right) + \beta\,.\left(x_{rand}^{d} - x_i^d(t)\right) & i = 2, \dots, N \end{cases} \tag{7}$$

$x_{rand}^{d}$ est une position aléatoire produite aléatoirement dans l'espace de recherche.

$Lb^{d}$ et $Ub^{d}$ sont respectivement les limites inférieure et supérieure de la $d$ ème dimension.

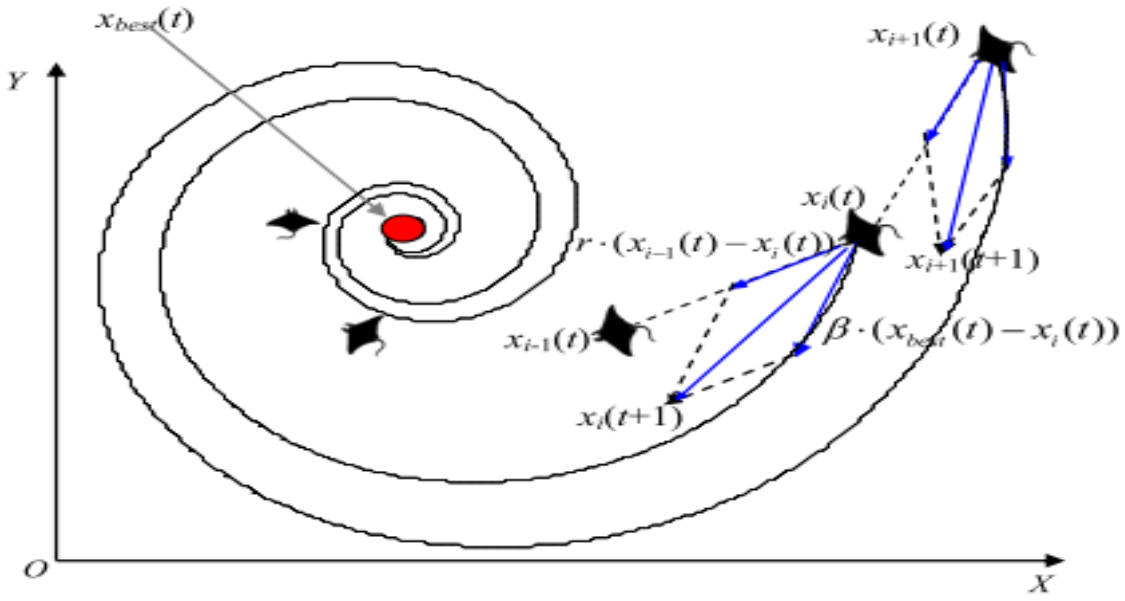



Figure 3.3. Comportement de recherche de nourriture des cyclones dans un espace 2-D.

### 4.3. La recherche de nourriture en culbute (*Somersault foraging*)

Chaque individu Manta se déplace autour de la nourriture et fait une culbute (saut périlleux) vers une nouvelle position. Ils mettent toujours à jour leurs positions autour d'une zone de concentration de plancton plus élevée (meilleure solution jusqu'à présent). Le modèle mathématique de recherche de nourriture en culbute peut être dérivé selon l'équation suivante [32].

$$x_i^d(t+1) = x_i^d(t) + S \cdot \left(r_2 \cdot x_{best}^d - r_3 \cdot x_i^d(t)\right), i = 1, \dots, N \quad (8)$$

$S$ est le facteur de culbute qui détermine la plage de culbute des Raies Manta.

$r_2$ et $r_3$ sont deux nombres aléatoires dans [0, 1].

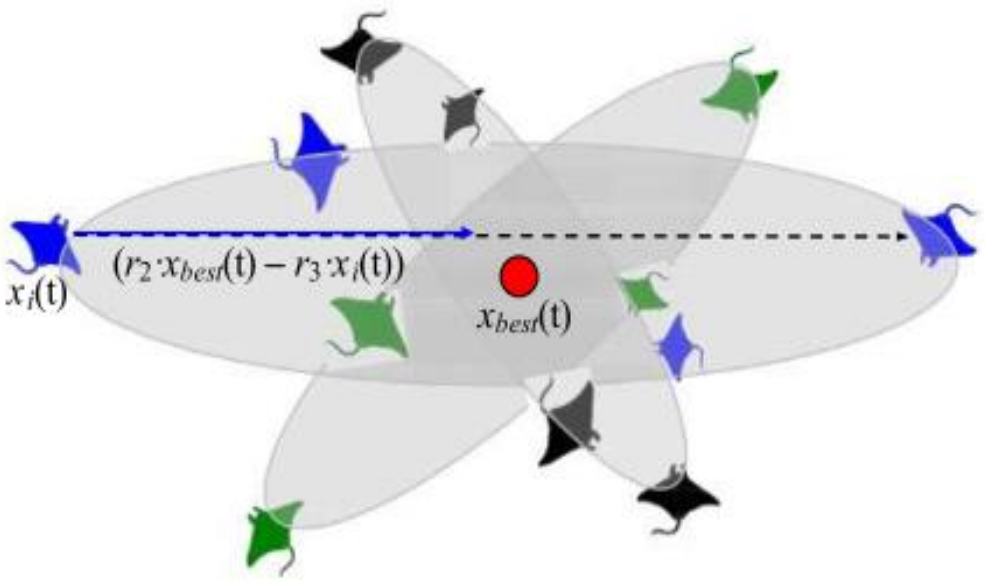



Figure 3.4. Comportement de recherche de nourriture en culbute dans le MRFO.

## 5. L'algorithme MRFO

Initialiser la taille de la population N, le nombre maximal d'itérations T et chaque Raie Manta

$x_i(t) = x_l + rand \cdot (x_u - x_l)$ For i = 1, ..., N and t = 1. Calculez le fitness de chaque individu $f_i = f(x_i)$ et obtenez la meilleure solution trouvée jusqu'à présent $x_{best}$, $x_u$ où et $x_l$ sont respectivement les limites supérieure et inférieure de l'espace du problème.

WHILE le critère d'arrêt n'est pas satisfait DO

FOR $i$ = 1 TO $N$ DO

IF $rand < 0.5$ THEN // La recherche de nourriture par cyclone

IF $t/\ T_{max} < rand$ THEN

$x_{rand} = x_l + rand\ .(x_u - x_l)$

$$x_i(t+1) = \begin{cases} x_{rand} + r\ .\ \left(x_{rand} - x_i(t)\right) + \beta\ .\ \left(x_{rand} - x_i(t)\right) & i = 1 \\ x_{rand} + r\ .\ \left(x_{i-1}(t) - x_i(t)\right) + \beta\ .\ \left(x_{rand} - x_i(t)\right) & i = 2, \dots, N \end{cases}$$

ELSE

$$x_i(t+1) = \begin{cases} x_{best} + r\ .\ \left(x_{best} - x_i(t)\right) + \beta\ .\ \left(x_{best} - x_i(t)\right) & i = 1 \\ x_{best} + r\ .\ \left(x_{i-1}(t) - x_i(t)\right) + \beta\ .\ \left(x_{best} - x_i(t)\right) & i = 2, \dots, N \end{cases}$$

ELSE IF.

ELSE // La recherche de nourriture en chaîne

$$x_i(t+1) = \begin{cases} x_i(t) + r\ .\ \left(x_{best} - x_i(t)\right) + \alpha\ .\ \left(x_{best} - x_i(t)\right) & i = 1 \\ x_i(t) + r\ .\ \left(x_{i-1}(t) - x_i(t)\right) + \alpha\ .\ \left(x_{best} - x_i(t)\right) & i = 2, \dots, N \end{cases}$$

ELSE IF.

Calculer le fitness de chaque individu $f(x_i(t+1))$.

IF $f\left(x_i(t+1)\right) < f(x_{best})$ THEN $x_{best} = x_i(t+1)$

END FOR

// La recherche de nourriture en culbute

FOR $i$ = 1 TO $N$ DO

$x_i(t+1) = x_i(t) + S\ .\ \left(r_2\ .x_{best} - r_3\ .x_i(t)\right)$

Calculez le fitness de chaque individu $f(x_i(t+1))$.

IF $f\left(x_i(t+1)\right) < f(x_{best})$ THEN $x_{best} = x_i(t+1)$

END FOR

END WHILE

Retourner la meilleure solution trouvée jusqu'à présent $x_{best}$

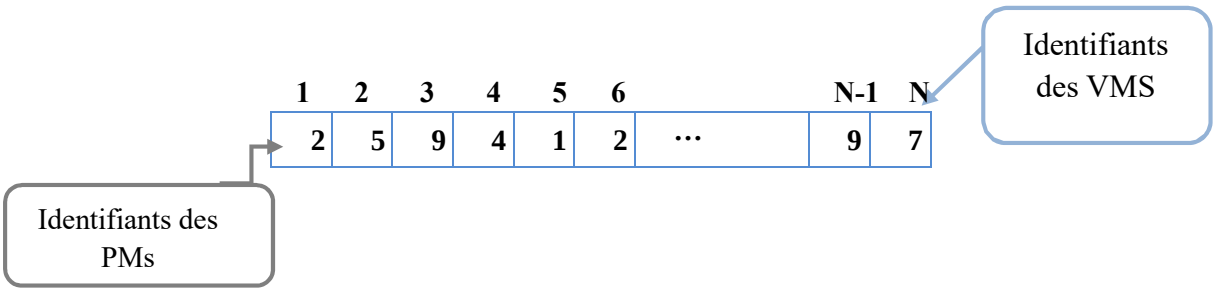
---

# 6. L'algorithme MRFO pour le placement des machines virtuelles dans le Cloud

## 6.1. Codification des individus

Chaque individu est représenté par un vecteur où l'index du tableau présente les identifiants des VMs et le contenu du tableau présente les identifiants des PMs.

**Exemple**




## 6.2. Formulation du problème

Le placement des machines virtuelles est très important pour l'utilisation efficace des ressources dans le centre de données. Chaque hôte du centre de données à deux états : soit l'hôte est à l'état actif, soit à l'état de veille. L'état actif indique que l'hôte est alloué à des fins d'exécution tandis que l'état de veille indique qu'il n'est alloué à aucune des machines. Le problème de placement de VM dans le Cloud Computing est un problème d'optimisation de complexité NP-complet. Pour trouver la solution optimale à ce problème on a utilisé l'algorithme MRFO avec les notations suivantes :

| | |
|---|---|
| N | Nombre de machines virtuelles |
| M | Nombre de machines physiques |
| $v_i$ | Le numéro i machine virtuelle |
| $p_j$ | Le nombre j machine physique |
| $vp_{ij}$ | La valeur binaire indiquant si la machine virtuelle $v_i$ est affectée au nœud physique $p_j$ |
| V | L'ensemble des N machines virtuelles, à savoir $v_1$, $v_2$, ..., $v_N$ |
| P | L'ensemble des M machines physiques, soit $p_1$, $p_2$, ..., $p_N$ |
| $u_j$ | Le pourcentage d'utilisation du CPU de $p_j$ |
| $e_j$ | La consommation d'énergie de $p_j$ |
| $e_{max}^j$ | Max la consommation d'énergie de $p_j$ quand $u_j = 100\%$ |
| $e_{idle}^j$ | Ralenti la consommation d'énergie de $p_j$ quand $u_j = 0\%$ |
| $v_{cpu}^i$ | La demande CPU de $v_i$ |
| $v_{mem}^i$ | La demande de RAM de $v_i$ |
| $v_{net}^i$ | La capacité de bande passante réseau de $v_i$ |
| $p_{cpu}^j$ | La capacité du processeur de $p_j$ |
| $p_{mem}^j$ | La capacité de RAM de $p_j$ |
| $p_{net}^j$ | La capacité de bande passante réseau de $p_j$ |

L'affectation de V à P, représentée par une matrice. Ou chaque $vi$ affecté a $pj$ présenté par 1, si non 0 [34].

$$\begin{pmatrix} vp_{11} & \cdots & vp_{1M} \\ \vdots & \ddots & \vdots \\ vp_{N1} & \cdots & vp_{NM} \end{pmatrix} \qquad (9)$$

Où

$$vp_{ij} = \begin{cases} 1, & si\ v_i\ est\ affecté\ à\ p_j \\ 0, & autrement \end{cases} \quad 1 \le i \le N\ , 1 \le j \le M \qquad (10)$$

Pour une affectation donnée V P, l'utilisation CPU de $p_j$ peut être calculée par [34] :

$$u_j = \frac{\sum_{i=1}^{N} v_{cpu}^{i} * vp_{ij}}{p_{cpu}^{j}} \qquad (11)$$

## 6.3. Modèle de l'énergie

La consommation d'énergie de $pj$ peut être calculée par l'équation suivante :

$$e_j = \begin{cases} 0, & if \sum_{i=1}^{N} vp_{ij} = 0 \\ \left(e_{max}^{j} - e_{idle}^{j}\right) * \frac{u_j}{100} + e_{idle}^{j}, & autrement \end{cases} \qquad (12)$$

Lorsque $\sum_{i=1}^{N} vp_{ij} = 0$ , cela signifie qu'aucune machine virtuelle n'est affectée à $pj$, il peut donc être éteint et ne consomme pas d'énergie [34].

L'objectif de la recherche est de trouver une affectation V P qui minimise :

$$\sum_{j=1}^{M} e_j \qquad (13)$$

## 6.4. Contraintes

Il faut prendre en compte un ensemble des contraintes qui doivent être remplis : Premièrement, une machine virtuelle ne peut être placée que dans une seule machine physique, l'équation (14) exprime cette contrainte [34];

$$\forall i, \sum_{j=1}^{M} vp_{ij} = 1 \qquad (14)$$

Deuxièmement, le CPU, la mémoire ou les ressources réseau totales affectées aux VMs d'un PM ne puissent pas dépasser les capacités respectives de l'hôte. Cette contrainte est présentée par les trois équations (15), (16), (17) [34].

$$\forall j, \sum_{i=1}^{N} v_{cpu}^{i} * \ vp_{ij} \leq \ p_{cpu}^{j} \qquad (15)$$

$$\forall j, \sum_{i=1}^{N} v_{mem}^{i} * \ vp_{ij} \leq \ p_{mem}^{j} \qquad (16)$$

$$\forall j, \sum_{i=1}^{N} v_{net}^{i} * \ vp_{ij} \leq \ p_{net}^{j} \qquad (17)$$

## 6.5. Algorithme du MRFO

Initialiser la taille de la population N, le nombre maximal d'itérations T et chaque solution $x_i(t) = x_l + rand\,.(x_u - x_l)$ For i = 1, ..., N and t = 1. Calculez l'énergie de chaque solution $f_i = f(x_i)$ et obtenez la meilleure solution trouvée jusqu'à présent $x_{best}$, $x_u$ où et $x_l$ sont respectivement les limites supérieure et inférieure de l'espace du problème.

WHILE le critère d'arrêt n'est pas satisfait DO

FOR *i* = 1 TO *N* DO

IF $rand < 0.5$ THEN

IF $t/\ T_{max} < rand$ THEN

*Mise à jour du placement de la VM selon le Cyclone Rand*

ELSE

*Mise à jour du placement de la VM selon le Cyclone best*

ELSE IF.

ELSE

*Mise à jour du placement de la VM selon le Chaîne*

ELSE IF.

Calculez l'énergie de chaque solution

Mettre à jour la meilleure solution

END FOR

FOR $i$ = 1 TO $N$ DO

*Mise à jour du placement de la VM selon le Culbute*

Calculez l'énergie de chaque solution

Mettre à jour la meilleure solution

END FOR

END WHILE

Retourner la meilleure solution trouvée jusqu'à présent $x_{best}$

---

## 6.6. Description de l'algorithme

L'algorithme MRFO commence par générer une population aléatoire dans le domaine du problème. A chaque itération, chaque individu met à jour sa position par rapport à la fois à celui qui le précède et à la position de référence. La valeur de t/T diminue de 1/T à 1 pour effectuer respectivement une recherche exploratoire et exploitante. La meilleure solution courante est

choisie comme position de référence pour l'exploitation lorsque t/T < rand, tandis qu'une position aléatoire générée aléatoirement dans l'espace de recherche est choisie comme position de référence pour l'exploration lorsque t/T > rand. Pendant ce temps, selon le nombre aléatoire, MRFO peut basculer entre le comportement de *Mise à jour du placement de la VM selon le Chaîne* selon l'équation (1) et le comportement de *Mise à jour du placement de la VM selon le Cyclone*. Et selon t/T < rand, MRFO peut basculer entre le comportement de *Mise à jour du placement de la VM selon le Cyclone Rand* selon l'équation (7) et le comportement de *Mise à jour du placement de la VM selon le Cyclone best* selon l'équation (4). Ensuite, les individus mettent à jour leurs propres positions par rapport à la meilleure position trouvée jusqu'à présent par *Mise à jour du placement de la VM selon le Culbute* selon l'équation (8). Toutes les mises à jour et tous les calculs sont effectués de manière interactive jusqu'à ce que le critère d'arrêt soit satisfait. Finalement, la position et la valeur de l'énergie du meilleur individu sont renvoyées.

Vue la nature discrète du problème du placement des VMs, les positions de individues réelles sont arrondie afin d'obtenir des valeurs entières. Si la position d'un individu déborde l'espace de recherche (inférieure à la première PM 0, ou supérieure à la dernière PM M) on a utilisé la méthode suivante :

Si num_PM < 0 alors num_PM ← 0 ;

Si num_PM > M alors num_PM ← M.

# 7. Conclusion

Dans ce chapitre, nous avons utilisé une nouvelle méta-heuristique bio-inspiré appelé MRFO pour le placement des machines virtuelles dans un environnement de Cloud. Ce placement garantira une optimisation des ressources allouées à ces machines. Nous avons développé une version discrète de l'algorithme MRFO vue la nature discrète du problème étudié. L'objectif de cette optimisation était la minimisation de l'énergie consommée dans le centre de données Cloud. Cependant, on peut s'étendre pour optimiser d'autres métriques, notamment le taux d'utilisation des ressources et le cout de placement. Dans le chapitre suivant, on présentera l'environnement de simulation ainsi que les résultats expérimentaux de l'algorithme proposé à travers le simulateur CloudSim plus.

# Chapitre 4 : Implémentation

# 1. Introduction

Dans le chapitre précédent, nous avons présenté la problématique de placement des machines virtuelles sur des machines physiques, et nous avons proposé une approche pour résolut les problèmes, ce qui est l'algorithme de MRFO. Et ce que nous allons faire dans cette partie est une implémentation de ce que nous avons présenté précédemment.

Dans ce chapitre nous allons présenter, le langage utilisé pour programmer notre application ce qu'est le langage Java, l'environnement de développent NetBeans, et nous allons présenter en détail le CloudSim (s'architecture, ses classes de base), puis nous allons présenter notre implémentation avec toutes les étapes, jusqu'à ce que nous arrivions au résultat final. Et nous allons finir notre chapitre avec une conclusion.

# 2. Langage et l'environnement de développement

Pour implémenter notre approche, nous allons utiliser le langage de programmation java, dans un environnement de développement intégré (EDI) NetBeans, et nous allons utiliser aussi le simulateur de Cloud CloudSim.

## 2.1. Langage de programmation 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.

Java est à la fois un langage de programmation et un environnement d'exécution. 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 avec peu ou pas de modifications... C'est la plate-forme qui garantit la portabilité des applications développées en Java.

Le langage reprend en grande partie la syntaxe du langage C++, très utilisé par les informaticiens. Néanmoins, Java a été épuré des concepts les plus subtils du C++ et à la fois les plus déroutants, tels que l'héritage multiple remplacé par l'implémentation des interfaces. Les concepteurs ont privilégié l'approche orientée objet de sorte qu'en Java, tout est objet à l'exception des types primitifs (nombres entiers, nombres à virgule flottante, etc.).

Java permet de développer des applications autonomes mais aussi, et surtout, des applications client-serveur. Côté client, les applets sont à l'origine de la notoriété du langage. C'est surtout côté serveur que Java s'est imposé dans le milieu de l'entreprise grâce aux servlets, le pendant serveur des applets, et plus récemment les JSP (JavaServer Pages) qui peuvent se substituer à PHP, ASP et ASP.NET.

Les applications Java peuvent être exécutées sur tous les systèmes d'exploitation pour lesquels a été développée une plate-forme Java, dont le nom technique est JRE (Java Runtime Environnement - Environnement d'exécution Java). Cette dernière est constituée d'une JVM (Java Virtual Machine - Machine Virtuelle Java), le programme qui interprète le code Java et le convertit en code natif. Mais le JRE est surtout constitué d'une bibliothèque standard à partir de laquelle doivent être développés tous les programmes en Java. C'est la garantie de portabilité qui a fait la réussite de Java dans les architectures client-serveur en facilitant la migration entre serveurs, très difficile pour les gros systèmes [35].


Figure 4.1. Logo de langage Java.

## 2.2. L'environnement de développement NetBeans

NetBeans est un environnement de développement intégré (EDI), placé en open source par Sun en juin 2000 sous licence CDDL et GPLv2 (Common Development and Distribution License). En plus de Java, NetBeans permet également de supporter différents autres langages, comme Python, C, C++, JavaScript, XML, Ruby, PHP et HTML. Il comprend toutes les caractéristiques d'un IDE moderne (éditeur en couleur, projets multi-langage, refactoring, éditeur graphique d'interfaces et de pages Web).

NetBeans est disponible sous Windows, Linux, Solaris (sur x86 et SPARC), Mac OS X ou sous une version indépendante des systèmes d'exploitation (requérant une machine virtuelle

Java). Un environnement Java Development Kit JDK est requis pour les développements en Java [36].

## 2.3. CloudSim

Le simulateur CloudSim a été développé au CLOUDS Laboratory de Melbourne en Australie.

Ce simulateur possède plusieurs fonctionnalités comme :

- Le support pour la modélisation et la simulation à grande échelle d'infrastructure de Cloud Computing, y compris des centres de données sur un seul nœud physique.
- Une plateforme indépendante pour la modélisation des DataCenters, des brokers, de l'ordonnancement et des politiques d'allocation des ressources.

CloudSim est caractérisé par :

- La disponibilité de moteur de virtualisation, ce qui facilite la création et la gestion de services virtualisés multiple, indépendants et hébergés sur un nœud du Datacenter.
- La flexibilité pour commuter entre l'allocation en espace partagé et en temps partagé des cœurs de traitement au service virtualisés. [37]

CloudSim a deux politiques d'ordonnancement :

- **Space shared**

Dans space shared, l'ordonnanceur donne une tache à une machine virtuelle à un instant donné et après la terminaison de cette tache une autre tâche est lancée sur cette machine virtuelle.

- **Time shared**

Dans time shared, l'ordonnanceur donne toutes les taches à une machine virtuelle en même temps. Il partage le temps entre toutes les taches et les planifie simultanément sur la machine virtuelle.

## 2.4. Architecture de CloudSim

L'architecture de CloudSim est organisée sous formes de plusieurs couches. Les différentes couches sont représentées dans la figure 4.2 suivante :

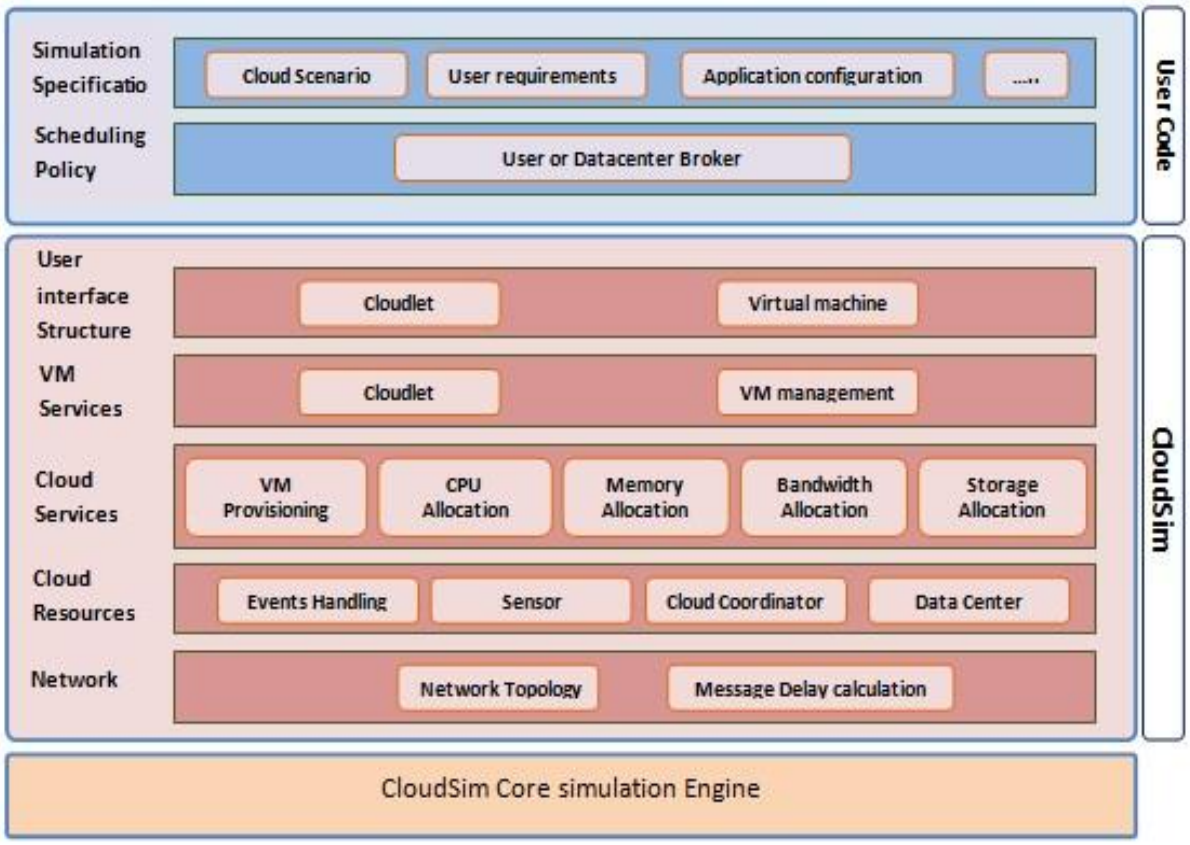



Figure 4.2. Architecture de CloudSim.

La première couche « User Code » contient les informations de base pour le fonctionnement des hôtes et des applications comme le nombre de Data Center, de machines physiques, de machines virtuelles et de tâches (Cloudlets).

Nous trouvons dans cette couche plusieurs paramètres liés aux :

- Caractéristiques des DataCenter : architecture, système d'exploitation, coûts de fonctionnement.
- Caractéristiques des hôtes : nombre de CPU, capacité des CPU, capacité mémoire, capacité de stockage, bande passante.
- Caractéristiques des VMs : capacité CPU, nombre de CPU, capacité mémoire, capacité de stockage.

- Caractéristiques des taches : nombre d'instructions à exécuter, taille des fichiers d'entrée et de sortie, le choix des stratégies d'ordonnancement du broker, etc.

La deuxième couche « CloudSim » contient cinq couches comme mentionner dans la figure 4.2.

Cette couche, permet d'allouer les Cloudlets aux machines virtuelles, de gérer les stratégies d'allocation des machines virtuelles et les topologies des différents réseaux, de mettre une communication possible entre différentes couches, de fournir l'interaction entre l'utilisateur et le simulateur et de fournir un support pour la modélisation des environnements dataCenters virtualisés avec des classes permettant la gestion et la configuration des machines virtuelles, mémoire, stockage, etc.

Les classes principales implémentées dans ces couches sont les suivantes : Datacenter, datacenterbroker, vm, cloudlet, host.

**Datacenter :** Elle permet de modéliser l'infrastructure du noyau du service (matériel, logiciel) dans le Cloud compting.

Il contient un ensemble de machines physiques appelées Hosts, et chaque Datacenter implémente un ensemble d'algorithmes pour l'allocation de la bande passante, la mémoire et le stockage aux différents hosts et machines virtuelles du Cloud.

**DatacenterBroker :** Elle permet de modéliser le Broker qui est le responsable de la médiation des négociations entre les utilisateurs et les prestataires de service selon les conditions de qualité de service des utilisateurs. Les négociations se font pour une allocation de ressources qui répond aux besoins de QoS des utilisateurs. Il permet aussi de déployer les tâches de service. Il déploie aussi les taches de service à travers les Clouds.

**Vm :** Elle permet de modéliser une instance de machine virtuelle qui est gérée et hébergée par une machine physique. Chaque composant VM a accès a un composant qui stock plusieurs caractéristique liées à une VM telles que : la mémoire, le processeur et la capacité de stockage.

**Cloudlet** : Elle permet de modéliser les services d'application du Cloud (comme la livraison, les réseaux sociaux et le workflow d'affaires). CloudSim représente la complexité d'une application en fonction de ses besoins informatiques. Chaque service d'application a une taille d'instruction pré-assignée et la quantité de flux de transfert de données qu'il doit entreprendre au cours de son cycle de vie. Cette classe peut également être étendue pour supporter la

modélisation de la performance et d'autres paramètres de composition pour les applications telles que les transactions dans les applications orientées bases de données.

**Host :** Elle permet de modéliser une ressource physique comme le serveur de stockage ou de calcul. Elle encapsule des informations importantes telles que la quantité de mémoire et de stockage, le type de cœurs de traitement (pour représenter une machine multi-core), une politique d'allocation pour le partage de la puissance du traitement entre les machines virtuelles et les politiques d'attribution de mémoire et de bande passante aux machines virtuelles.

La dernière couche de l'architecture représente le moteur de simulation d'événements discrets javasim, qui prend plusieurs fonctionnalités de base telles que la mise en file d'attente et le traitement des événements, la création de Cloud entités système (services, hôte, Datacenter, broker, VMs) [38].

# 3. Implémentation

## 3.1. Interface principale

Le lancement de l'application donne droit à la page accueil suivante :

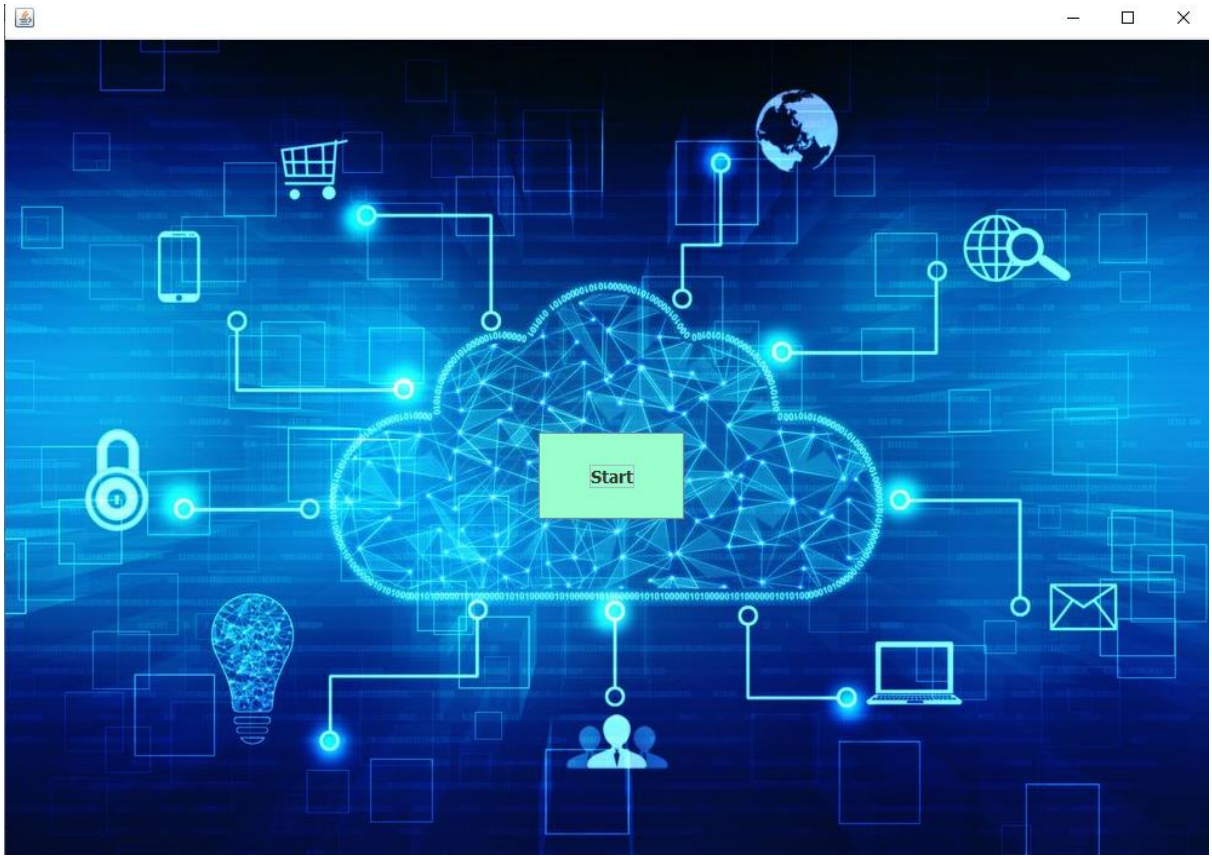



Figure 4.3. Interface principale.

### 3.2. Configuration de la Manta Ray

La figure 4.4 permet de configurer Manta Ray en remplissant les informations suivantes : le facteur de culbute (s), le nombre d'itération (T), la taille de la population (N).

Le bouton « Next » pour enregistrer les informations et passer à l'interface suivante.

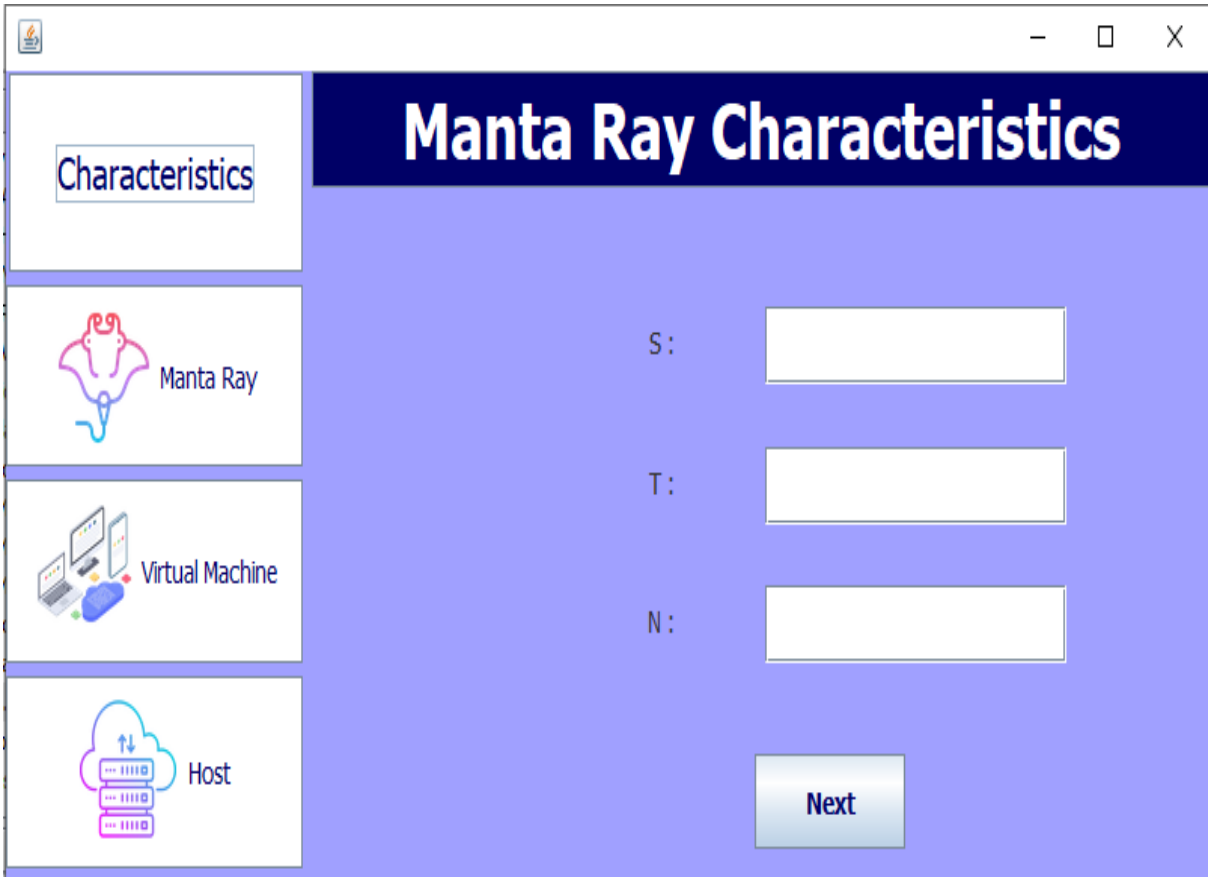



Figure 4.4. Interface de Manta Ray.

### 3.3. Configuration de la machine virtuelle

La figure 4.5 permet de configurer les machines virtuelles en remplissant les informations suivantes : le nombre de coeurs cpu de vm, la vitesse de cpu en MIPS, la ram en MB, stockage en MB, la bande passante en Mbit/s, le nombre de vm. Pour la création des machines virtuelles, il faut cliquer sur «create».

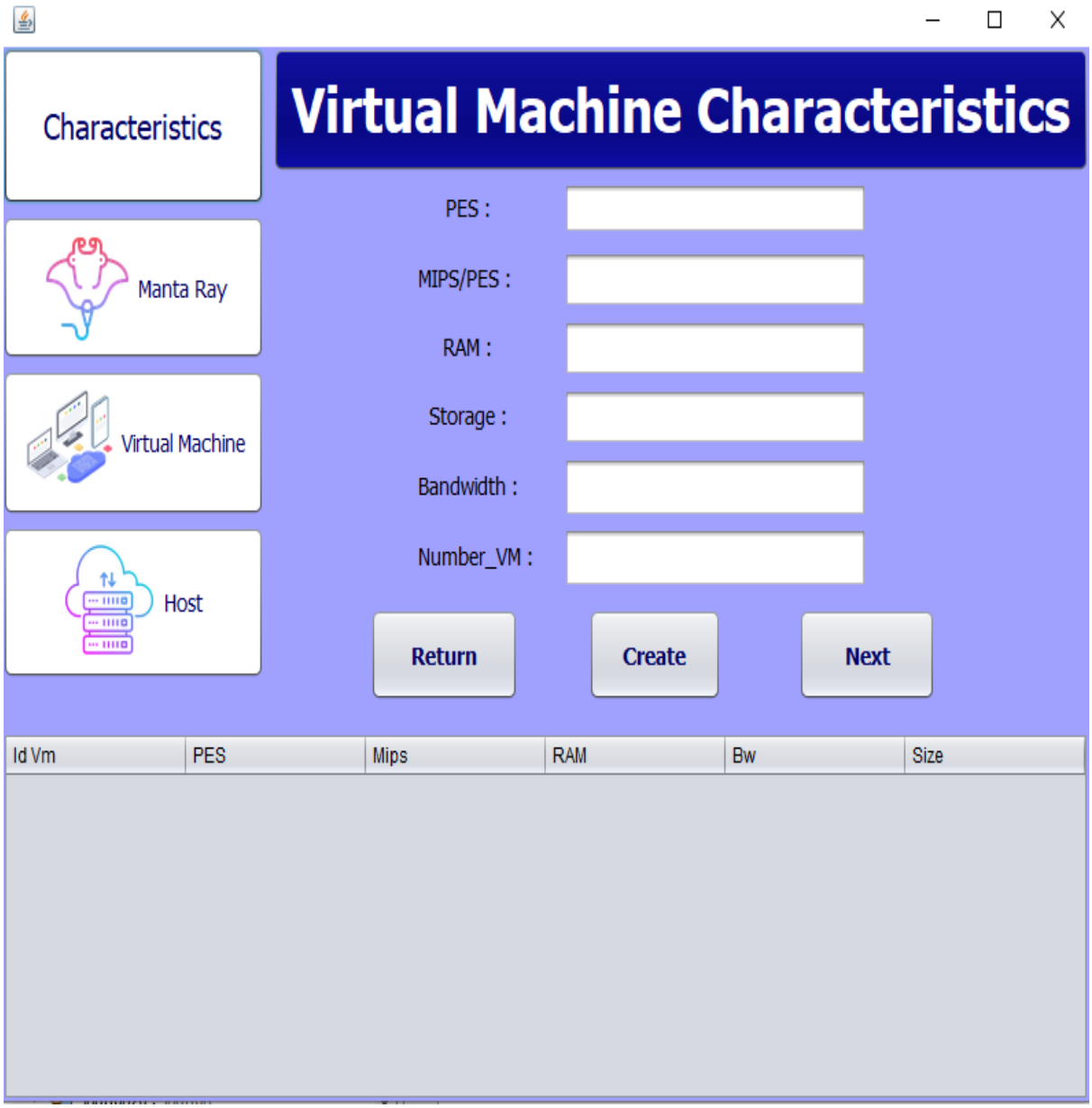



Figure 4.5. Interface de machine virtuelle.

### 3.4. Configuration des machines physiques (Host)

La figure 4.6 permet de configurer les machines physiques en remplissant les informations suivantes : le nombre d'hôtes, le nombre des coeurs de cpu, la vitesse de cpu en MIPS, la ram en MB, stockage en MB, la bande passante en Mbit/s. Puis pour la création, il clique sur «create».

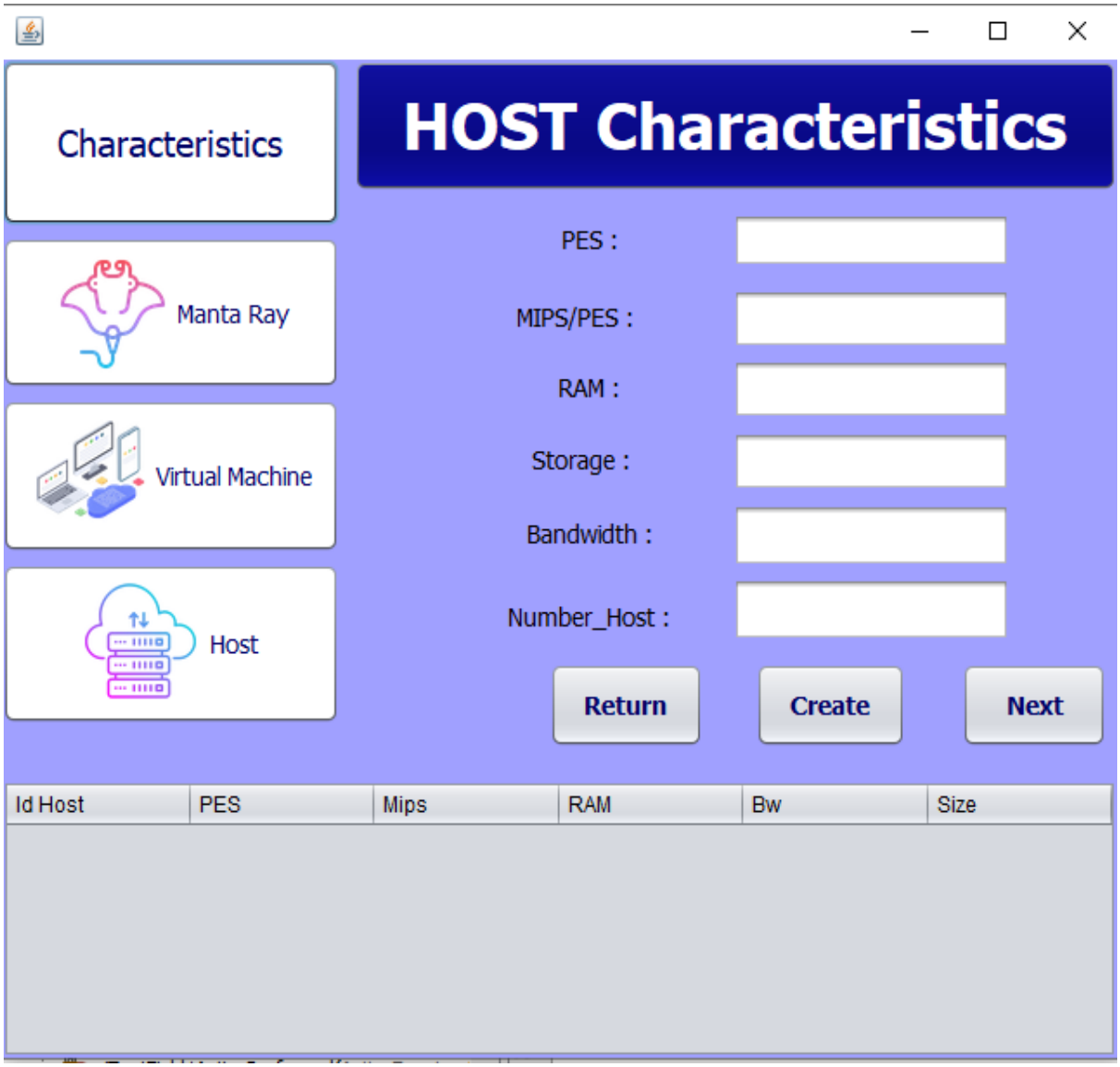



Figure 4.6. Interface de machine physique.

## 3.5. Interface d'affichage de résultat

Le bouton « Simulation » permet de démarrer la simulation, Après la terminaison de la simulation, le résultat est affiché dans le tableau qui contient le meilleur individu et leur Energie.

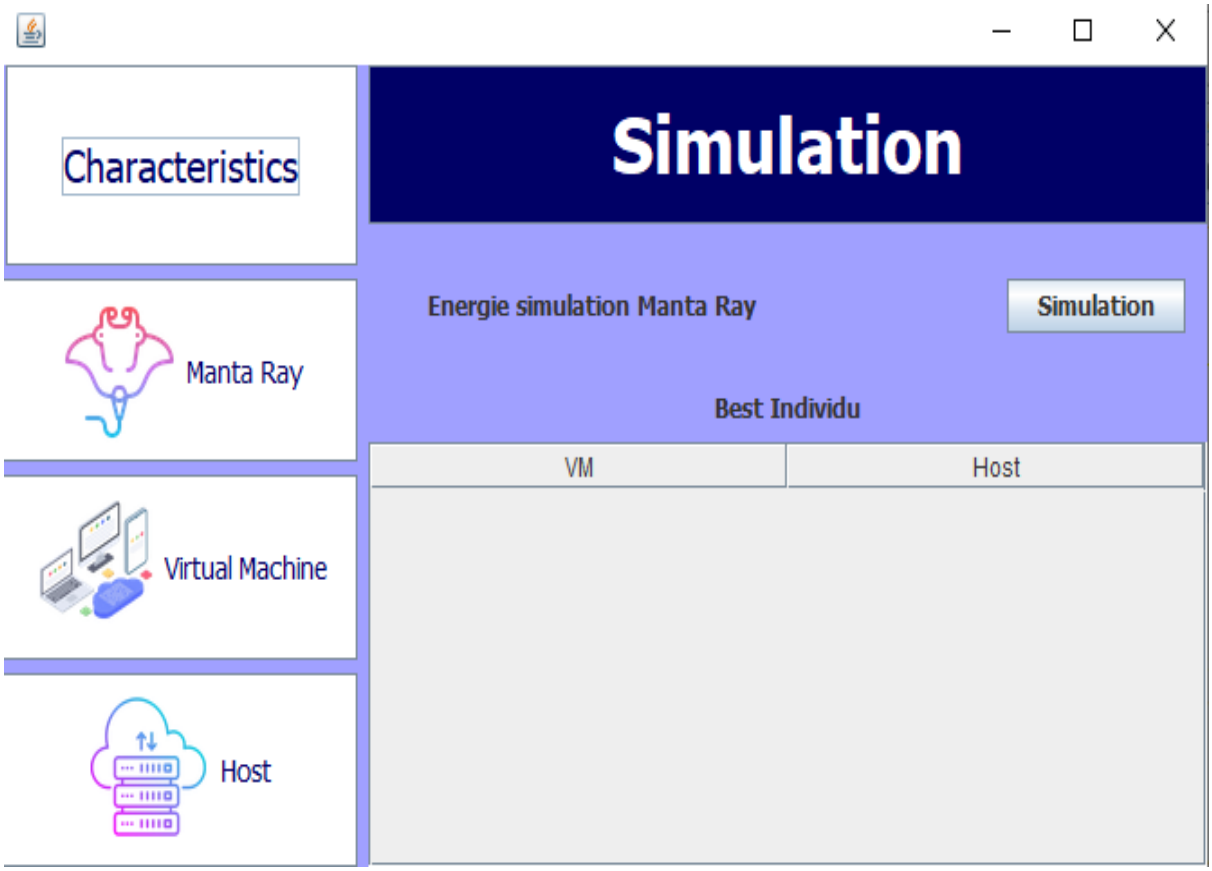



Figure 4.7. Interface d'affichage de résultat.

## 4. Résultats expérimentaux

Pour évaluer la technique proposée on a effectué des simulations sur une machine avec un processeur Intel core-5 de fréquence 1.8 GHZ et avec une RAM de 8 Go.

Dans la simulation, on a commencé par des petits nombres de VM et on a ensuite augmente ce nombre.

Les résultats de simulation de la technique Optimisation de l'alimentation des Raies Manta (MRFO) sont comparés avec deux méthodes reconnais dans la littérature à savoir le placement selon la politique Round Robin et le placement Random.

Cette simulation a été réalisée avec les paramètres suivants dans le tableau 1 et tableau 2

*Tableau 1 Les paramètres des VMs et Hôtes*

| | PES | MIPS | RAM | Size | BW |
|---|---|---|---|---|---|
| **VM** | 1 | 1000 | 512 | 1000 | 100 |
| **Hôte** | 2 | 10000 | 6400000 | 1000000 | 1000000 |

*Tableau 2 Les Nombres des VMs et Hôtes*

| **Nombre de VM** | 30 | 60 | 80 | 100 | 150 |
|---|---|---|---|---|---|
| **Nombre de Hôte** | 10 | 20 | 30 | 40 | 60 |

**Comparaison par rapport à l'énergie consommée**

Le tableau suivant montre les résultats de comparaison entre le MRFO, Random et Round Robin par rapport à l'énergie consommée.

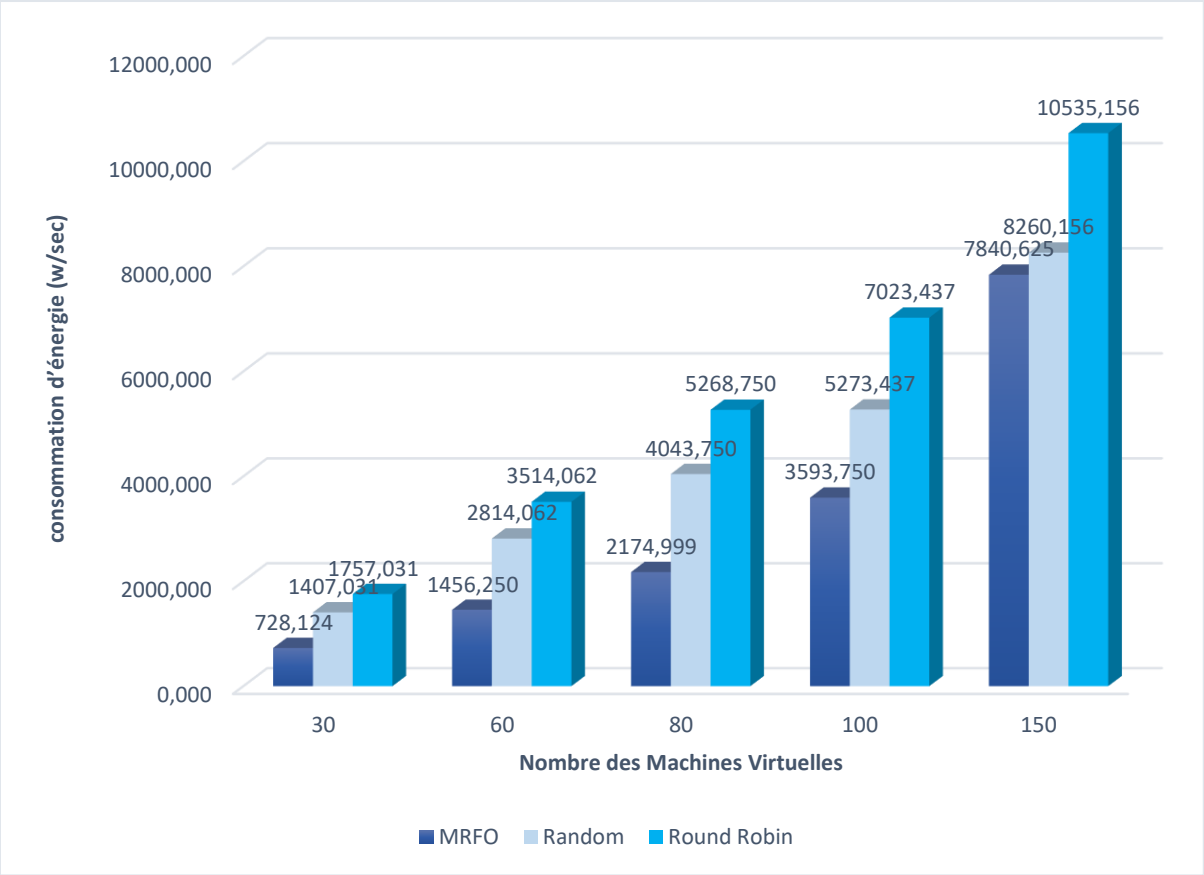



Figure 4.8. Comparaison entre le MRFO, Random et Round Robin en termes d'énergie.

Nous remarquons selon la figure 4.8 que le MRFO a donné de meilleur résultat par rapport au Random et Round Robin en termes d'énergie.

## 5. Conclusion

Le problème de la gestion des ressources et de la consommation d'énergie est devenu un sujet très important et d'actualité dans le Cloud Computing. Dans ce chapitre nous avons implémenté l'approche d'Optimisation de l'alimentation des Raies Manta (MRFO), quelle est notre solution proposée pour résoudre le problème de placement des VM, nous avons utilisé le simulateur CloudSim qui nous a permis d'intégrer notre algorithme, dans l'environnement de développement IDE NetBeans, et bien sur le langage que nous avons utilisé c'est Java. Nous avons présenté toutes les étapes de notre implémentation, et les résultats obtenus.

Les résultats de simulation montrent l'efficacité de l'algorithme proposé par rapport à d'autres approches notamment Random et Round Robin.

# Conclusion générale et perspectives

Le Cloud Computing est devenu au cours des dernières années un paradigme important dans le monde informatique. Son principe est de fournir des services décentralisés aux clients qui les utilisent et les payent selon le modèle de payement à l'utilisation. La demande croissante pour ce type de services oblige les fournisseurs de services Cloud à augmenter la taille de leurs infrastructures au point que la consommation d'énergie et les coûts associes deviennent très importants.

Le problème de la gestion des ressources et de la consommation d'énergie est devenu un sujet très important et d'actualité dans le Cloud Computing. Comme tout autre système distribué, les centres de données du Cloud consomment une énorme quantité d'énergie électrique. Cette énergie ne provoque pas seulement la diminution des bénéfices des fournisseurs, mais aussi l'émission d'une grande quantité de dioxyde de carbone. Afin d'améliorer l'utilisation des ressources et réduire cette consommation d'énergie, plusieurs techniques sont utilisées par les fournisseurs des services.

Dans ce travail, on a mis en place une solution pour l'optimisation de la consommation d'énergie et l'utilisation des ressources. Nous avons également utilisé, pour la première fois, une métaheuristique bio-inspirée appelée l'algorithme MRFO. Nous avons adopté une version discrète de cet algorithme vu la nature discrète du problème étudié. Les résultats de simulation on été assez satisfaisantes ce qui nous a permis de les comparer avec d'autres techniques utilisées largement dans le domaine d'optimisation dans le Cloud.

Comme étant l'objectif de la technique proposée dans ce travail était principalement la minimisation de l'énergie consommée dans le centre de données, nous visons à étendre l'algorithme afin de traiter d'autres objectifs simultanément. Il s'agit de l'optimisation multi-objectif qui présente un grand challenge notamment lors des objectifs contradictoires.

# Bibliographie

[1] SOKOL, Annie W., HOGAN, Michael D., et al. Nist cloud computing standards roadmap. 2013.

[2] R. Buyya, J. Broberg et M. Andrzej, “Cloud Computing: Principles and Paradigms”. Hoboken, New Jersey: John Wiley & Sons, 2011.

[3] C. Helier, B. Morlon, « Le Cloud Computing est-il une solution d’avenir pour l’entreprise », Mémoire de Recherche, ESGI, 2012

[4] ZHAO, Liang, SAKR, Sherif, LIU, Anna, et al. Cloud data management. Cham, Switzerland : Springer, 2014.

[5] FEMMAM, Manel. Une approche formelle pour la planification des tâches pour la QoS dans le cloud-computing. Diss. Université Mohamed Khider–BISKRA, 2018.

[6] https://www.tophebergeur.com/blog/grid-computing-vs-cloud-computing/; 2 septembre 2020, consulté le : 17/04/2022.

[7] https://vivavideoappz.com/all-that-you-need-to-know-about-utility-computing/; 6 decembre 2019, consulté le : 20/04/2022.

[8] https://docplayer.net/18944784-Cloud-computing-virtual-machines-provisioning-and-migration-services-mohamed-el-refaey.html, consulté le : 09/05/2022.

[9] HAMDAQA, Mohammad et TAHVILDARI, Ladan. Cloud computing uncovered: a research landscape. Advances in computers, 2012, vol. 86, p. 41-85.

[10] KARTIT, Zaid. Contribution à la sécurité du Cloud Computing: Application des algorithmes de chiffrement pour sécuriser les données dans le Cloud Storage. 2016.

[11] KUMAR, P. Ravi, RAJ, P. Herbert, et JELCIANA, P. Exploring data security issues and solutions in cloud computing. Procedia Computer Science, 2018, vol. 125, p. 691-697.

[12] VAQUERO, Luis M., RODERO-MERINO, Luis, CACERES, Juan, et al. A break in the clouds: towards a cloud definition. ACM sigcomm computer communication review, 2008, vol. 39, no 1, p. 50-55.

[13] https://www.ba-info.fr/post/iaas-paas-saas-comment-bien-choisir-son-mod%C3%A8le-cloud; 8 avril 2021, consulté le : 09/05/2022.

[14] Yann Collette, Patrick Siarry ; Optimisation Multiobjectif ; Septembre 2002 ; Edition Eyrolles, 75240 Paris Cedex 05, France.

[15] Mohammad Masdari, Sayyid Shahab Nabavi, Vafa Ahmadi, An Overview of Virtual Machine Placement Schemes in Cloud Computing, Journal of Network and Computer Applications, 15 January 2016.

[16] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli, "Hybrid metaheuristics in combinatorial optimization: A survey," Applied Soft Computing, vol. 11, pp. 4135-4151, 2011.

[17] Layeb, Abdesslem, Utilisation des Approches d'Optimisation Combinatoire pour la Vérification des Applications Temps Réel, thése de doctorat, université Mentouri de Constantine, 2010.

[18] BECHIR, AZIZA, Résolution des problèmes d'optimisation par les systèmes multi-agents et les approches évolutionnaires, thèse de doctorat, Université Mohamed Khider-Biskra, 2016.

[19] Alaoui Abdiya, Application des techniques des métaheuristiques pour l'optimisation de la tâche de la classification de la fouille de données. Thèse de doctorat en informatique, Université des sciences et de la technologie d'Oran Mohamed Boudiaf, 2012.

[20] S. M. Sait et H. Youssef; iterative computer algorithms with applications in ingineering: solving combinatorial optimization problems, IEEE computers society, 1999.

[21] Gherboudj, Amira. Méthodes de résolution de problèmes difficiles académiques. Thèse de doctorat en informatique. Université de Constantine 2. 2013.

[22] E. Bonomi et J.-L. Lutton ; le recuit simulé pour la science ; numéro 129, pages 68-77 ; Juillet 1988.

[23] M. O'Keeffe et M. Ô Cinnéide; A Stochastic Approach to automated Design Improvement; 2003.

[24] Le Recuit Simulé ; 17 Novembre 2004.

[25] F. Glover; Artificial Intelligence, heuristic frameworks and tabu search; Managerial and decision economics; volume 11, pages 365-375; 1990.

[26] D. De Werra ; heuristics for graph coloring ; computing Suppl ; volume 7, pages 191-208 ; 1990.

[27] PAPADIMITRIOU, Christos H. et STEIGLITZ, Kenneth. Combinatorial optimization: algorithms and complexity. Courier Corporation, 1998.

[28] Ilhem boussaïd, perfectionnement de métaheuristiques pour l'optimisation continue,Thèse de doctorat en informatique, Université des sciences et de la technologie houari Boumediene, 2013.

[29] AMÉDÉE, Souquet et FRANCOIS-GÉRARD, Radet. 'ALGORITHMES GENETIQUES'. TE de fin d'année, 2004.

[30] Kennedy, J. and Eberhart, R. C. Particle Swarm Optimization. In : Proceedings of the IEEE Int. Conf. on Neural Networks IV, pages 1942-1948, Perth, Australia, 1995.

[31] Kennedy, J., Eberhart, R.C., Shi, Y. Swarm Intelligence, Morgan Kaufmann Academic Press. 2001

[32] ZHAO, Weiguo, ZHANG, Zhenxing, et WANG, Liying. Manta ray foraging optimization: An effective bio-inspired optimizer for engineering applications. Engineering Applications of Artificial Intelligence, 2020, vol. 87, p. 103300.

[33] DUBEY, Kalka, SHARMA, Subhash Chander, et NASR, Aida A. A Simulated Annealing based Energy-Efficient VM Placement Policy in Cloud Computing. In : 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE). IEEE, 2020. p. 1-5.

[34] WU, Yongqiang, TANG, Maolin, et FRASER, Warren. A simulated annealing algorithm for energy efficient virtual machine placement. In : 2012 IEEE international conference on systems, man, and cybernetics (SMC). IEEE, 2012. p. 1245-1250.