الجمهورية الجزائرية الديمقراطية الشعبية
République algérienne démocratique et populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique
جامعة عين تموشنت بلحاج بوشعيب
Université –Ain Temouchent– Belhadj Bouchaib
Faculté des Sciences et de Technologie
Département Mathématique et Informatique

Projet de Fin d'Etudes
Pour l'obtention du diplôme de Master en Informatique
Domaine : Mathématiques et Informatique
Filière : Informatique
Spécialité : Cybersécurité et Intelligence artificielle.
Thème

Ordonnancement des tâches éco-énergétique dans les environnements Cloud et le Fog Computing.

Présenté par :

Boudieb Chaimaa

Tebbat Serina

Devant le jury composé de :

Dr Merad Boudia.D	UAT.B.B	Président
Dr MEDEDJEL.M	UAT.B.B	Examinateur
Dr BOUAFIA.Z	UAT.B.B	Encadrant

Année Universitaire : 2025–2026

Remerciement

Nous tenons tout d'abord à remercier **Allah**, le Sage et le Miséricordieux, pour nous avoir accordé la force, le courage et la patience nécessaires à la réalisation de ce modeste travail.

Nous tenons aussi à remercier notre encadrant **Monsieur BOUAFIA Zouheyr** pour ses précieux conseils, encouragements, sa disponibilité constante et la qualité scientifique de son accompagnement tout au long de cette étude. Ses orientations éclairées et sa rigueur scientifique ont été d'une valeur inestimable pour nous.

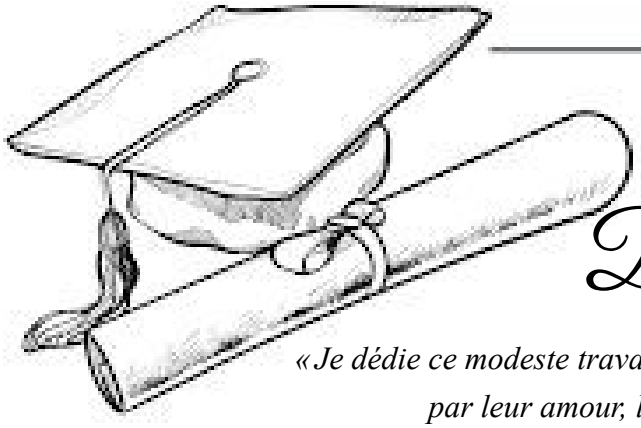
Nos remerciements chaleureux aux **membres du jury** pour l'honneur qu'ils nous ont fait en acceptant d'évaluer ce travail. Nous leur sommes reconnaissantes pour le temps qu'ils y consacrent, la rigueur de leur analyse, ainsi que pour leurs remarques constructives et enrichissantes.

Nos sincères remerciements vont également à l'ensemble des **enseignants du Département des Mathématiques et Informatique** de l'Université d'Ain Temouchent – Belhadj Bouchaib, pour leurs conseils précieux et leur accompagnement tout au long de notre formation.

Nos remerciements à nos **très chers parents** de nous avoir toujours donné le courage et d'être l'une des raisons qui nous pousse à nous battre face à chaque obstacle. Votre soutien indéfectible et votre amour inconditionnel ont été notre plus grande source de motivation.

Nous remercions enfin l'ensemble des **personnes** qui ont contribué d'une manière ou d'une autre à l'élaboration de ce travail, ainsi qu'à nos **collègues et amis de promotion** pour leur solidarité, leur entraide et tous les moments de partage vécus au cours de ces années de l'étude.

Merci



Dédicace

« Je dédie ce modeste travail à tous ceux qui ont illuminé mon chemin par leur amour, leur soutien et leur confiance... »

À l'âme de mon très cher père,

Que **Allah** t'enveloppe de Sa miséricorde et t'accueille en Son paradis. Tu m'as appris la persévérance et l'amour du savoir. Tu vivras à jamais dans mon cœur. *Allah yar7amek ya baba.*

À ma très chère maman,

Toi qui m'as donné la vie et la force de ne jamais abandonner. Aucune dédicace ne saurait traduire ma gratitude pour tes sacrifices et tes prières. Que **Allah** te préserve et te comble de bonheur.

À mon cher frère,

Tu as été mon soutien et ma force dans les moments de doute. Je te dédie ce travail en témoignage de mon affection sincère.

À ma chère sœur,

Tu es ma confidente et ma source d'inspiration au quotidien. Que **Allah** nous garde toujours unies et nous bénisse.

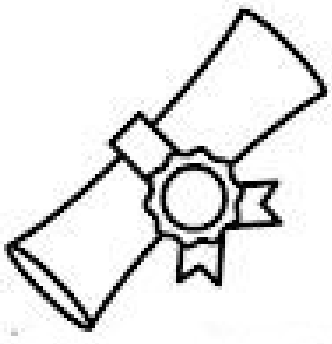
À tous mes enseignants,

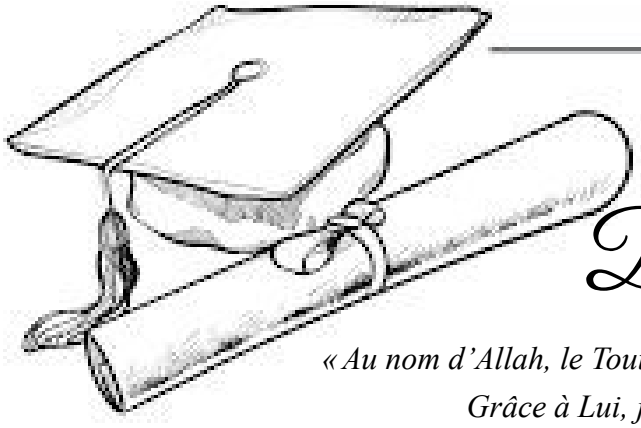
Vous avez semé en moi les graines du savoir avec patience et dévouement. Je vous exprime ma profonde et sincère reconnaissance.

À tous mes amis,

Pour chaque moment d'entraide qui a rendu ces années inoubliables. Je vous dédie ce travail avec toute mon affection.

Serina





Dédicace

« Au nom d'Allah, le Tout Miséricordieux, le Très Miséricordieux.
Grâce à Lui, j'ai pu accomplir ce travail. »

À moi-même,

Pour ma patience, ma persévérance et tous les efforts fournis malgré les difficultés rencontrées tout au long de ce parcours.

À mon cher père Boudiab Houari,

Qui m'a soutenue et encouragée malgré les circonstances difficiles. Merci pour tous les sacrifices, la force et le courage que tu m'as transmis.

À ma très chère mère Boulfadaoui Mama,

Pour son amour infini, ses prières, sa tendresse et son soutien inconditionnel. Que **Allah** la protège et la garde toujours à mes côtés.

À mes chères sœurs Khadidja et Wafaa,

Pour leur présence, leurs encouragements et leur affection.

À mes beaux-frères Amine et Imad,

Pour leur gentillesse, leur soutien et leur respect.

À mes adorables neveux Mehdi Aness, Abdelkrim et mohammed,

À qui je souhaite un avenir rempli de réussite, de bonheur et de succès.

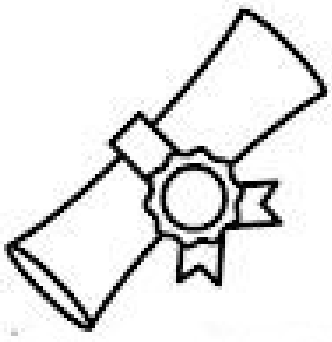
À ma tante Boudiab Malika,

Pour son soutien et sa bienveillance.

À toutes les personnes qui m'ont aidée et soutenue,

De près ou de loin, je dédie ce travail avec une profonde gratitude et un immense respect.

Chaimaa



RÉSUMÉ

Ce mémoire étudie l'ordonnancement des tâches *IoT* dans les environnements de *Cloud* et de *Fog Computing*, visant à minimiser la consommation énergétique et le temps d'exécution, tout en garantissant une qualité de service satisfaisante. Nous avons analysé et comparé diverses approches d'ordonnancement, incluant les méthodes classiques ainsi que les techniques intelligentes fondées sur l'apprentissage par renforcement. Une simulation développée avec SimPy et Python a permis d'évaluer les performances de ces algorithmes dans différents scénarios. Les résultats démontrent que les approches par apprentissage par renforcement améliorent l'efficacité globale du système en termes de consommation énergétique, de *makespan* et de respect des délais. L'approche *Double Deep Q-Network* (DDQN), en particulier, a montré une adaptabilité supérieure dans les environnements distribués et hétérogènes.

Mots-clés : *Cloud Computing*, *Fog Computing*, *IoT*, ordonnancement des tâches, *Q-Learning*, *Deep Q-Learning*, consommation énergétique, *makespan*.

ABSTRACT

This thesis investigates the scheduling of *IoT* tasks in *Cloud* and *Fog Computing* environments, with the objective of minimizing energy consumption and execution time while ensuring satisfactory Quality of Service (QoS). Various scheduling approaches were analyzed and compared, including traditional methods as well as intelligent techniques based on Reinforcement Learning. A simulation developed using SimPy and Python was used to evaluate the performance of these algorithms under different scenarios. The results demonstrate that Reinforcement Learning-based approaches improve the overall efficiency of the system in terms of energy consumption, *makespan*, and deadline compliance. In particular, the *Double Deep Q-Network (DDQN)* approach showed superior adaptability in distributed and heterogeneous environments.

Keywords : *Cloud Computing*, *Fog Computing*, *IoT*, Task Scheduling, *Q-Learning*, *Deep Q-Learning*, Energy Consumption, Makespan.

الملخص

تتناول هذه المذكرة دراسة جدولة مهام إنترنت الأشياء (IoT) في بيئات الحوسبة السحابية (Cloud Computing) والحوسبة الضبابية (Fog Computing)، بهدف تقليل استهلاك الطاقة وزمن التنفيذ، مع ضمان مستوى مُرضٍ من جودة الخدمة (QoS). وقد تم تحليل ومقارنة مجموعة من أساليب الجدولة المختلفة، بما في ذلك الطرق التقليدية والتقنيات الذكية المعتمدة على التعلم المعزز (Reinforcement Learning).

تم تطوير بيئة محاكاة باستخدام لغتي Python وSimPy لتقييم أداء هذه الخوارزميات في سيناريوهات متعددة. وأظهرت النتائج أن الأساليب القائمة على التعلم المعزز تُحسن الكفاءة العامة للنظام من حيث استهلاك الطاقة، ومدة التنفيذ الكلية (Makespan)، واحترام الآجال الزمنية المحددة للمهام.

وعلى وجه الخصوص، أظهرت خوارزمية الشبكة العصبية المزدوجة العميقة للتعلم المعزز (Double Deep Q-Network - DDQN) قدرة أكبر على التكيف مع البيئات الموزعة وغير المتجانسة، مما يجعلها حلاً واعدًا لتحسين جدولة المهام في أنظمة الحوسبة الحديثة.

الكلمات المفتاحية: الحوسبة السحابية، الحوسبة الضبابية، إنترنت الأشياء، جدولة المهام، Q-Learning، Deep Q-Learning، DDQN، استهلاك الطاقة، مدة التنفيذ الكلية (Makespan).

Table des matières

Liste des abréviations

API : Application Programming Interface

BCO : Bee Colony Optimization

BiLSTM : Bidirectional Long Short-Term Memory

BoT : Bag of Tasks

BPaaS : Business Process as a Service

CapEx : Capital Expenditure

CNN : Convolutional Neural Network

CPU : Central Processing Unit

CSV : Comma-Separated Values

DAG : Directed Acyclic Graph

DBO : Dung Beetle Optimization

DDQN : Double Deep Q-Network

DMA : Deadline Monotonic Analysis

DQN : Deep Q-Network

EDF : Earliest Deadline First

FCFS : First Come First Served

FIFO : First In First Out

GA : Genetic Algorithm

GPU : Graphics Processing Unit

HaaS : Hardware as a Service

HPC : High Performance Computing

IaaS : Infrastructure as a Service

IDE : Integrated Development Environment

IIoT : Industrial Internet of Things

IoT : Internet of Things

LLF : Least Laxity First

MAS : Multi-Agent System

MDP : Markov Decision Process

MI : Million d'Instructions

MIPS : Million d'Instructions Par Seconde

MINLP : Mixed Integer Nonlinear Programming

ML : Machine Learning

NIST : National Institute of Standards and Technology

NP : Non-deterministic Polynomial

PaaS : Platform as a Service

PM : Physical Machine

QoS : Quality of Service

RAM : Random Access Memory

RL : Reinforcement Learning

RMA : Rate Monotonic Analysis

RR : Round Robin

SA : Simulated Annealing (Recuit Simulé)

SaaS : Software as a Service

SJF : Shortest Job First

SLA : Service Level Agreement

SOA : Service Oriented Architecture

SRTF : Shortest Remaining Time First

VM : Virtual Machine

XaaS : X as a Service

Liste des algorithmes

Table des figures

Liste des tableaux

Introduction générale

Introduction générale

L'Internet des objets (IoT) est un secteur en pleine expansion, caractérisé par une évolution rapide et continue. Cette technologie innovante, qui permet une communication intelligente entre les objets physiques et leur environnement, a profondément transformé les services numériques dans plusieurs domaines tels que l'industrie, la santé, les transports et les villes intelligentes [89]. En connectant des milliards d'appareils, l'IoT joue un rôle clé dans l'optimisation de l'efficacité des opérations industrielles et facilite le quotidien des utilisateurs [89].

Par ailleurs, l'augmentation significative du nombre d'appareils connectés génère un volume important de données et de calcul, entraînant une surcharge des infrastructures cloud, notamment en ce qui concerne le stockage, le traitement et la bande passante réseau [89]. Pour y remédier, de nouveaux paradigmes de calcul distribué, tels que le *Fog Computing* et l'*Edge Computing*, ont émergé. Ces derniers ont pour objectif de rapprocher les ressources de calcul des utilisateurs et des objets connectés [90, 91]. Ces technologies innovantes offrent une gestion optimisée des données en temps réel, réduisant significativement la latence, améliorant la réactivité du système et maximisant l'efficacité énergétique [90, 91].

Dans ce contexte, ce mémoire se concentre sur le défi de l'ordonnancement des tâches IoT dans un environnement *fog* et *cloud computing*, où les ressources sont diverses et les contraintes de performance multiples. Notre travail est structuré autour de trois chapitres, précédés d'une introduction et suivis d'une conclusion générale.

Le premier chapitre :Ce chapitre clarifie les concepts fondamentaux du *Cloud*, du *Fog* et de l'*Edge Computing*. Chaque environnement est ensuite présenté,incluant sa définition, ses caractéristiques, son architecture, ainsi que ses avantages et ses limites.

Le deuxième chapitre : Ce chapitre présente les principaux algorithmes d'ordonnancement des tâches : approches classiques, heuristiques, méta-heuristiques et méthodes basées sur l'intelligence artificielle. L'étude permet de dégager leurs principes de fonctionnement. Ensuite, une comparaison de travaux récents met en évidence leurs objectifs, leurs métriques et leurs principales limites.

Le troisième chapitre porte sur l'ordonnancement des tâches IoT dans les environnements *fog* et *cloud computing*. Nous avons justifié le choix de SimPy comme outil de simulation principal à l'issue d'une comparaison, puis détaillé l'architecture du système et les algorithmes implémentés, en particulier les approches traditionnelles et celles fondées sur le *Q-Learning* et le *DDQN*. Nous avons également défini les paramètres de simulation et les

métriques d'évaluation. Enfin, une étude comparative selon plusieurs scénarios a évalué l'efficacité des approches en termes de consommation d'énergie, de *makespan* et de respect des délais.

CHAPITRE 1

Notions et Concepts des Environnements Cloud, Fog et Edge Computing

I.1 Introduction

Dans le contexte de l'essor fulgurant des technologies numériques et de la croissance exponentielle des données générées par les objets connectés et les applications intelligentes, ce chapitre présente les bases théoriques des environnements Cloud, fog et edge computing. Il expose leurs architectures, leurs principes de fonctionnement ainsi que leurs avantages et leurs limites. Enfin, une analyse comparative entre ces différents environnements.

I.2. Cloud Computing

I.2.1. Définition :

Bien qu'il n'existe pas de définition unique du Cloud Computing, la recherche scientifique a permis d'en tracer les contours. En analysant plus d'une vingtaine de sources,Vaquero et ses collègues [1] présentent le Cloud comme un réservoir de ressources virtuelles (matériels et logiciels) flexibles, accessibles facilement et facturées à l'usage.
De son côté,George Reese[2] propose une approche plus concrète pour différencier le Cloud d'un simple service internet. Pour lui, le Cloud se définit par trois critères clés :

- Une accessibilité via une interface web ou une API.
- L'absence d'investissement matériel lourd au départ.
- Un paiement basé uniquement sur la consommation réelle.

Finalement, il convient de retenir que la définition du National Institute of Standards and Technology (NIST) [3] reste aujourd'hui la référence la plus acceptée. Selon cette définition, le Cloud est un modèle permettant d'accéder, n'importe où et à la demande, à un ensemble de ressources informatiques partagées (réseaux, serveurs, stockage). Ce qui le distingue des anciens systèmes de partage de temps (Time-sharing), c'est sa capacité à combiner des services variés, la vitesse des réseaux actuels et l'innovation constante des fournisseurs majeurs du secteur [6].

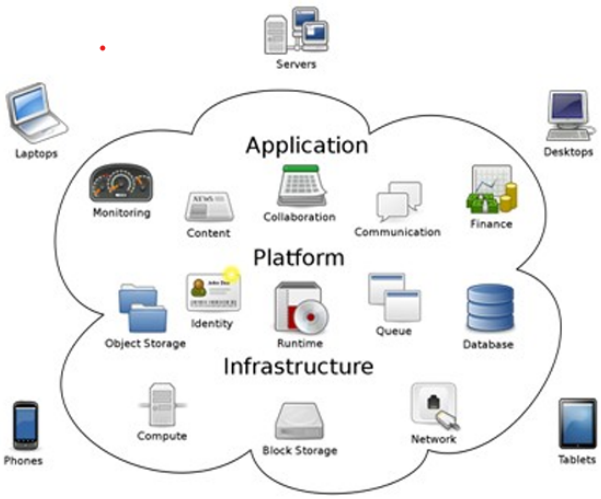


Figure I.1 : Cloud Computing [4].

I.2.2. Caractéristiques de cloud computing :

Selon le National Institute of Standards and Technology (NIST) [3], le cloud computing est un modèle qui repose sur cinq caractéristiques essentielles, qui le distinguent de tout autre système informatique traditionnel.

- **Le Libre-service à la demande (On-demand Self-service) :** Cette caractéristique confère à l'utilisateur une autonomie complète dans l'accès aux ressources informatiques. En effet, il peut obtenir de la puissance de calcul ou du stockage automatiquement, sans intervention d'un technicien ou d'un administrateur. Ainsi, si un utilisateur a besoin d'un serveur à 2h du matin, il peut l'obtenir instantanément, sans attendre l'approbation de qui que ce soit [3].

- **L'Accès universel au réseau (Broad Network Access) :** Le cloud se caractérise par son accessibilité universelle. Les services sont disponibles via Internet depuis n'importe quel appareil connecté qu'il s'agisse d'un smartphone, d'une tablette ou d'un ordinateur portable. Cette disponibilité permanente supprime toute barrière matérielle et offre à l'utilisateur une liberté totale de connexion, quel que soit l'endroit où il se trouve [3][6].

- **La Mise en commun des ressources (Resource Pooling) :** Cette caractéristique repose sur le principe de mutualisation des ressources. Le fournisseur de services cloud

dispose d'importants centres de données dont les ressources sont regroupées afin de servir plusieurs clients simultanément. Ces ressources sont allouées et réajustées dynamiquement selon les besoins de chaque utilisateur, sans que ce dernier sache sur quel serveur physique ses données sont réellement hébergées [1][3].

- **L'élasticité rapide (Rapid Elasticity) :** Il s'agit de la capacité du cloud à s'adapter dynamiquement aux besoins en ressources. Lorsqu'une application connaît une hausse soudaine de la demande, le système alloue instantanément des ressources supplémentaires pour absorber le trafic. À l'inverse, dès que la demande diminue, les ressources sont automatiquement réduites. De ce fait, les ressources disponibles semblent illimitées et s'ajustent en temps quasi réel, ce qui représente un avantage considérable pour les entreprises en pleine croissance [1][6].

- **Le service mesuré (Measured Service) :** Selon le principe du pay-per-use, l'utilisation des ressources est surveillée, mesurée et facturée avec précision à l'aide de compteurs dédiés. Cette approche garantit une transparence totale des coûts pour l'utilisateur, qui ne paie que ce qu'il consomme réellement, à l'image d'une facture d'électricité [3][6].

I.2.3. Architecture de cloud computing :

Le cloud computing utilise des technologies telles que la virtualisation, l'architecture orientée services et les services web. Le cloud est souvent confondu avec plusieurs paradigmes informatiques, tels que le Grid computing, l'Utility computing et l'Autonomic computing, chacun partage certains aspects avec le cloud computing. La figure I.2 montre la convergence des technologies qui ont contribué à l'avènement du cloud computing.

- **Le grid computing :** peut se définir comme un paradigme de calcul réparti. Ce paradigme coordonne des ressources autonomes et géographiquement distribuées afin d'atteindre un objectif de calcul commun. Le grid est un système informatique qui repose sur le partage dynamique des ressources entre divers acteurs, qu'il s'agisse de participants, d'organisations ou d'entreprises. Cette approche vise à optimiser la mutualisation des ressources pour l'exécution d'applications scientifiques exigeantes en calcul, telles que des simulations numériques intensives ou le traitement de grands

volumes de données. Le cloud computing et le grid computing présentent des similitudes, notamment en ce qui concerne l'approche conceptuelle qui consiste à fournir des ressources sous forme de services. Cependant, ces deux systèmes présentent des différences en termes de finalité. En effet, la technologie des grilles a pour vocation première de permettre à des groupes distincts de partager l'accès en libre-service à leurs ressources. En revanche, le cloud a pour objectif de fournir aux utilisateurs des services « à la demande », selon le principe du « paiement à l'usage ». En outre, le cloud s'appuie sur les technologies de virtualisation à plusieurs niveaux, à savoir matériels et plateformes d'application, afin de permettre le partage et l'approvisionnement dynamique des ressources.

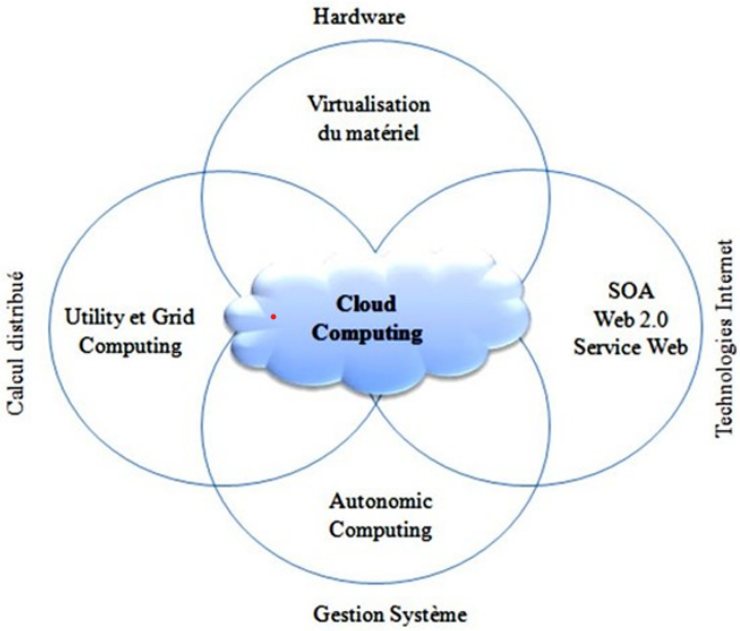


Figure I.2 : Convergence des différentes avancées menant à l'avènement du cloud computing [30].

- **Utility computing :(informatique utilitaire)** :L'utility computing est tout simplement un transfert hors site d'un système de calcul ou de stockage. Il offre un modèle d'affectation des ressources à la demande et facture les utilisateurs en fonction de leur consommation. Le cloud computing peut être vu comme une mise en œuvre de l'informatique utilitaire. Il utilise un système de tarification basé sur l'utilité, exclusivement pour des motifs économiques. Grâce à l'approvisionnement des ressources à la demande et au paiement à l'usage, les fournisseurs de services peuvent optimiser l'utilisation des ressources et réduire leurs coûts d'exploitation [32].

- **L'autonomie computing :** vise à développer des systèmes informatiques capables de s'auto administrer en s'adaptant aux changements internes ou externes sans intervention humaine. L'objectif de l'informatique autonome est de faire face à la complexité de la gestion des systèmes informatiques actuels. Le cloud computing a bien des caractéristiques autonomes, comme l'approvisionnement automatique des ressources, mais il a pour but de diminuer le coût des ressources et non de réduire la complexité du système [32].

- **La virtualisation :** est une technologie qui, par essence, fait abstraction des détails du matériel physique. Elle fournit des ressources virtuelles pour les applications de haut niveau. En effet, la virtualisation est un concept qui englobe l'ensemble des méthodes matérielles et logicielles permettant la mise en fonctionnement, sur une seule machine physique, de plusieurs configurations informatiques (systèmes d'exploitation, etc.). Ces dernières forment alors plusieurs machines virtuelles, qui reproduisent le comportement des machines physiques. La virtualisation, qui consiste à utiliser des représentations numériques de ressources informatiques pour les partager et les exploiter de manière dématérialisée, constitue le fondement du cloud computing. En effet, elle permet la mise en commun de ressources informatiques au sein de clusters de serveurs, permettant ainsi l'allocation dynamique de machines virtuelles aux applications à la demande. La Figure I.3 illustre l'architecture en couches de la technologie de virtualisation, mettant en évidence la composition stratifiée de ses composants et leur interaction dynamique. Le moniteur de machine virtuelle, ou hyperviseur, est un dispositif qui permet la virtualisation des ressources physiques des serveurs. Il s'agit d'un système qui divise la ressource physique du serveur physique en plusieurs machines virtuelles, chacune fonctionnant sous un système d'exploitation et une pile de logiciels distincts [32].

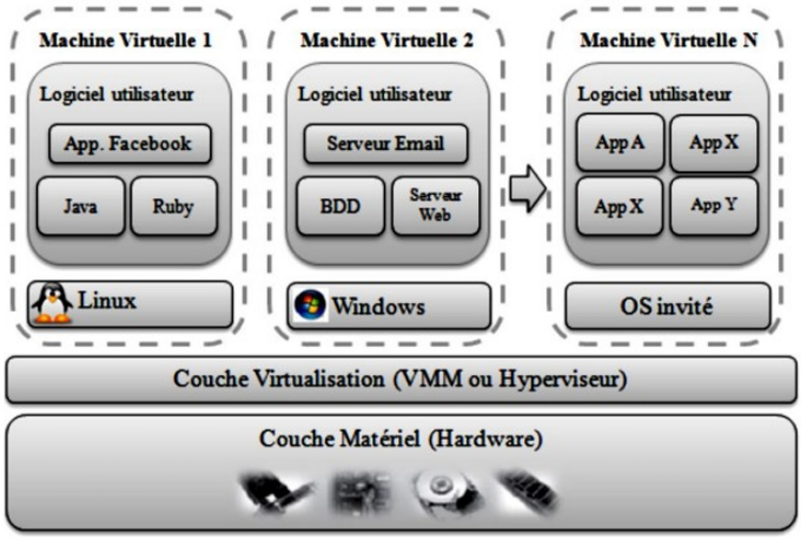


Figure I.3 : Une architecture de cloud computing en couches [32].

- **Acteurs du cloud computing :**Dans l'architecture de référence du cloud computing décrite par le NIST [31], on identifie cinq acteurs principaux comme le montre la Figure I.4.

  – **Consommateur (cloud Consumer) :** personne ou organisation qui se sert d'un service fourni par un fournisseur.

  – **Fournisseur (cloud Provider) :** personne, organisation ou entité qui met à la disposition des parties intéressées un service.

  – **Courtier (cloud broker) :** entité qui gère l'usage, la performance et l'approvisionnement de services, et qui négocie les relations entre les fournisseurs et les consommateurs.

  – **Auditor (cloud auditor) :** une entité susceptible d'effectuer une évaluation indépendante des services de cloud, comme les performances et la sécurité du cloud.

  – **Porteur (cloud carrier) :** un intermédiaire qui assure la connectivité et le transport des services des fournisseurs vers les consommateurs.

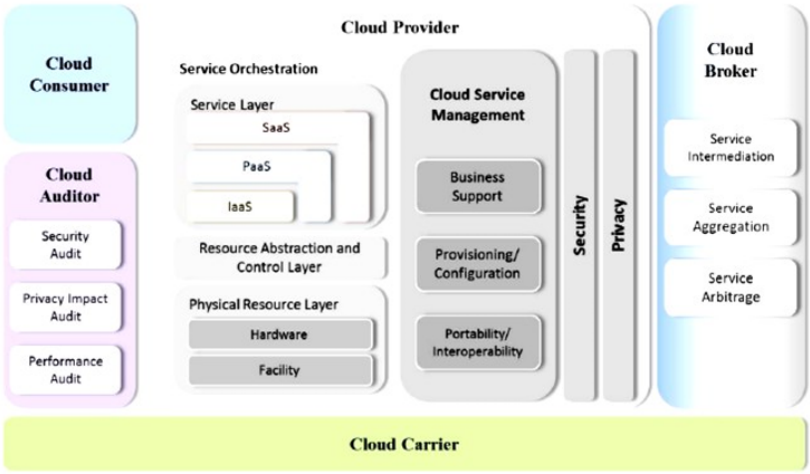


Figure I.4 : Modèle de référence conceptuel du cloud NIST 2011.

I.2.4. Modèles de déploiement :

On peut distinguer quatre types principaux de modèles de déploiement pour le cloud computing : le cloud privé, le cloud public et le cloud hybride, le cloud communautaire . La Figure I.5 présenter Modèles de déploiement de cloud computing :

- **Le cloud public :** est le modèle de déploiement le plus répandu. Les ressources appartiennent à un tiers, tel qu'Amazon Web Services, Microsoft Azure ou Google Cloud Plateforme, et sont accessibles via Internet, ce qui permet de réaliser des économies d'échelle [6].

- **Le cloud privé :** ne peut être utilisé que par une seule organisation, ce qui lui permet de contrôler pleinement et de renforcer la sécurité de ses ressources informatiques. L'organisation peut également choisir de gérer elle-même le cloud privé ou de faire appel à une entreprise externe pour fournir ces services[3].

- L'architecture de **cloud hybride** combine un cloud privé et un cloud public. Ces deux environnements sont interconnectés grâce à une technologie commune permettant le transfert de données. Les données sensibles sont stockées dans le cloud privé, tandis que le cloud public est dédié aux charges de travail importantes nécessitant une puissance de traitement élevée [5].

- **Community Cloud :** fournit une infrastructure partagée entre plusieurs organisations,

chacune ayant des intérêts ou des contraintes similaires, notamment en matière de conformité et de réglementation. Au lieu de construire leurs propres infrastructures, elles mettent en commun leurs ressources dans un environnement géré, contrôlé soit par un tiers, soit collectivement [3].

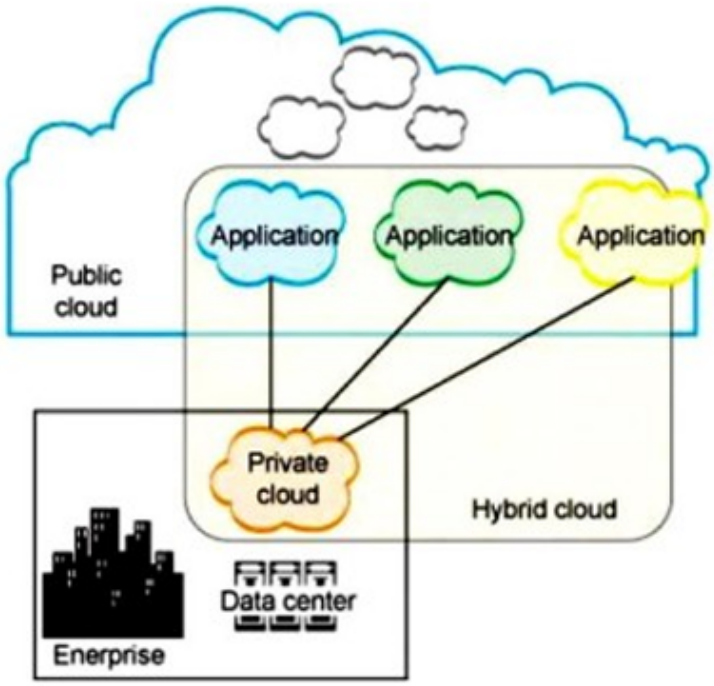


Figure I.5 : Modèles de déploiement de cloud computing [33].

I.2.5. Modèles du cloud computing :

Cette section présente les modèles de cloud. Nous allons d'abord détailler les modèles de service.

I.2.5.1 Modèles de services de cloud :

Le modèle XaaS (X as a Service) représente le fondement du paradigme du cloud computing, où la variable X désigne un service tel qu'un logiciel, une plateforme, une infrastructure, un Business Process, etc. La figure I.6 montre le modèle classique et les différents modèles de service de cloud.

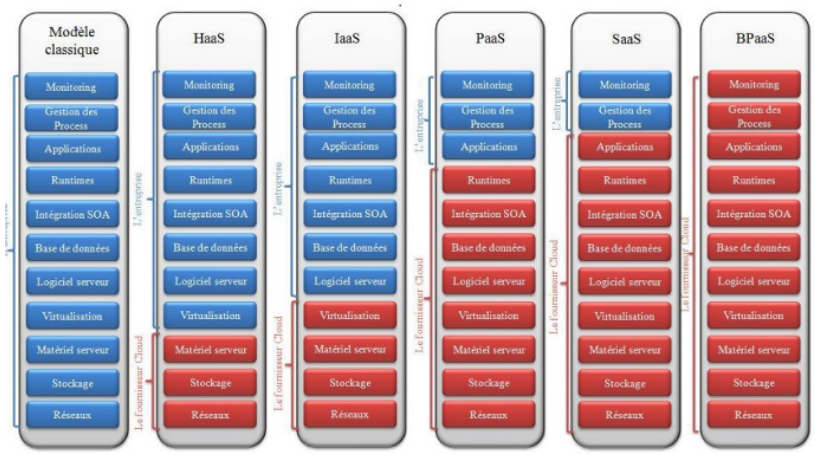


Figure I.6 : Les services XaaS du cloud computing [30].

Dans ce contexte, il est courant de distinguer quatre modèles de services principaux :

1. **Software as a Service (SaaS) :** C'est le modèle le plus connu du grand public. Il s'agit d'applications accessibles à distance, où l'utilisateur n'a rien à installer ni à gérer. Le fournisseur s'occupe de tout : maintenance, mises à jour et sécurité. Parmi les exemples courants, on peut citer Google Workspace ou Salesforce CRM. Ces services sont généralement accessibles via un abonnement mensuel [6].

2. **Platform as a Service (PaaS) :** Ce modèle est principalement destiné aux développeurs.Plutôt que de fournir une application finie, il offre une plateforme de travail complète incluant les systèmes d'exploitation, les bases de données et les langages de programmation. Le développeur peut ainsi se concentrer sur l'écriture de son code, car le service gère tout le cycle de vie technique du logiciel en arrière-plan [8].

3. **Infrastructure as a Service (IaaS) :** C'est le niveau le plus flexible. L'utilisateur loue des ressources informatiques de base à la demande (puissance de calcul, mémoire, stockage) sous forme de machines virtuelles. C'est le modèle où l'utilisateur a le plus de contrôle : il est responsable de l'installation du système d'exploitation et de la gestion de ses données, tandis que le fournisseur assure le bon fonctionnement du matériel physique et de la couche de virtualisation [6].

4. **Hardware as a Service (HaaS) :**repose sur la mise à disposition de ressources matérielles à distance, accessibles via Internet et configurables selon les besoins des utilisateurs. Il permet notamment l'exécution d'applications intensives en calcul et en trai-

tement de données, autrefois limitées aux systèmes HPC, désormais réalisables dans des environnements cloud [34][35]. Cette approche offre une grande flexibilité tout en laissant aux utilisateurs le contrôle sur les systèmes et les configurations. Parmi les principaux fournisseurs de ce type de services, on retrouve Baremetalcloud [36], Softlayer [38] et Grid5000 [37].

5. **Business Process as a Service (BpaaS) :** Le BPaaS s'adresse aux processus métiers, à la différence du SaaS qui vise les applications logicielles, et les ressources sont partagées entre les différents utilisateurs. Il s'intègre au-dessus des applis. Ce service est important pour les entreprises puisqu'il permet l'automatisation et l'orchestration des flux d'actions et la supervision de ces flux. En tant que service de cloud, le modèle BPaaS est accessible par le biais des technologies Internet. Des exemples de BPaaS sont : la gestion des voyages, les paiements en ligne, la gestion financière, etc. Les fournisseurs de ce service sont IBM, Atos, Wipro, entre autres [34].

I.2.6. Avantages du cloud computing :

L'adoption du cloud computing représente une évolution majeure dans la gestion des ressources numériques. Selon les travaux de Gupta et al. [11] et d'Alzahrani [12], cette technologie repose sur plusieurs piliers fondamentaux :

- **Accessibilité universelle :** L'utilisateur n'est plus limité à un poste de travail physique ; il peut accéder à ses services et données à tout moment, depuis n'importe quel terminal disposant d'une connexion internet [11].
- **Fiabilité et continuité de service :** Grâce à une architecture modulaire et des mécanismes d'équilibrage de charge (load balancing), les fournisseurs garantissent une disponibilité quasi permanente. Des sauvegardes régulières assurent la sécurité des données face aux incidents matériels [12].
- **Élasticité des ressources :** Le système s'adapte instantanément aux besoins réels, qu'il s'agisse de passer d'un stockage de quelques Go à plusieurs To. Cette flexibilité permet aux organisations de maîtriser leur budget sans saturer leurs serveurs [12].
- **Optimisation économique :** Le Cloud transforme les coûts fixes en coûts variables. En éliminant le besoin d'installations locales lourdes, l'utilisateur se décharge de la

maintenance technique, permettant ainsi de concentrer les ressources sur le cœur de métier [11] [12].

I.2.7. Limites du Cloud Computing :

Malgré ses nombreux avantages, le Cloud Computing présente des limites importantes, notamment face à l'émergence massive de l'Internet des Objets (IoT). Sa structure centralisée crée des goulots d'étranglement que l'on peut résumer en quatre points [13] :

- **La latence :** L'envoi de données vers des serveurs distants de milliers de kilomètres engendre des délais incompatibles avec les applications critiques comme les véhicules autonomes ou la robotique industrielle [13].
- **La saturation du réseau (Bande passante) :** Des milliards d'objets connectés transmettant simultanément leurs données vers le Cloud peuvent saturer les réseaux, rendant le système inefficace et coûteux en ressources [15].
- **La dépendance au réseau :** Le Cloud ne fonctionne qu'en présence d'une connexion internet active. Une interruption du réseau entraîne une indisponibilité totale du service, ce qui est inacceptable pour des environnements critiques comme les hôpitaux ou les usines [13].
- **La sécurité et la vie privée :** La centralisation massive de données sensibles crée une cible de choix pour les cyberattaques. De nombreuses organisations hésitent à confier leurs données à des serveurs dont elles ne contrôlent pas totalement la localisation physique [13].

I.3. Fog Computing

I.3.1. Définition :

Le Fog Computing se définit comme un nouveau paradigme architectural qui vient combler le fossé entre les objets connectés et les centres de données distants. Contrairement aux modèles centralisés, le Fog agit comme une couche intelligente, à la fois physique et virtuelle, située au plus près des utilisateurs. En proposant des ressources de calcul, de stockage

et de connectivité distribuées, le Fog ne remplace pas le Cloud, mais le prolonge là où la latence n'est plus tolérable [15].

I.3.2. Caractéristiques de Fog computing :

Le Fog Computing apporte des avantages uniques mis en évidence par Yi et al.[16] et Ai et al. [17]. On distingue cinq caractéristiques principales :

- **La faible latence (Low Latency) :** Comme le traitement se fait à proximité de l'utilisateur, le temps de réponse est extrêmement réduit. C'est crucial pour les applications en temps réel telles que les véhicules autonomes ou la chirurgie à distance [16].
- **La distribution géographique :** Contrairement aux centres de données centralisés, les ressources du Fog sont déployées au plus près du terrain — dans les rues, les usines ou les hôpitaux — permettant une couverture large et réactive [15].
- **La mobilité :** Le Fog est conçu pour gérer des appareils en mouvement constant (smartphones, drones, véhicules connectés), en assurant la continuité du service lors des déplacements [16].
- **L'hétérogénéité :** Il peut intégrer des appareils de natures très différentes (routeurs, passerelles, serveurs locaux) et les faire coopérer au sein d'un même écosystème [17].
- **L'interopérabilité avec le Cloud :** Le Fog travaille en complémentarité avec le Cloud : il traite les données urgentes localement et transfère les données moins critiques vers le Cloud pour un stockage et une analyse à long terme [15][17].

I.3.3. Architecture du Fog Computing :

L'architecture du Fog Computing s'organise en trois couches distinctes, formant un pont entre le monde physique et les centres de données distants [18], comme le montre la Figure I.7[18] :

- **La couche des objets connectés (IoT Layer) :** Cette couche de base regroupe tous les terminaux physiques : capteurs industriels, smartphones, montres intelligentes. Ces

appareils, souvent appelés « nœuds terminaux », collectent les données brutes sur le terrain [15].

- **La couche Fog (Fog Layer) :** C'est la couche intermédiaire, composée d'équipements réseau intelligents (routeurs, passerelles, points d'accès). Ces équipements ne se contentent plus de transmettre l'information : ils la traitent et la stockent localement, réduisant ainsi les allers-retours vers le Cloud [17][18].
- **La couche Cloud (Cloud Layer) :** Au sommet de la hiérarchie, le Cloud assure les traitements lourds et le stockage massif à long terme que les niveaux inférieurs ne peuvent pas gérer. Il constitue le niveau de la puissance de calcul quasi infinie [18].

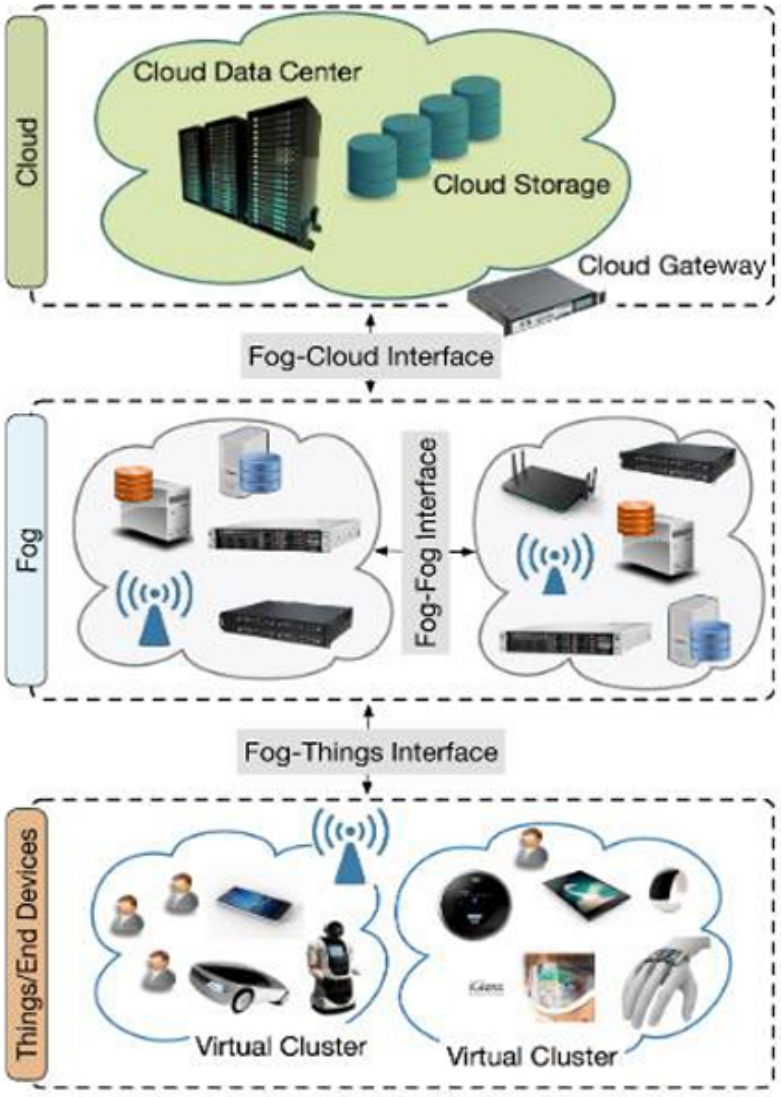


Figure I.7 : Architecture du Fog computing [18].

I.3.4. Avantages du Fog Computing :

Le Fog Computing représente la réponse logique aux limites du Cloud, en particulier pour les applications IoT critiques [19] :

- **Faible latence :** Le Fog traite l'information sur place, évitant les délais liés aux échanges avec des serveurs distants. Dans des secteurs comme la santé connectée ou les transports intelligents, chaque milliseconde est cruciale [19].
- **Distribution géographique :** Sa proximité physique avec les appareils lui permet de gérer des millions d'objets connectés de manière fluide, sans saturer les réseaux principaux [19].
- **Mobilité :** Le Fog maintient une connexion stable pour les terminaux en déplacement (drones, véhicules, smartphones) en passant le relais d'un nœud à l'autre sans interruption de service [16].
- **Hétérogénéité :** Il joue le rôle de médiateur universel, permettant à des appareils de technologies différentes de coopérer efficacement au sein d'un écosystème IoT cohérent [17][20].

I.3.5. Limites du Fog Computing :

Malgré ses nombreux avantages, le Fog Computing présente certaines limites qui peuvent affecter son déploiement et ses performances.

- **Complexité :** La diversité des dispositifs IoT et des configurations matérielles et logicielles rend la gestion et le déploiement du Fog complexes [15].
- **Sécurité :** La distribution géographique et le grand nombre de nœuds augmentent la surface d'attaque, rendant la sécurisation plus difficile [21].
- **Gestion des ressources :** Les nœuds Fog disposent de ressources limitées (CPU, mémoire), nécessitant une allocation efficace pour garantir les performances [39].

I.4. Edge Computing

I.4.1. Définition :

Le Edge Computing (ou informatique en périphérie) marque une rupture avec la centralisation du Cloud. L'idée fondamentale est de ne plus déplacer la donnée vers la puissance de calcul, mais d'amener la puissance de calcul là où la donnée naît.

Cette approche peut être comprise à travers trois dimensions clés définies par la littérature scientifique :

La mutation des objets connectés : Comme l'expliquent Shi et al. [21], l'appareil terminal n'est plus un simple capteur passif. Il devient un acteur hybride qui produit et traite l'information simultanément. C'est une vision centrée sur l'utilisateur où le réseau s'adapte à la source [22].

L'immédiateté du service : Le principal atout du Edge est la suppression du "voyage" de la donnée. En traitant les informations localement, Tseng [23] et Linthicum [24] soulignent que l'on atteint des performances en temps réel impossibles avec un Cloud lointain. C'est une réponse directe aux besoins d'agilité de l'industrie moderne.

Une architecture de micro-services : Pour Satyanarayanan [25], le pionnier du domaine, cela se traduit par l'utilisation de Cloudlets (petits centres de données de proximité). Ce modèle ne remplace pas le Cloud, mais collabore avec lui : le Edge gère l'urgence et la confidentialité, tandis que le Cloud s'occupe de l'archivage massif et des analyses à long terme [26].

En résumé : Le Edge Computing est un écosystème distribué qui optimise la bande passante et renforce la sécurité en traitant l'information au plus près de la réalité physique [20].

I.4.2. Caractéristiques du Edge Computing :

En s'appuyant sur les travaux de Singh [28], on peut identifier cinq caractéristiques majeures du Edge Computing :

- **Réactivité instantanée (Vitesse et latence) :** En analysant les données là où elles naissent, on supprime le temps de trajet vers le Cloud. Cette immédiateté rend l'information exploitable en temps réel, ce qui se traduit par des services plus performants [28].

- **Réduction des coûts :** Traiter et gérer les données localement coûte souvent moins cher que de recourir à la puissance de calcul des centres de données distants, notamment pour les volumes importants générés par l'Internet des Objets (IoT) [28].
- **Efficacité énergétique :** En évitant les échanges incessants avec des serveurs distants, les objets connectés consomment moins d'énergie, ce qui prolonge leur durée de vie et améliore leurs performances globales [28].
- **Analyse en temps réel :** L'analyse se fait pendant que l'action se déroule et non après coup, car tout se passe au niveau de l'appareil local. Cela est particulièrement utile dans les environnements industriels ou médicaux où la rapidité de décision est primordiale [21][28].
- **Réduction de la charge réseau :** En ne transmettant que les données strictement nécessaires vers le Cloud, le Edge agit comme un filtre intelligent qui réduit les risques de saturation du réseau et améliore la fluidité globale du trafic [28].

I.4.3. Architecture du Edge Computing :

L'architecture du Edge Computing repose sur une organisation distribuée qui rapproche le traitement des données de leur source [27]. Elle s'articule autour de trois niveaux complémentaires, comme l'illustre la Figure I.8. :

- **La couche des appareils (Device Layer) :** Elle regroupe l'ensemble des objets connectés (capteurs, caméras, actionneurs) qui génèrent les données à la source. Ces appareils disposent d'une capacité de traitement locale limitée mais suffisante pour les tâches immédiates [21].
- **La couche Edge (Edge Layer) :** Composée de serveurs Edge et de nœuds de calcul déployés à proximité des appareils, cette couche effectue le traitement en temps réel des données collectées. Elle réduit considérablement la quantité d'informations transmises vers le Cloud [25][27].
- **La couche Cloud (Cloud Layer) :** Elle assure les traitements complexes, l'apprentissage automatique à grande échelle et le stockage à long terme des données agrégées transmises par les nœuds Edge [26].

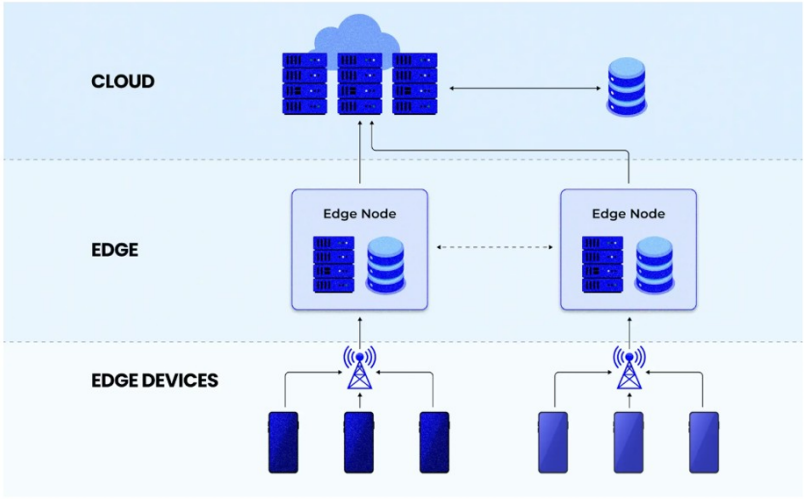


Figure I.8 : Architecture du Edge computing [14].

I.4.4. Avantages du Edge Computing :

Le Edge Computing offre des avantages concrets qui répondent aux défis actuels du numérique, en particulier dans les environnements à forte contrainte de temps [21] [28] :

- **Latence ultra-faible :** En traitant les données au plus près de leur source, le Edge élimine les délais de transmission vers le Cloud, permettant des prises de décision en temps réel indispensables dans des domaines comme la chirurgie robotique ou les véhicules autonomes [21].
- **Sécurité et confidentialité renforcées :** Les données sensibles sont traitées localement sans transiter par des serveurs distants, réduisant ainsi les risques d'interception et garantissant une meilleure conformité aux réglementations sur la protection des données [26].
- **Fonctionnement hors ligne :** Contrairement au Cloud, le Edge peut continuer à fonctionner même en cas d'interruption de la connexion internet, assurant ainsi la continuité du service dans des environnements à connectivité limitée [28].
- **Scalabilité horizontal**e : Le Edge permet d'étendre facilement l'infrastructure en ajoutant de nouveaux nœuds de calcul à proximité des sources de données, sans nécessiter d'investissements massifs dans des centres de données centralisés [27][28].

I.4.5. Limites de l' edge computing :

- **Ressources limitées :** Les dispositifs en périphérie (edge devices) disposent généralement de capacités limitées en termes de CPU, mémoire et stockage, ce qui restreint l'exécution de tâches complexes [21].
- **Sécurité et confidentialité :** Le traitement des données en dehors des centres de données centralisés augmente les risques de sécurité et de violation de la vie privée, car les nœuds edge sont plus exposés aux attaques physiques et réseau [15][21].
- **Gestion et orchestration complexes :** La grande diversité et la distribution massive des dispositifs rendent la gestion, la coordination et l'orchestration des ressources difficiles à grande échelle [21].

I.5. Comparaison entre Cloud computing vs Fog computing vs Edge computing :

Afin de mieux comprendre les différences entre les paradigmes Cloud, Fog et Edge Computing, il est essentiel de présenter une comparaison basée sur plusieurs critères techniques et architecturaux. Ces modèles de calcul se distinguent principalement par leur niveau de centralisation, leur proximité avec les utilisateurs, ainsi que leur capacité de traitement et leur latence. Le tableau 1.1 présente une synthèse comparative mettant en évidence les principales caractéristiques de chaque paradigme.

Critère	**Cloud Computing**	**Fog Computing**	**Edge Computing**
Taille	Centres de données de grande taille avec une forte capacité de calcul et de stockage.	Ressources limitées réparties sur plusieurs nœuds intermédiaires.	Ressources très limitées intégrées directement dans les dispositifs (capteurs, IoT).
Déploiement	Centralisé et fortement contrôlé.	Déploiement distribué entre le cloud et les dispositifs.	Déployé directement au niveau des dispositifs finaux.
Emplacement	Loin des utilisateurs (data centers distants).	Proche des utilisateurs (réseau intermédiaire).	Très proche des utilisateurs (au niveau des objets).
Latence	Élevée (temps de réponse plus long).	Moyenne (réduction de la latence).	Très faible (temps réel).
Mobilité	Faible prise en charge de la mobilité.	Bonne gestion de la mobilité.	Très adaptée aux environnements mobiles.
Traitement	Traitement global et centralisé.	Traitement partiellement distribué.	Traitement local et immédiat.

Tableau 1.1 : Comparaison entre le Cloud, le Fog et l'Edge computing [29].

I.6. Conclusion :

Pour résumer, ce premier chapitre nous a permis de clarifier les concepts fondamentaux du Cloud, du Fog et de l'Edge Computing. Nous avons présenté, pour chaque environnement, sa définition, ses caractéristiques, son architecture, ainsi que ses avantages et ses limite

À partir de cette analyse comparative, nous avons mis en évidence que ces trois environnements ne fonctionnent pas de manière isolée, mais qu'ils collaborent de façon complémentaire. Chacun joue un rôle spécifique en fonction de ses caractéristiques, notamment en termes de latence, de capacité de traitement et de proximité des données, ce qui permet d'assurer une gestion efficace des applications distribuées.

Le chapitre suivant sera consacré à l'étude de l'ordonnancement des tâches (Task Scheduling). Nous y présenterons les notions fondamentales associées, les principales métriques d'évaluation, ainsi que les différentes approches d'ordonnancement, notamment les méthodes heuristiques, métaheuristiques et celles basées sur l'intelligence artificielle.

CHAPITRE 2

Algorithmes d'ordonnancement des tâches dans les environnements Cloud et Fog Computing

II.1. Introduction :

Le chapitre précédent a permis de présenter les concepts fondamentaux ainsi qu'un aperçu global des environnements Cloud, Fog et Edge Computing, en mettant en évidence leurs caractéristiques, leurs architectures et leur rôle essentiel dans les systèmes distribués modernes. Ces paradigmes constituent aujourd'hui des solutions incontournables pour assurer un traitement, un stockage et une gestion efficaces et distribués des ressources informatiques.

Ce chapitre est consacré au problème de l'ordonnancement des tâches, dont l'objectif principal est d'assurer une utilisation optimale des ressources, tout en minimisant le temps d'exécution et la consommation énergétique. Il débute par l'introduction des notions fondamentales liées à l'ordonnancement, avant d'analyser les approches classiques et avancées existantes. Ensuite, il propose une vue d'ensemble des méthodes basées sur l'intelligence artificielle, en mettant en lumière leur application dans les environnements Cloud, Fog et Edge Computing, afin d'améliorer à la fois les performances globales et l'efficacité énergétique des systèmes.

II.2. Concepts fondamentaux :

II.2.1. Définition de l'ordonnancement :

L'ordonnancement des tâches(planification des tâches) fait référence à la manière dont les ressources de calcul sont attribuées aux tâches des utilisateurs au sein des systèmes distribués. Cette activité revêt une importance capitale dans les infrastructures de cloud et de fog computing, où les ressources, telles que les machines virtuelles, les unités de traitement et la bande passante, sont partagées et doivent être réparties de façon à atteindre des objectifs de performance et de qualité de service. L'objectif principal de l'ordonnancement est de « planifier les tâches de manière à réduire les temps d'attente et à optimiser les performances », tout en garantissant une utilisation « à la fois efficace et efficiente » des ressources. En plus des critères traditionnels comme le temps d'achèvement (makespan), le respect des délais (deadlines) et l'équilibrage de la charge (load balancing), les recherches récentes prennent en compte des contraintes supplémentaires liées à la consommation d'énergie.Dans ce cadre,l'ordonnancement « conscient de l'énergie » cherche à harmoniser la performance opéra-

tionnelle avec une approche de sobriété énergétique, un enjeu crucial pour les environnements cloud et fog [40].

II.2.2. Nature des algorithmes d'ordonnancement :

II.2.2.1 PREEMPTIF :

L'ordonnancement préemptif autorise l'interruption d'une tâche en cours afin de redistribuer les ressources à une autre tâche.[41], « il est possible d'interrompre chaque tâche pendant qu'elle s'exécute et de la transférer vers une autre ressource pour achever son traitement ». Cette méthode s'avère particulièrement efficace dans les environnements où il est crucial de respecter des délais stricts et de garantir une grande réactivité.

II.2.2.2 NON-PREEMPTIF :

L'ordonnancement non-préemptif se caractérise par l'impossibilité d'interrompre une tâche une fois qu'elle a débuté. Comme indiqué dans [41], « les machines virtuelles restent affectées à la tâche en cours jusqu'à son achèvement ». Cette méthode offre l'avantage de diminuer les frais associés aux changements de contexte et facilite la gestion des ressources, surtout dans des environnements distribués.

II.2.3. Définition d'une tâche (Task/Job) :

Dans les systèmes distribués tels que l'informatique en nuage (cloud computing)et l'informatique en brume(fog computing) , une tâche représente l'unité fondamentale de travail soumise à un ordonnanceur pour être assignée et traitée sur des ressources de calcul variée[42] présentent une modélisation formelle d'une tâche $task_n$ à travers un quadruplet de caractéristiques essentielles :

$$task_n = (task\_id_n, L_n, S_n, AT_n)$$

où :

- $task\_id_n$:identifiant unique de la tâche dans le système.

- L_n : longeur d'une tâche,exprimée en million d'instruction(MI),qui représente la charge de calcul à traiter .
- S_n :Taille des données de la tâche (kB, MB, GB) déterminant la charge de transmission.
- AT_n :temps d'arriver de la tâche dans le systéme, esprimé en secondes.

II.2.4. Types de tâches :

Dans les contextes de cloud et de fog computing, il est essentiel de classer les tâches en fonction de leur niveau d'interdépendance pour développer des algorithmes d'ordonnancement appropriés.une tâche est considérée comme une unité indivisible de traitement qui est gérée par l'ordonnanceur. Les recherches identifient principalement trois grandes catégories [43].

II.2.4.1 Tâches indépendantes :

Deux tâches $task_i$ et $task_j$ sont considérées comme indépendantes lorsque leur exécution ne dépend pas l'une de l'autre. Dans ce contexte, elles peuvent être traitées séparément sur différentes ressources du réseau et exécutées en même temps [43].Cette caractéristique permet de tirer pleinement parti du parallélisme que les architectures distribuées peuvent offrir. Les tâches indépendantes se divisent en deux types en fonction de leur mode d'arrivée [44] :
Les tâches périodiques : se manifestent à des intervalles fixes et peuvent être planifiées à l'aide d'algorithmes comme l'analyse Rate Monotonic (RMA), l'analyse Deadline Monotonic (DMA), l'algorithme Earliest Deadline First (EDF) ou encore le Least Laxity First (LLF).
Les tâches apériodiques : se présentent de façon imprévisible et peuvent être gérées par des méthodes telles que le traitement en arrière-plan, le serveur de tâches ou la technique Slackstealer [44].

II.2.4.2 Tâches dépendantes (Workflow) :

Deux tâches $task_i$ et $task_j$sont considérées comme dépendantes lorsque l'achèvement de la tâche $task_j$ est nécessaire avant de pouvoir commencer la tâche $task_i$ [43]. Cette contrainte de précédence requiert une attention particulière aux relations de dépendance entre les

différentes tâches lors de la planification.
Un workflow se modélise par un graphe orienté acyclique (Directed Acyclic Graph - DAG) , noté $G = (V, E)$[43].

- $V = task_1, task_2, ..., task_n$: est L'ensemble des tâches qui forment ce workflow
- $E = e_ij = (task_i, task_j)$:représente l'ensemble des arêtes orientées qui indiquent les dépendances de données existant entre ces tâches.

  Chaque tâche $task_i \in V$ est définie par un quadruplet [43].

$$task_i = (task\_id_i, L_i, S_i, MEM_i)$$

  où :

- $task_id_i$: représente son identifiant,
- L_i : indique sa charge de travail en cycles CPU,
- S_i : correspond à son besoin en stockage,
- MEM_i : désigne la quantité de mémoire requise.

Chaque arête $e_ij \in E$ est associée à un poids non négatif C_ij qui illustre le volume de données à transférer entre les deux tâches concernées.
Une tâche qui ne possède pas de prédécesseur $PERD(v_i) = \emptyset$ est désignée comme tâche d'entrée (v_{ENTRY}) , tandis qu'une tâche sans successeur($SUCC(v_i) = \emptyset$) est appelé tâche de sortie (v_{EXIT}) [43].

II.2.4.3 Ensemble de tâches (Bag of Tasks) :

Un Bag of Tasks (BoT) fait référence à un groupe de tâches qui sont indépendantes les unes des autres, sans qu'il y ait d'ordre d'exécution prédéfini, et qui peuvent être exécutées simultanément sur des ressources distribuées [43]. Ce concept est particulièrement pertinent pour les applications dites embarassingly parallel, où chaque tâche traite un segment des données sans lien avec les autres.

Dans ce type de situation, chaque tâche est associée à divers paramètres, tels que ses exigences en matière de ressources, sa taille et d'éventuelles dépendances[44] . En général, deux indicateurs temporels sont liés à chaque tâche : le temps de préparation, qui désigne le moment où toutes les données nécessaires sont prêtes, et la date de début la plus anticipée, qui indique le moment le plus précoce auquel un nœud de traitement peut commencer à exécuter la tâche [43].

II.2.5. Métriques de performance pour l'evaluation des algorithmes :

L'évaluation des algorithmes de planification dans les contextes du Fog et de l'Edge Computing se base sur diverses mesures de performance, permettant d'apprécier leur performance et leur capacité à répondre aux besoins des applications. Les indicateurs fréquemment utilisés incluent la durée d'exécution, le makespan, la latence, le respect des délais et la consommation d'énergie. Ces indicateurs fournissent un fondement objectif pour évaluer les diverses méthodes de planification et quantifier leur influence sur les performances globales du système. Dans cette perspective, le but principal est de diminuer les temps de traitement et la consommation énergétique, tout en assurant une exploitation optimale des ressources et un niveau de service satisfaisant [46][69].

II.2.5.1 Temps d'exécution (Execution Time) :

Le temps d'exécution représente la durée nécessaire pour exécuter une tâche sur une ressource de calcul donnée. Il constitue une métrique essentielle pour évaluer la performance des algorithmes d'ordonnancement dans les environnements de cloud et de fog computing.
Pour une tâche $task_n$, le temps d'exécution dépend principalement de sa longueur L_n (exprimée en millions d'instructions - MI) ainsi que de la capacité de traitement de la ressource, généralement exprimée en MIPS (Million d'Instructions Par Seconde).
Le temps d'exécution est donné par la relation suivante :

$$Execution\_time_n = \frac{L_n}{MIPS \times corr_s}$$

Où :

- L_n : longeur de la tâche $task_n$ (en MI).

- $MIPS$: capacité de calcul de la ressource.
- $corr_s$: facteur de correction lié à la performance réelle de la machine.
- $Execution\_time_n$: temps d'execution de la tâche $task_n$.

Cette métrique permet de comparer l'efficacité des différentes stratégies d'ordonnancement en fonction du temps nécessaire pour traiter les tâches sur les ressource disponibles [42].

II.2.5.2 Makespan :

Le makespan représente la durée totale nécessaire pour exécuter un ensemble de tâches sur les ressources disponibles. Il correspond au temps de fin de la dernière tâche terminée dans l'ordonnancement.
Pour un ensemble de tâches $\{task_1, task_2, \ldots, task_n\}$, le makespan est défini comme suit :

$$C_{max} = \max_{1 \leq i \leq n} (C_i)$$

Où :

- C_i : temps de fin d'exécution de la tâche $task_i$.
- C_{max} : makespan (temps total d'exécution).

Le temps de fin de chaque tâche $task_i$ dépend de son temps d'arrivée AT_i et de son temps d'exécution $Execution_time_i$, tel que :

$$C_i = AT_i + Execution\_time_i$$

Le makespan constitue une métrique essentielle pour l'évaluation des algorithmes d'ordonnancement, car il permet de mesurer la performance globale du système. En général, l'objectif principal est de minimiser C_{max} afin de réduire le temps total d'exécution et d'optimiser l'utilisation des ressources[70].

II.2.5.3 Latence (Délai de réponse) :

La latence désigne le délai total entre l'envoi d'une requête par un appareil IoT et la réception de la réponse. À la différence du Cloud, le Fog Computing diminue cette latence en rapprochant les ressources de calcul des bords du réseau[46].
Pour une tâche $task_i$ exécutée sur une ressource nœud j, le temps de réponse (Response Time) est décomposé comme suit [47] :

$$RT_{i,j} = TT_{i,j}^{Total} + Execution\_time_{i,j} + W_{i,j}$$

- Temps de transmission ($TT_{i,j}^{Total}$) :acheminement des données.
- Temps d'execution $Execution_time_{i,j}$:traitement de la tâche i sur le nœud j.
- Temps d'attente ($W_{i,j}$) : durée en file d'attente.

Pour respecter l'échéance, la condition $RT_{i,j} \leq T_i^{deadline}$ doit être remplie.
Les méthodes d'ordonnancement visant à améliorer l'efficacité temporelle comprennent l'application de l'algorithme Earliest Deadline First (EDF), le déchargement dynamique et les files d'attente avec priorité [46].

II.2.5.4 Deadline (Échéance) :

La notion d'échéance (deadline) représente une contrainte temporelle essentielle dans les systèmes IoT en temps réel, définissant le délai maximal dans lequel une tâche doit être réalisée [47].
Pour évaluer la capacité d'un algorithme d'ordonnancement à respecter cette contrainte, trois métriques sont établies.
Tout d'abord, une variable binairez_i est introduite pour signaler si l'échéance est respectée pour chaque tâche T_i : si

$$z_i = \begin{cases} 1 & \text{si } RT_{i,j} \leq T_i^{deadline} \\ 0 & \text{sinon} \end{cases}$$

Où :

- $RT_{i,j}$: temps de réponse de la tâche T_i exécutée sur le nœud F_j.

- $T_i^deadline$: échéance maximale autorisée pour la tâche T_i.
- $z_i = 1$: la tâche a terminé son exécution avant son échéance.
- $z_i = 0$: la tâche n'a pas respecté son échéance.

En deuxième lieu, le total des tâches qui respectent leur délai, désigné par NT , est obtenu en additionnant les variables individuelles :

$$NT = \sum_{\forall T_i \in T} z_i$$

Où :

- Z : ensemble de toutes les tâches à ordonnancer.
- NT : nombre de tâches ayant respecté leur échéance.

En troisième lieu, le pourcentage de tâches qui respectent leurs délais, noté $T^{\%}$, se présente de la manière suivante :

$$NT^{\%} = \frac{NT \times 100}{n}$$

Où :

- n : nombre total de tâches dans l'ensemble T.
- $NT^{\%}$: pourcentage de tâches ayant respecté leur échéance.

Cette dernière métrique représente l'indicateur de performance le plus pertinent, car elle reflète directement la capacité de l'ordonnanceur à répondre aux exigences temporelles des applications IoT critiques [47].

De plus, la contrainte de délai est incluse dans le problème d'optimisation à multiples objectifs sous la forme suivante :

$$RT_{i,j} \leq T_i^{deadline}, \quad \forall i \in T$$

Où :

- $RT_{i,j}$: temps de réponse de la tâche T_i sur le nœud F_j.
- $T_i^{deadline}$: échéance de la tâche T_i.
- $\forall i \in T$: pour toutes les tâches de l'ensemble.

Cette inégalité assure que le temps de réponse de chaque tâche ne dépasse jamais son délai, formant ainsi une contrainte stricte (hard constraint) dans le modèle d'optimisation [47].

II.2.5.5 Consommation d'énergie :

Les serveurs constituent une source majeure de consommation d'énergie dans les centres de données. De nombreuses études montrent que leur consommation électrique est proportionnelle à l'utilisation du processeur (CPU). En particulier, un serveur inactif consomme environ les deux tiers de sa puissance maximale. Le tiers restant varie de manière quasi linéaire avec l'augmentation de la charge CPU.
La puissance consommée par un serveur peut être exprimée par la relation linéaire suivante :

$$P_{server} = P_{idle} + (P_{max} - P_{idle}) \times U$$

Où :

- P_{server} : puissance instantanée du serveur (en watts (W), joule ou kw) ;
- P_{idle} : puissance consommée lorsque le serveur est inactif (aucune charge CPU) ;
- P_{max} : puissance consommée lorsque le serveur fonctionne à pleine charge (utilisation CPU maximale) ;
- U : taux d'utilisation du CPU, compris entre 0 et 1 [71].

II.2.6. Objectifs de l'ordonnancement :

L'ordonnancement des tâches implique d'attribuer un ensemble de tâches à des ressources disponibles, tout en tenant compte de diverses contraintes, dans le but d'optimiser

un ou plusieurs critères de performance [51]. Dans le cadre du Cloud computing, cette question est considérée comme un problème NP-difficile en raison de l'ampleur significative de l'espace de recherche et du nombre élevé de combinaisons possibles entre les tâches et les ressources virtualisées [52]. La résolution de ce type de problème requiert souvent l'utilisation de méthodes approchées, comme les métaheuristiques, qui sont capables d'offrir des solutions satisfaisantes dans un délai de calcul raisonnable [51].

II.2.6.1 Mono-objectif :

L'approche mono-objectif a pour but d'optimiser un seul critère. Elle est souvent utilisée lorsque l'une des exigences se démarque nettement des autres (comme par exemple, la réduction du temps de réalisation ou de la consommation d'énergie). Dans ce cas, la fonction objectif est unique et l'algorithme de résolution s'efforce d'atteindre l'optimum global [51].

II.2.6.2 Multi-objectif :

Dans des contextes réels, il arrive souvent que les objectifs soient en conflit : améliorer un critère peut nuire à un autre. L'ordonnancement multi-objectif a pour but de trouver un compromis satisfaisant entre plusieurs critères. Ce processus se caractérise par la recherche non pas d'une solution unique, mais d'un ensemble de solutions qualifiées de non-dominées (front de Pareto), où aucune amélioration d'un objectif ne peut être réalisée sans détériorer au moins un autre objectif [51]. Dans les recherches récentes appliquées au Cloud, les objectifs considérés incluent la minimisation du makespan, la réduction de la consommation énergétique et la limitation des violations des contrats de niveau de service (SLA) [52]. La fonction objectif peut alors être exprimée comme la minimisation simultanée de ces trois métriques [52] :

$$F = \min \sum (ms^n, e_{con}, sla_{violation})$$

Pour aborder ce problème, on fait appel à des métaheuristiques basées sur des populations, telles que l'algorithme d'optimisation par bousier (DBO) ou l'algorithme des colonies d'abeilles (BCO). Ces approches alternent entre des phases de diversification (exploration de l'espace de recherche) et d'intensification (recherche locale autour des solutions prometteuses) dans le but de converger vers un front de Pareto constitué de solutions de haute qualité [51, 52].

II.3. Algorithmes d'ordonnancement classiques :

Les algorithmes d'ordonnancement traditionnels forment la base des méthodes de répartition des tâches au sein des systèmes informatiques, que ce soit dans les centres de données Cloud ou dans les infrastructures Fog. Leur simplicité et leur faible coût en termes de calcul expliquent leur utilisation encore répandue, même si leurs limites se font sentir dans des environnements dynamiques et aux ressources limitées. Cette section présente quatre algorithmes clés : First Come First Served (FCFS), Shortest Job First (SJF), Round Robin (RR) et Priority Scheduling. Pour chacun d'eux, nous exposons le principe, un pseudo-code ainsi que les illustrations disponibles dans la littérature [58][61].

II.3.1. FCFS (First Come First Served) :

FCFS, souvent désigné sous le terme FIFO (First In, First Out), représente l'algorithme le plus basique : les tâches sont traitées dans l'ordre exact de leur arrivée, sans possibilité de préemption. Sa mise en œuvre est simple et son coût en calcul est nul. Néanmoins, il est affecté par l'effet de convoyage, où une tâche longue empêche toutes les tâches courtes qui arrivent après elle d'être exécutées, ce qui dégrade le temps de réponse moyen Algorithme FCFS (First-Come, First-Served) [58] : Dans les environnements Fog, FCFS est fréquemment utilisé comme point de référence pour évaluer des méthodes plus sophistiquées, mais il ne prend pas en compte la criticité des tâches ni la consommation d'énergie [60][61] La Figure II.1 présente le mécanisme d'ordonnancement FCFS, dans lequel les tâches sont traitées dans l'ordre de leur arrivée sans interruption ni réorganisation de la file d'attente.
Algorithme FCFS [58] :

Entrée : Ensemble des tâches T avec leurs temps d'arrivée et d'exécution

Sortie : Ordonnancement des tâches

Début

Initialiser une file FIFO vide.

Trier les tâches selon leur temps d'arrivée.

Pour chaque tâche $t \in T$ **faire**

Enfiler t dans la file FIFO.

Fin Pour

Tant que la file FIFO n'est pas vide **faire**

$t \leftarrow$ Défiler la première tâche.

Exécuter t jusqu'à sa terminaison.

Enregistrer l'ordre d'exécution de t.

Fin Tant que

Retourner la liste d'ordonnancement.

Fin

Dans le cadre de Fog, le FCFS est intégré par défaut dans le simulateur iFogSim sous l'appellation Edgeward ; ses performances en matière de latence, de consommation d'énergie et d'utilisation du réseau sont clairement inférieures à celles des algorithmes qui prennent en compte la criticité [60, 61].

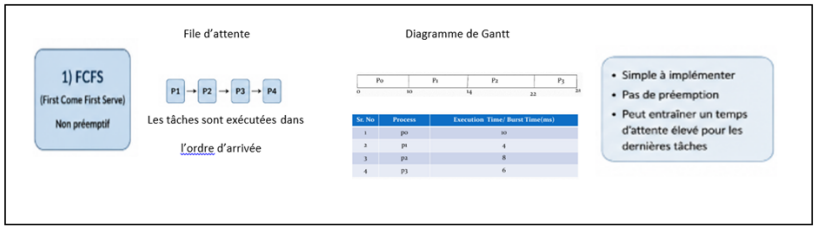


Figure II.1 : Schéma représentatif de l'algorithme FCFS [72].

II.3.2. SJF (Shortest Job First) :

L'algorithme SJF choisit la tâche dont la durée d'exécution prévue est la plus courte. Il peut être soit non préemptif (la tâche sélectionnée s'exécute jusqu'à son achèvement) soit préemptif (appelé Shortest Remaining Time First, SRTF). En théorie, SJF réduit le temps d'at-

tente moyen ainsi que le makespan. Cependant, il présente deux principaux inconvénients : la famine des tâches longues si un flux constant de tâches courtes se présente, et l'obligation de prévoir avec précision la durée d'exécution [58, 59]La Figure II.2 présente le mécanisme d'ordonnancement SJF, où les tâches ayant le temps d'exécution le plus court sont sélectionnées en priorité pour être exécutées.

Algorithm SJF nom prémitif [58, 59] :

Entrée : Ensemble des tâches T avec leur durée estimée

Sortie : Ordonnancement des tâches

Début

Initialiser la liste des tâches avec leur durée estimée.

Tant que il existe des tâches non ordonnancées **faire**

Sélectionner la tâche T^* ayant la plus petite durée.

Exécuter T^* jusqu'à sa terminaison.

Supprimer T^* de la liste des tâches.

Fin Tant que

Retourner l'ordonnancement.

Fin

Dans les architectures Fog, l'algorithme SJF a été mis en œuvre et comparé à FCFS dans des applications liées à la santé et au stationnement intelligent [61]. Les résultats indiquent que SJF diminue la latence par rapport à FCFS, mais il s'avère peu efficace pour les tâches critiques de grande taille (avec des MIPS élevés) car il tend à les retarder systématiquement [60].

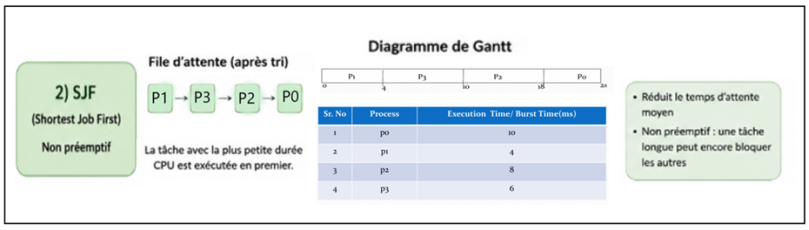


Figure II.2 : Shéma représentatif d'algorithme SJF [72].

II.3.3. Round Robin(RR) :

Le Round Robin est un système d'ordonnancement préemptif basé sur un principe de rotation. Chaque tâche se voit attribuer un quantum de temps fixe ; si elle n'est pas achevée à l'issue de ce quantum, elle est renvoyée à la fin de la file d'attente. Cette méthode assure une distribution équitable du temps processeur et prévient le risque de famine. Le choix du quantum est crucial : s'il est trop court, cela augmente le nombre de changements de contexte (overhead) ; s'il est trop long, cela nuit à la réactivité, se rapprochant alors du FCFS. Bien que ce soit une approche classique dans les systèmes d'exploitation, le Round Robin est peu souvent abordé dans les études récentes sur le Fog [58] La Figure II.3 présente le mécanisme d'ordonnancement Round Robin, dans lequel chaque tâche reçoit un quantum de temps avant de céder le processeur à la tâche suivante de la file d'attente.

Algorithme Round Robin [58] :

Entrée : Ensemble des tâches T et quantum de temps Q

Sortie : Ordonnancement des tâches

Début

Initialiser une file d'attente circulaire contenant les tâches.

Fixer le quantum de temps Q.

Tant que la file d'attente n'est pas vide **faire**

Extraire la première tâche t de la file.

Exécuter t pendant $\min(Q, \text{durée restante})$.

Si t n'est pas terminée **alors**

Réinsérer t à la fin de la file.

Fin Si

Fin Tant que

Retourner l'ordonnancement.

Fin

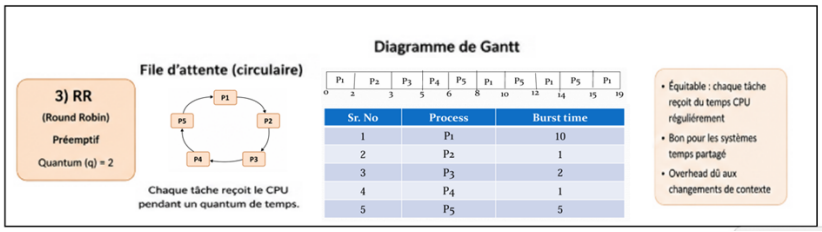


Figure II.3 : Shéma représentatif d'algorithme RR [72].

II.4. Algorithmes heuristiques et méta-heuristiques :

II.4.1. Algorithmes heuristiques :

Les algorithmes heuristiques forment un ensemble de méthodes approchées couramment employées pour traiter les problèmes d'ordonnancement NP-difficiles dans les environnements Cloud et Edge-Cloud. Leur principal avantage réside dans leur faible complexité en termes de calcul et leur capacité à offrir des solutions de qualité satisfaisante dans un délai respectant les contraintes opérationnelles [53]. Dans la littérature, ces méthodes sont généralement regroupées en trois catégories : les heuristiques simples, les métaheuristiques et les heuristiques hybrides [53]. Cette section met l'accent sur deux heuristiques simples représentatives : Min-Min et Max-Min .

II.4.1.1 Max-Min :

L'algorithme Max-Min est une heuristique de type ordonnancement séquentiel par liste de priorité, élaborée pour l'ordonnancement de tâches dans des environnements de calcul hétérogènes [53, 57].son objectif est d'équilibrer la charge entre les différentes ressources en privilégiant les tâches dont le temps d'exécution minimal est le plus élevé.

Principe de fonctionnement : pour chaque tâche qui n'est pas encore ordonnancée, on détermine son temps d'exécution minimal sur l'ensemble des ressources disponibles.On choisit ensuite la tâche pour laquelle ce temps minimal est le plus important (maximum des minima). Cette tâche est alors assignée à la ressource qui lui permet d'avoir le temps d'exécution minimal. Ce processus se répète jusqu'à ce que toutes les tâches soient complètement ordonnancées.

Cette approche évite que les tâches longues soient continuellement retardées par l'arrivée de

tâches courtes, ce qui la rend particulièrement adaptée aux environnements où les capacités des ressources varient considérablement[53].

Algorithme – Max-Min [53, 57] :

Entrée : Ensemble des tâches T et ensemble des ressources R

Sortie : Ordonnancement des tâches

Début

Calculer, pour chaque tâche T_i et chaque ressource R_j, le temps d'exécution $E(T_i, R_j)$.

Tant que il existe des tâches non ordonnancées **faire**

Pour chaque tâche T_i **faire**

Déterminer la ressource $R_{\min}(T_i)$ qui minimise $E(T_i, R_j)$.

Fin Pour

Identifier la tâche T^* ayant la plus grande valeur de $E(T_i, R_{\min}(T_i))$.

Affecter T^* à la ressource $R_{\min}(T^*)$.

Supprimer T^* de l'ensemble des tâches non ordonnancées.

Mettre à jour le temps de disponibilité de la ressource sélectionnée.

Fin Tant que

Retourner l'ordonnancement.

Fin

II.4.1.2 Min-Min :

L'algorithme Min-Min est une heuristique de type ordonnancement séquentiel par liste de priorité fréquemment employée pour l'ordonnancement des tâches dans des systèmes hétérogènes [53, 57]. Il met l'accent sur les tâches pouvant être exécutées le plus rapidement, ce qui permet de réduire le temps d'attente global.

Principe de fonctionnement est le suivant : pour chaque tâche qui n'est pas encore ordonnancée, on détermine son temps d'exécution minimal en tenant compte de toutes les ressources disponibles. On choisit ensuite la tâche dont ce temps minimal est le plus bas (minimum des minima). Cette tâche est alors assignée à la ressource qui lui garantit ce temps minimal. Ce processus se répète jusqu'à ce que toutes les tâches soient ordonnancées.

Cette méthode s'avère efficace lorsque de nombreuses tâches courtes doivent être traitées rapidement, mais elle peut également entraîner une famine (starvation) pour les tâches plus longues qui restent en attente [53].

Algorithme – Min-Min [53, 57] :

Entrée : Ensemble des tâches T et ensemble des ressources R

Sortie : Ordonnancement des tâches

Début

Calculer, pour chaque tâche T_i et chaque ressource R_j, le temps d'exécution $E(T_i, R_j)$.

Tant que il existe des tâches non ordonnancées **faire**

Pour chaque tâche T_i **faire**

Déterminer la ressource $R_{\min}(T_i)$ qui minimise $E(T_i, R_j)$.

Fin Pour

Identifier la tâche T^* ayant la plus petite valeur de $E(T_i, R_{\min}(T_i))$.

Affecter T^* à la ressource $R_{\min}(T^*)$.

Supprimer T^* de l'ensemble des tâches non ordonnancées.

Mettre à jour le temps de disponibilité de la ressource sélectionnée.

Fin Tant que

Retourner l'ordonnancement.

Fin

II.4.2. Algorithmes méta-heuristiques :

Les métaheuristiques représentent des approches d'optimisation avancées qui orientent la recherche vers des solutions de qualité en alternant entre des phases de diversification (exploration) et d'intensification (exploitation) [53]. Elles se révèlent particulièrement efficaces pour les problèmes d'ordonnancement NP-difficiles dans les environnements Cloud et Edge computing, où il est nécessaire d'optimiser simultanément plusieurs objectifs souvent contradictoires (makespan, consommation énergétique, coût, fiabilité) [52]. Cette section met en lumière deux métaheuristiques couramment citées dans la littérature : l'algorithme génétique

(GA),et le recuit simulé (SA) . Pour chacune d'elles, un cadre algorithmique est proposé.

II.4.2.1 Genetic Algorithm (GA) :

S'inspirant de la théorie de l'évolution naturelle, l'algorithme génétique fonctionne sur un ensemble de solutions (individus) qui se développent au fil des générations grâce à des processus de sélection, de croisement et de mutation [53]. La performance de chaque individu est mesurée par une fonction de fitness.

Algorithme Genetic : Algorithme Génétique [53, 57] :

Entrée : Taille de la population N, probabilité de croisement p_c, probabilité de mutation p_m

Sortie : Meilleure solution trouvée

Début

Initialiser une population de N individus aléatoirement.

Évaluer la fitness $f(x)$ de chaque individu.

Tant que le critère d'arrêt n'est pas atteint **faire**

Sélectionner des parents selon leur fitness (tournoi, roulette, etc.).

Appliquer le croisement avec probabilité p_c pour générer des enfants.

Appliquer la mutation avec probabilité p_m sur les enfants.

Évaluer la fitness des enfants.

Remplacer une partie de la population par les enfants (élitisme ou générationnel).

Fin Tant que

Retourner le meilleur individu rencontré.

Fin

Cet algorithme a été fréquemment utilisé pour l'ordonnancement dans le Cloud, souvent en association avec d'autres techniques afin de répondre à des objectifs variés [53].

II.4.2.2 Recuit simulé :

Le recuit simulé s'inspire du processus physique de recuit des métaux. Il examine l'espace des solutions en acceptant parfois des mouvements moins favorables selon une probabilité qui diminue avec la température, ce qui lui permet d'éviter les optima locaux [53].

Algorithme Recuit Simulé [53, 57] : Algorithme Recuit Simulé :

Entrée : Solution initiale x, température initiale T, température minimale T_{min}

Sortie : Meilleure solution trouvée

Début

Générer une solution initiale x aléatoirement.

Initialiser la température T et définir le schéma de refroidissement.

Évaluer la fonction objectif $f(x)$.

Tant que $T > T_{min}$ **faire**

Générer une solution voisine x' de x.

Calculer $\Delta = f(x') - f(x)$.

Si $\Delta < 0$ **ou** $\exp(-\Delta/T) > rand(0,1)$ **alors**

$x \leftarrow x'$

Fin Si

Mettre à jour la température T selon le schéma de refroidissement.

Fin Tant que

Retourner la meilleure solution rencontrée.

Fin

Cet algorithme est connu pour sa solidité, même si sa convergence peut être lente lorsque le refroidissement est trop rapide [53]. Il a été employé dans des méthodes hybrides pour l'ordonnancement en tenant compte des contraintes de coût et de temps [52].

II.5. Algorithmes avancés basés sur l'IA :

L'organisation des tâches dans les environnements *Cloud* et *Cloud,Fog et Edge* représente un défi dynamique et NP-difficile, pour lequel les méthodes traditionnelles (heuristiques

et métaheuristiques) montrent leurs limites face à la variabilité des charges de travail et à la diversité des objectifs (temps de réponse, consommation d'énergie, coût). Afin de surmonter ces verrous, les approches basées sur l'intelligence artificielle, notamment l'apprentissage profond (*deep learning*) et l'apprentissage par renforcement, ont récemment suscité un intérêt croissant. Les méthodes d'apprentissage profond permettent d'extraire automatiquement des représentations complexes à partir des données, tandis que les algorithmes d'apprentissage par renforcement (*Reinforcement Learning* — RL) permettent à un agent d'acquérir une politique d'ordonnancement optimale en interagissant directement avec l'environnement, sans nécessiter de modèle analytique préétabli [54, 55]. Cette section met en lumière les principales variantes de ces approches avancées exploitées dans la littérature pour l'ordonnancement dans le *Cloud*, notamment les modèles fondés sur l'apprentissage profond, ainsi que les algorithmes de RL tels que *Q-Learning*, *Deep Q-Network* (DQN) et *Double Deep Q-Network* (DDQN).

II.5.1. Deep Learning based Scheduling :

Les méthodes d'apprentissage profond (*deep learning*) permettent de modéliser la complexité et la dynamique des environnements *Fog* et *Cloud computing*, facilitant ainsi l'optimisation des décisions d'ordonnancement. Dans cette section, nous allons explorer un algorithme basé sur CNN représentatif tiré des recherches récentes.

II.5.1.1 Approche basée sur CNN :

Les réseaux de neurones convolutifs (CNN) sont employés pour identifier des caractéristiques spatiales à partir des données de charge (CPU, mémoire, réseau) disposées sous forme de matrices 2D. Dans EcoTaskSched [62], un modèle hybride combinant CNN et BiLSTM est présenté pour un ordonnancement éco-énergétique. Le CNN convertit l'état du système en une représentation compacte des caractéristiques, après quoi une couche de décision binaire (*Fog* ou *Cloud*) est mise en œuvre.

Algorithme Extraction CNN [62] :

Algorithme basé sur CNN pour la prise de décision Fog/Cloud :

Entrée : Métriques des ressources du système (CPU, mémoire, bande passante, charge des nœuds, etc.)

Sortie : Décision d'exécution (Fog ou Cloud)

Début

Collecter les métriques des ressources du système.

Transformer les métriques en une matrice bidimensionnelle représentant l'état du système.

Appliquer une couche convolutionnelle Conv1D (64 filtres, noyau = 3, activation ReLU) pour extraire les caractéristiques.

Appliquer une opération de pooling pour réduire la dimension et conserver les informations essentielles.

Aplatir (Flatten) les caractéristiques extraites.

Passer les données à une couche dense entièrement connectée.

Appliquer une fonction sigmoïde pour la classification binaire.

Retourner la décision optimale (Fog ou Cloud).

Fin

II.5.2. Reinforcement Learning (RL) :

Le cadre général du RL décrit le problème d'ordonnancement comme un processus décisionnel de Markov (Markov Decision Process, MDP), qui est défini par un ensemble d'états S, d'actions A, d'une fonction de transition de probabilité P et d'une fonction de récompense R [54, 55]. À chaque instant, l'agent observe l'état actuel $s_t \in S$, choisit une action $a_t \in A$ selon une politique donnée π, reçoit une récompense immédiate r_t et passe à l'état suivant s_{t+1}. L'objectif consiste à apprendre une politique π^* qui maximise la somme actualisée des récompenses futures.

II.5.2.1 Q-Learning :

Le Q-learning est un algorithme d'apprentissage par renforcement hors politique off-policy qui acquiert de façon itérative la valeur d'un couple état-action, notée $Q(s, a)$, qui représente l'espérance de la récompense future cumulée en exécutant l'action a dans l'état s, puis en

suivant la politique optimale. La mise à jour se fait selon l'équation [55] :

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

où α est le taux d'apprentissage et γ le facteur d'actualisation.

Dans un environnement Cloud, l'état peut représenter la charge des machines virtuelles (VM), la file d'attente des tâches ainsi que la disponibilité des ressources ; les actions se traduisent par l'attribution d'une tâche à une VM ou par une décision de mise en attente ; la récompense est établie sur la base de métriques telles que le temps de réponse, l'utilisation des ressources ou le coût [54]. Algorithme Q-Learning tabulaire [53, 57] :

Entrée : Taux d'apprentissage α, facteur de discount γ, paramètre ε

Sortie : Politique optimale π

Début

Initialiser la table $Q(s,a)$ à zéro.

Pour chaque épisode faire

Initialiser l'état s.

Tant que s n'est pas terminal **faire**

Choisir une action a selon une politique ε-greedy basée sur Q.

Exécuter l'action a, observer la récompense r et le nouvel état s'.

Mettre à jour :

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

$s \leftarrow s'$.

Fin Tant que

Fin Pour

Retourner la politique $\pi(s) = \arg\max_a Q(s,a)$.

Fin

II.5.2.2 Deep Q-Network (DQN) :

Le Deep Q-Network (DQN) est une extension du Q-Learning qui utilise un réseau de neurones profond pour approximer la fonction de valeur d'action $Q(s, a)$. Cette approche permet de traiter efficacement des espaces d'états de grande dimension où les méthodes tabulaires classiques deviennent impraticables. Le réseau reçoit l'état courant comme entrée et produit les valeurs Q associées à toutes les actions possibles. Ainsi, l'agent peut sélectionner l'action maximisant la récompense cumulée attendue[56].

Afin d'améliorer la stabilité de l'apprentissage, DQN utilise deux mécanismes principaux : le *replay buffer* et le réseau cible. Le *replay buffer* stocke les expériences passées sous forme de transitions (s, a, r, s'). Lors de l'entraînement, des mini-lots sont échantillonnés aléatoirement à partir de cette mémoire, ce qui permet de réduire la corrélation entre les observations successives. Le réseau cible est une copie du réseau principal dont les paramètres sont mis à jour périodiquement afin de fournir des estimations plus stables pendant l'apprentissage.La valeur cible utilisée pour la mise à jour du réseau est donnée par :

$$y = r + \gamma \max_{a'} \hat{Q}(s_{t+1}, a')$$

où :

- r représente la récompense immédiate,
- γ le facteur d'actualisation et
- $\hat{Q}$ le réseau cible

Algorithme Deep Q-Network (DQN) [53, 57] :

Entrée : Taux d'apprentissage α, facteur de discount γ, taille du mini-lot, fréquence de mise à jour C

Sortie : Réseau Q entraîné θ

Début

Initialiser le réseau principal Q avec des poids aléatoires θ.

Initialiser le réseau cible $\hat{Q}$ avec $\hat{\theta} \leftarrow \theta$.

Initialiser le replay buffer B.

Pour chaque épisode faire

Initialiser l'état s.

Tant que l'état s n'est pas terminal **faire**

Choisir une action a selon une politique ε-greedy.

Exécuter a, observer la récompense r et le nouvel état s'.

Stocker la transition (s, a, r, s') dans le buffer B.

Échantillonner un mini-lot depuis B.

Calculer la cible y :

$$y = r + \gamma \max_{a'} \hat{Q}(s', a'; \hat{\theta})$$

Calculer la perte entre y et $Q(s, a; \theta)$.

Mettre à jour les poids θ du réseau principal.

Si nombre de pas mod$C = 0$ **alors**

Mettre à jour le réseau cible : $\hat{\theta} \leftarrow \theta$.

Fin Si

$s \leftarrow s'$.

Fin Tant que

Fin Pour

Fin

II.5.2.3 Double Deep Q-Network (DDQN) :

Bien que le DQN ait montré des performances remarquables dans de nombreux problèmes de prise de décision séquentielle, il présente une limitation importante : la surestimation des valeurs Q. Ce phénomène est dû au fait que le même réseau est utilisé à la fois pour

sélectionner et évaluer l'action optimale lors du calcul de la cible d'apprentissage.
Le Double Deep Q-Network (DDQN) a été proposé pour réduire ce biais de surestimation. Son principe consiste à dissocier les étapes de sélection et d'évaluation des actions. Le réseau principal est utilisé pour choisir l'action optimale tandis que le réseau cible est chargé d'évaluer cette action. Cette séparation permet d'obtenir des estimations plus précises des valeurs Q et améliore la stabilité de l'apprentissage[73].
La fonction cible utilisée dans DDQN est définie par :

$$y = r + \gamma \hat{Q}(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a'))$$

où :

- Q désigne le réseau principal .
- $\hat{Q}$ le réseau cible.

Grâce à cette dissociation, DDQN réduit le biais de surestimation présent dans DQN et améliore la qualité de la politique apprise [73].

Algorithme Double Deep Q Network (DDQN) : Algorithme Double Deep Q-Network (DDQN) [53, 57] :

Entrée : Taux d'apprentissage α, facteur de discount γ, taille du mini-lot, fréquence de mise à jour C

Sortie : Réseau Q entraîné θ

Début

Initialiser le réseau principal Q avec des poids aléatoires θ.

Initialiser le réseau cible $\hat{Q}$ avec $\hat{\theta} \leftarrow \theta$.

Initialiser le replay buffer B.

Pour chaque épisode faire

Initialiser l'état s.

Tant que l'état s n'est pas terminal **faire**

Choisir une action a selon la stratégie ε-greedy.

Exécuter l'action a, observer la récompense r et le nouvel état s'.

Stocker la transition (s, a, r, s') dans le buffer B.

Échantillonner un mini-lot depuis B.

Sélectionner l'action optimale dans s' à l'aide du réseau principal Q :

$a^* = \arg\max_{a'} Q(s', a'; \theta)$

Évaluer cette action avec le réseau cible $\hat{Q}$ pour calculer la cible :

$$y = r + \gamma \hat{Q}(s', a^*; \hat{\theta})$$

Calculer la perte entre y et $Q(s, a; \theta)$.

Mettre à jour les poids θ du réseau principal.

Si nombre de pas $\bmod C = 0$ **alors**

Mettre à jour le réseau cible : $\hat{\theta} \leftarrow \theta$.

Fin Si

$s \leftarrow s'$.

Fin Tant que

Fin Pour

Fin

II.6. Quelques travaux de recherche sur l'ordonnancement Eco-énergétique en Cloud et Fog Computing :

Le tableau II.1 résume quelques travaux récents portant sur l'ordonnancement des tâches conscient de l'énergie dans des environnements cloud et fog computing, en mettant en évidence leurs objectifs, métriques et principales limites :

Référence	Année	Type d'environnement	Algorithme utilisé	Objectifs	Résultats / Avantages	Limites
[47]	2024	Fog computing	Heuristique temps réel Energy & Cost Aware	Réduire la consommation énergétique et le coût tout en respectant les deadlines	Amélioration de l'efficacité énergétique et réduction du coût et des violations de deadlines	Forte dépendance au modèle de prix et de tarification des ressources
[64]	2023	Cloud computing	Algorithme heuristique Energy-Aware	Minimiser makespan et la consommation énergétique	Amélioration de +30% sur le makespan et +20% sur l'efficacité énergétique	Moins adapté aux applications IoT/temps réel
[65]	2024	Fog–Cloud computing	Méta-heuristique d'optimisation de workflow	Optimiser énergie et makespan simultanément	Bon compromis multi-objectifs entre énergie et performance temporelle	Complexité de paramétrage du front de Pareto
[66]	2022	Fog computing IoT	Algorithme heuristique semi-greedy	Minimiser énergie et délais tout en respectant deadlines	Réduction des violations de deadline jusqu'à ∼97%	Modèle MINLP coûteux en temps de calcul
[42]	2026	Fog / IoT	Approche hybride méta-heuristique	Optimiser énergie et makespan dans IoT	Réduction du makespan et de l'énergie	Overhead de calcul élevé
[62]	2025	Fog–Cloud / ML	Deep Reinforcement Learning hybride	Minimiser énergie et makespan	Surpasse les modèles de base en énergie et overhead	Besoin de grands jeux de données
[67]	2023	Fog / IoT	Algorithme heuristique Energy Aware	Réduire la consommation d'énergie	Stratégie efficace pour applications IoT	Ne prend pas en compte le coût financier
[68]	2022	Cloud / Fog-IoT	Méta-heuristique évolutionnaire	Optimiser énergie, makespan et coût	Bon équilibre multi-objectifs	Sensibilité élevée aux paramètres

Tableau II.1 : Synthèse des travaux liés à l'ordonnancement Eco-énergétique.

Les travaux analysés indiquent que les approches heuristiques permettent de réduire le temps d'exécution et la consommation énergétique avec une faible complexité de calcul, mais leur efficacité demeure limitée dans des environnements dynamiques et en temps réel [47], [64], [67]. Les méthodes méta-heuristiques [65], [42], [68] offrent un meilleur équilibre entre énergie, makespan et coût grâce à leurs capacités d'optimisation multi-objectifs. Toutefois, elles se caractérisent souvent par un temps de calcul élevé et une forte sensibilité aux paramètres. Les approches fondées sur l'apprentissage automatique et l'apprentissage par renforcement profond, telles qu'EcoTaskSched [62], améliorent davantage les performances en s'adaptant de manière dynamique à l'état du système. Malgré leur efficacité, elles exigent des ressources de calcul importantes et de vastes ensembles de données pour l'apprentissage. En conséquence, les approches hybrides combinant heuristiques, méta-heuristiques et intelligence artificielle apparaissent comme les solutions les mieux adaptées aux environnements Cloud–Fog–IoT modernes.

II.7. Conclusion :

Dans ce chapitre, nous avons étudié les principaux algorithmes d'ordonnancement des tâches ainsi que plusieurs travaux de recherche relatifs à l'optimisation énergétique dans les environnements Cloud et Fog Computing. Cette analyse a permis de mettre en évidence les caractéristiques, les avantages et les limites des différentes approches proposées dans la littérature.

Le chapitre suivant sera consacré à la mise en œuvre d'une simulation permettant d'évaluer et d'optimiser l'ordonnancement des tâches, dans le but de réduire la consommation énergétique et le temps d'exécution tout en respectant les contraintes de délai.

CHAPITRE 3

Simulation et évaluation des algorithmes d'ordonnancement

III.1. Introduction :

Ce chapitre propose une étude comparative des algorithmes d'ordonnancement des tâches IoT dans les environnements Cloud et Fog Computing.. L'objectif principal de cette étude est de réduire la consommation énergétique et le Makespan et le respect de délai afin d'améliorer les performances du système proposé.

Ce chapitre est structuré en deux sections principales. La première section présente une étude comparative des simulateurs de Cloud et de Fog Computing ainsi que les outils et l'environnement de développement utilisés. La deuxième section est consacrée à la conception et à la réalisation du système proposé. Elle est subdivisée en quatre parties : la première décrit l'architecture du système proposé, la deuxième présente son implémentation, la troisième expose les métriques de performance utilisées pour l'évaluation, et la quatrième est dédiée à l'analyse des résultats obtenus.

III.2. Présentation des outils et de l'environnement de développement :

III.2.1. Étude comparative des simulateurs Cloud/Fog Computing :

Les simulateurs jouent un rôle important dans l'étude et l'évaluation des performances des environnements Cloud et Fog Computing. Ils permettent de tester différents scénarios, d'analyser le comportement du système et de mesurer plusieurs paramètres tels que la consommation énergétique, le temps d'exécution et l'utilisation des ressources, sans avoir besoin d'une infrastructure réelle.

Il existe aujourd'hui plusieurs simulateurs dédiés aux architectures Cloud/Fog/IoT, chacun possédant des caractéristiques et des fonctionnalités spécifiques selon le domaine d'utilisation. Dans le cadre de ce travail, une étude comparative des principaux simulateurs a été réalisée afin de sélectionner l'outil le plus adapté aux besoins du projet.

Cette comparaison repose sur plusieurs critères importants, notamment la prise en charge des environnements Cloud, Fog et IoT, la gestion de l'ordonnancement des tâches, la modélisation énergétique, le calcul du Makespan, la scalabilité ainsi que la possibilité d'intégrer des techniques d'intelligence artificielle comme le reinforcement learning.

Présentation des simulateurs :

CloudSim est un simulateur développé en Java pour les environnements Cloud Computing. Il permet de simuler les centres de données, les machines virtuelles et les algorithmes d'ordonnancement. Cependant, il ne prend pas en charge directement les environnements Fog et Edge. [74].

iFogSim2 est une extension de CloudSim conçue pour les architectures Fog et IoT. Il permet de simuler la consommation énergétique, la latence et l'ordonnancement des tâches dans les environnements distribués [75].

FogNetSim++ est un simulateur basé sur OMNeT++ utilisé pour les environnements Fog Computing. Il permet de simuler les communications réseau et les performances des nœuds Fog et IoT [76].

PyFog est un framework Python destiné à la simulation des environnements Fog. Il offre une architecture simple et flexible adaptée au développement rapide des scénarios de simulation [77].

YAFS est un simulateur open-source développé en Python pour les architectures Fog et IoT. Il permet de gérer des scénarios dynamiques et l'exécution des tâches dans les réseaux distribués [78].

SimPy est une bibliothèque Python de simulation à événements discrets. Il offre une grande flexibilité pour créer des environnements Cloud/Fog personnalisés et facilite l'intégration des algorithmes d'intelligence artificielle comme le Q-Learning [79].

Simulateur	Langage	Open Source	Cloud	Fog	Edge	IoT	Ordonnanceur	Makespan	Modèle énergie	Scalabilité	Intégration IA
CloudSim	Java	Oui	Oui	Non	Non	Oui	Oui	Non	Basique	Moyenne	Non
iFogSim2	Java	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Bonne	Non
FogNetSim++	C++	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Élevée	Implémentable
PyFog	Python	Oui	Oui	Oui	Oui	Oui	Implémentable	Implémentable	Implémentable	Moyenne	Implémentable
YAFS	Python	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Bonne	Implémentable
SimPy	Python	Oui	Implémentable	Implémentable	Implémentable	Implémentable	Implémentable	Oui	Implémentable	Élevée	Implémentable

Tableau III.1 : Comparaison entre les différents simulateurs

D'après cette comparaison du tableau III.1, nous remarquons que certains simulateurs sont spécialisés dans les environnements Cloud, tandis que d'autres prennent en charge les architectures Fog et IoT. Cependant, peu de simulateurs offrent une grande flexibilité pour l'intégration des techniques d'intelligence artificielle.

Dans le cadre de cette étude, le choix de SimPy a été retenu suite à une analyse compara-

tive des principaux simulateurs utilisés dans les environnements Cloud et Fog Computing. Contrairement à des outils spécialisés tels que CloudSim, iFogSim2 ou YAFS, SimPy offre une approche plus flexible basée sur la simulation à événements discrets, permettant une modélisation fine et personnalisée des systèmes distribués.

Ce choix est principalement motivé par sa simplicité d'utilisation, sa compatibilité avec le langage Python et sa grande flexibilité dans la modélisation des processus, des ressources et des files d'attente. SimPy permet également de représenter efficacement les environnements hétérogènes tels que le Cloud, le Fog, l'Edge et l'IoT.

En outre, cette bibliothèque facilite l'implémentation des algorithmes d'ordonnancement ainsi que l'intégration des Algorithmes reinforcement learning (Q-Learning,DDQN). Cette flexibilité a permis de concevoir un système de simulation adapté aux objectifs de cette étude, en particulier pour l'évaluation de la consommation énergétique et du Makespan.

III.2.2. Outils et environnement de développement :

III.2.2.1 Configuration matérielle :

Le travail proposé dans ce mémoire a été implémenté et testé dans un environnement possédant les caractéristiques suivantes :

- Processeur : Intel(R) Core(TM) i5-5300U CPU @ 2.30 GHz
- Fréquence : 2.29 GHz
- Mémoire vive (RAM) : 8 Go
- Système d'exploitation : Windows 10 (64 bits)

III.2.2.2 Environnement logiciel :

III.2.2.2.1 Python :

Python est un langage de programmation open-source puissant et facile à apprendre. Il possède des structures de données de haut niveau efficaces et une approche simple mais efficace de la programmation orientée objet. La syntaxe élégante et le typage dynamique de Python, ainsi que sa nature interprétée, en font un langage idéal pour l'écriture de scripts et le dé-

veloppement rapide d'applications dans de nombreux domaines et sur la plupart des plates-formes. L'interpréteur Python et la vaste bibliothèque standard sont disponibles gratuitement sous forme de source ou de binaire pour toutes les principales plates-formes [82].

III.2.2.2.2 Visual Studio Code :

Visual Studio Code (Vs Code) est un éditeur de code source et un environnement de développement intégré (IDE) développé par Microsoft. Open-source et multiplate-forme, il fonctionne sous Windows, Linux et macOS. Initialement conçu pour les développeurs web, il supporte également de nombreux autres langages de programmation, tels que C++, C, Python, Java, entre autres. Vs Code propose de nombreuses fonctionnalités, notamment la coloration syntaxique, l'auto-complétion, la détection d'erreurs, la navigation dans le code, le débogage, la gestion de versions et une intégration poussée avec Git. De plus, il est hautement extensible grâce à une large gamme d'extensions créées par la communauté, permettant aux utilisateurs d'adapter l'éditeur à leurs besoins spécifiques[81].

III.2.2.2.3 Bibliothèques Python utilisées :

Random : est un module Python qui regroupe diverses fonctions dédiées à la manipulation de valeurs aléatoires. La génération des nombres aléatoires repose sur le générateur de nombres pseudo-aléatoires Mersenne Twister, reconnu comme l'un des générateurs les plus éprouvés et largement utilisés en informatique [84].

Matplotlib : est une bibliothèque Python open-source. Son objectif initial était de reproduire les fonctionnalités graphiques de MATLAB en Python. Aujourd'hui, elle est devenue la référence pour la création de graphiques, animés et interactifs dans l'écosystème Python [83].

NumPy : est une bibliothèque open-source fondamentale en Python destinée au calcul scientifique. Elle fournit un support efficace pour la manipulation de tableaux multidimensionnels (arrays) ainsi que des fonctions mathématiques optimisées pour les opérations vectorielles et matricielles. NumPy est largement utilisée dans les domaines de la science des données, de l'apprentissage automatique et de la simulation numérique grâce à ses performances élevées et sa simplicité d'utilisation [80].

Torch (PyTorch) : est un framework open-source de Deep Learning développé par Meta. Il fournit des tenseurs accélérés par GPU ainsi qu'un système de différentiation automatique appelé Autograd.

PyTorch est largement utilisé en recherche en intelligence artificielle pour :

- construire des réseaux de neurones,
- entraîner des modèles de Deep Learning,
- implémenter des algorithmes de Reinforcement Learning comme DQN [85].

Time : Le module time est un module standard de Python qui fournit des fonctions pour manipuler le temps et les horloges système. Il permet notamment de mesurer la durée d'exécution d'un programme, de suspendre l'exécution d'un script pendant un temps donné ou de récupérer l'heure actuelle (time()). Ce module est couramment utilisé dans les simulations, les boucles d'attente, les mesures de performances, ou encore les animations temporelles [86].
datetime : Le module datetime permet de manipuler les dates et heures en Python.
Il est utilisé pour :

- générer des timestamps,
- enregistrer l'heure des simulations,
- gérer les durées et calendriers [87].

csv : est un module Python permettant de lire et écrire des fichiers CSV (Comma-Separated Values).Il est souvent utilisé pour sauvegarder des résultats de simulation, des métriques ou des datasets [87].
SimPy : est une bibliothèque Python de simulation à événements discrets (Discrete Event Simulation). Elle permet de modéliser et simuler des systèmes complexes évoluant dans le temps, tels que les systèmes distribués, les réseaux Cloud/Fog et les environnements IoT. SimPy repose sur une approche basée sur les processus et les ressources, ce qui facilite la modélisation des files d'attente, des serveurs et des tâches concurrentes. Elle est particulièrement adaptée à l'évaluation des algorithmes d'ordonnancement et à l'intégration de techniques d'intelligence artificielle comme le Q-Learning [79].

III.2.2.3 Paramètres de simulation :

Dans le cadre de l'environnement de simulation de cette étude, plusieurs paramètres ont été définis afin de représenter de manière réaliste les différentes couches du système. Ces

dernières comprennent le Fog, le Cloud et les tâches à exécuter. Dans ce contexte, il est essentiel de reconnaître l'existence de paramètres spécifiques pour chaque composant du système. Pour le Cloud, les machines physiques PMs sont caractérisées par leur Vitesse du processeur (MIPS) , le nombre de cœurs et la mémoire vive (RAM). Ces dernières sont définies par deux paramètres énergétiques : la consommation en mode inactif (ou « idle power ») et la consommation en mode actif (ou « busy power »).Le tableau III.2 récapitule les caractéristiques matérielles et énergétiques des machines physiques du Cloud. Chaque PM héberge plusieurs machines virtuelles VMs, ce qui permet une optimisation de l'utilisation des ressources et une flexibilité accrue dans l'exécution des tâches.Le tableau III.3 présente la configuration des machines virtuelles utilisées dans l'environnement de simulation.

Paramètres	**Valeurs**
Nombre de PM	3
Vitesse du processeur (MIPS)	5000, 7000, 9000
Nombre de cœurs	8, 16 ou 32 cœurs selon le PM
Mémoire vive (Go)	16, 32 ou 64 Go selon le PM
puissance inactive	[120,250]
puissance active	[200,400]
Bande passante (MB/s)	1000
Vitesse du signal (m/s)	2×10^8
Distance entre cloud et IoT (m)	1000000

Tableau III.2 : Paramètres des PMs

Paramètres	**Valeurs**
Nombre de VM	Chaque PM contient 4 VM
Vitesse du processeur (MIPS)	[1500,3000]
Nombre de cœurs	2, 4 ou 8 cœurs selon la VM
Mémoire vive (Go)	4, 8 ou 16 Go selon la VM
puissance inactive	[40,60]
puissance active	[70,120]

Tableau III.3 : Paramètres des VMs

Au niveau de la couche Edge/Fog, les nœuds disposent de ressources de calcul plus limitées que celles du Cloud. Leur caractérisation repose sur plusieurs paramètres essentiels, tels que la Vitesse du processeur (MIPS), le power en mode actif et inactif, le nombre de cœurs et la capacité de la mémoire vive (RAM).Les caractéristiques détaillées des nœuds Edge/Fog retenus dans notre environnement de simulation sont présentées dans le tableau III.4.

Paramètres	**Valeurs**
Nombre de nœuds	16
Vitesse du processeur (MIPS)	[500,4000]
Nombre de cœurs	1, 2 ou 4 cœurs selon le nœud
Mémoire vive (Go)	1, 2, 4 ou 8 Go selon le nœud
puissance inactive	[20,50]
puissance active	[35,160]
Bande passante (Mb/s)	100
Vitesse du signal (m/s)	2×10^8
Distance entre Fog et IoT (m)	10500

Tableau III.4 : Paramètres des nœuds Edge/Fog

En ce qui concerne les tâches, elles se distinguent principalement par leur longueur, exprimée en millions d'instructions (MI), ainsi que par les ressources mémoire (RAM) nécessaires à leur exécution. Chaque tâche est associée à une échéance, ou deadline, ainsi qu'à une taille de données(input et output), Dans le cadre de cette étude, l'ensemble des paramètres est généré en fonction des caractéristiques des applications.Les différentes caractéristiques des tâches et les intervalles de valeurs retenus pour leur génération sont récapitulés dans le tableau III.5.

Paramètres	**Valeurs**
longueur (MI)	$[500, 9000]$
ressources mémoire (RAM)	$[1, 3]$ Go
data size (Mb)	$[0.1, 7]$
deadline (s)	$[1, 8]$

Tableau III.5 : Paramètres des tâches

III.3. Conception, implémentation et évaluation du système proposé :

III.3.1. Architecture du système proposé :

Afin de mieux comprendre le fonctionnement global du système proposé, nous présentons dans cette section l'architecture générale de l'environnement de simulation Cloud/-Fog/IoT. Cette architecture est organisée en plusieurs couches hiérarchiques, notamment la couche IoT, la couche Fog/Edge et la couche Cloud, interconnectées par un broker de gestion des tâches. Ce dernier joue un rôle central dans la prise de décision et l'orientation des tâches vers les ressources les plus appropriées en fonction de leurs caractéristiques.La figure III.1 illustre l'architecture générale de l'environnement Cloud/Fog/IoT proposé, ainsi que les échanges entre les différentes couches du système.

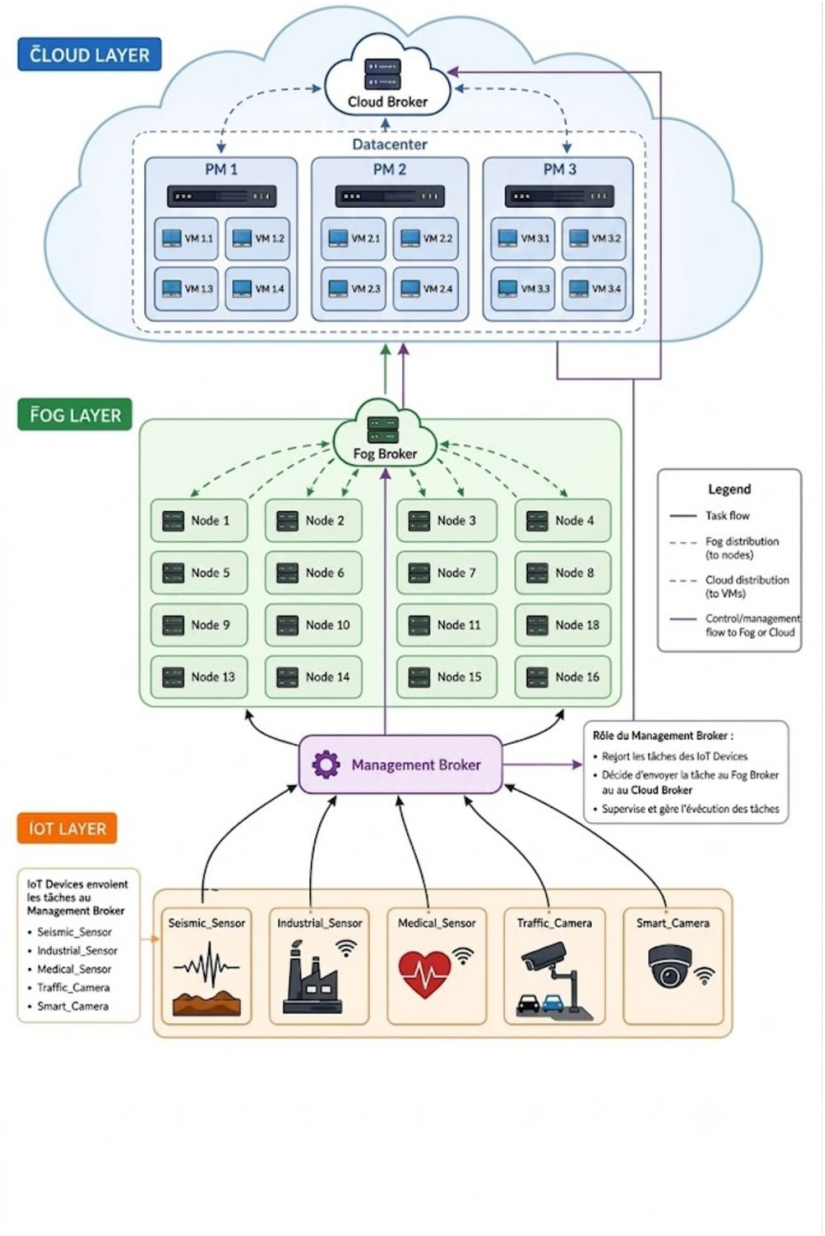


Figure III.1 : Architecture du système proposé.

III.3.1.1 Couche IoT :

Cette couche permet de simuler un environnement Internet des objets (IoT) en générant des tâches représentant les flux de données issus de dispositifs hétérogènes. L'objectif principal est de fournir une charge de travail réaliste afin d'évaluer et de comparer les performances des algorithmes d'ordonnancement dans une architecture hybride Fog/Cloud/IoT.

Chaque application est modélisée comme un service IoT (détection de mouvement, électrocardiogramme (ECG), reconnaissance faciale, etc.). Elle est caractérisée par plusieurs paramètres essentiels, notamment la charge de calcul (length), la deadline, la taille des données

ainsi que les ressources mémoire requises.
Afin de reproduire la variabilité des charges de travail réelles, ces paramètres sont générés de manière aléatoire selon des intervalles prédéfinis. Les applications sont ensuite regroupées au sein de dispositifs IoT statiques représentant différents types de capteurs et de caméras (médicaux, industriels et intelligents). Chaque dispositif génère des lots de tâches en sélectionnant aléatoirement ses applications.
Enfin, la fonction `generate_all_tasks` assure la génération globale de l'ensemble des tâches, lesquelles sont ensuite mélangées afin de simuler un flux de données réaliste, asynchrone et non ordonné. Ce modèle permet ainsi de reproduire fidèlement un environnement de charge dynamique destiné à l'évaluation des systèmes Fog/Cloud.

III.3.1.2 Couche Fog :

La couche Fog constitue une couche intermédiaire de traitement dans l'architecture hybride Fog/Cloud/IoT, située entre les dispositifs de l'Internet des objets (IoT) et le Cloud. Elle est composée d'un ensemble de nœuds hétérogènes (Fog Nodes), disposant de ressources limitées mais situés à proximité des sources de données. Cette proximité permet de réduire significativement le temps de réponse des applications et d'améliorer leur réactivité.
Cette couche reçoit les tâches générées par les objets IoT et les exécute localement en s'appuyant sur différentes politiques et algorithmes d'ordonnancement. Ceux-ci incluent les algorithmes classiques (FCFS,Min-Min et Max-Min), les approches méta-heuristiques (notamment les algorithmes génétiques), et les méthodes basées sur le Reinforcement Learning (Q Learning et DDQN).
La gestion, la sélection et l'affectation des tâches vers les différents nœuds Fog sont assurées par le FogBroker. Ce dernier joue un rôle central dans le processus de décision en sélectionnant le nœud le plus approprié en fonction de l'algorithme d'ordonnancement utilisé ainsi que de l'état courant des ressources disponibles.
L'objectif principal de cette couche est d'assurer un traitement rapide des tâches urgentes tout en optimisant l'utilisation des ressources de calcul et en réduisant la charge de travail transférée vers le Cloud.

III.3.1.3 Couche Cloud :

Ce passage représente la couche de Cloud Computing dans un système Fog/Cloud. Il s'agit du niveau supérieur de l'architecture, utilisé pour traiter les tâches qui ne sont pas exécutées dans la Fog Layer en raison de leur complexité . Cette couche repose sur des centres de données virtuels composés de PM, chacune hébergeant plusieurs VMs fonctionnant comme des unités de calcul indépendantes. L'objectif principal est de fournir une forte capacité de calcul afin de traiter des tâches lourdes dans un environnement de simulation réaliste.
La couche Cloud reçoit les tâches issues du système, puis les planifie en s'appuyant sur différentes politiques d'ordonnancement. Celles-ci comprennent les algorithmes classiques (FCFS,Min-Min et Max-Min), les algorithmes méta-heuristiques, notamment les algorithmes génétiques, ainsi que les méthodes basées sur le Reinforcement Learning telles que le Q-Learning et le DDQN. L'affectation des tâches aux VMs est assurée par le CloudBroker, qui sélectionne la VM la plus appropriée en fonction de l'algorithme utilisé et de l'état des ressources disponibles.
Globalement, la Cloud Layer vise à traiter les tâches lourdes nécessitant de grandes ressources, tout en simulant de manière réaliste les performances, la latence et la consommation énergétique des environnements cloud, afin de permettre une évaluation complète du système IoT distribué.

III.3.1.4 Broker de gestion :

Le Broker est un composant central dans l'architecture proposée. Il est chargé d'assurer la distribution intelligente des tâches générées par les dispositifs IoT vers la couche la plus appropriée. Son fonctionnement repose sur une logique de décision simple, basée sur deux critères principaux : la date limite de la tâche et sa taille. Une tâche est orientée vers la couche Fog si elle est considérée comme urgente, c'est-à-dire si sa deadline est inférieure ou égale à un seuil prédéfini, et si elle est également légère en termes de charge de traitement. Dans le cas contraire, la tâche est dirigée vers la Cloud Layer, qui dispose de ressources plus importantes, mais avec une latence plus élevée. Le Broker joue ainsi le rôle d'intermédiaire entre les objets IoT et les infrastructures de calcul, en assurant la répartition des tâches de manière efficace. Pour optimiser les performances globales du système, il regroupe les tâches dans des buffers distincts pour la couche Fog et la couche Cloud, puis les envoie par lots, ce qui

permet de réduire la surcharge et d'améliorer l'efficacité du traitement. Parallèlement, il collecte des statistiques sur la répartition des tâches afin d'analyser le comportement du système et d'évaluer la proportion de tâches traitées par chaque couche. Grâce à ce mécanisme, le broker contribue à réduire la latence, à respecter les délais d'exécution et à optimiser l'utilisation des ressources dans les environnements IoT distribués.

III.3.2. Implémentation du système proposé :

III.3.2.1 Présentation des algorithmes classiques d'ordonnancement :

Dans cette partie, nous présentons les principaux algorithmes classiques d'ordonnancement implémentés dans notre système. Ces algorithmes constituent des approches de référence largement utilisées dans les systèmes distribués pour l'ordonnancement des tâches . Elles servent de base de comparaison pour évaluer les performances des algorithmes avancées.

- **FCFS (First Come First Served) :** les tâches sont exécutées selon leur ordre d'arrivée, sans prise en compte des caractéristiques des ressources disponibles.
- **Min-Min :** le scheduler sélectionne, pour chaque tâche, le nœud offrant le plus faible temps d'exécution estimé, dans le but de minimiser le temps global de traitement.
- **Max-Min :** le scheduler privilégie les nœuds présentant les meilleurs temps d'exécution pour les tâches longues, afin d'éviter leur retard et améliorer leur prise en charge.

III.3.2.2 Présentation de l'algorithme génétique :

L'algorithme génétique (GA) est utilisé dans ce travail comme une méthode d'optimisation pour l'ordonnancement des tâches dans un environnement Fog/Cloud. Chaque solution représente une affectation des tâches aux nœuds de calcul, codée sous forme de chromosome. L'algorithme commence par générer une population initiale de solutions aléatoires, puis évalue leur qualité à l'aide d'une fonction de fitness prenant en compte le temps d'exécution global (makespan), la consommation énergétique, le retard des tâches et des pénalités en cas de contraintes non respectées. Les meilleures solutions sont ensuite sélectionnées et combinées à l'aide des opérateurs de crossover et de mutation afin de générer de nouvelles solutions.

Ce processus évolutif est répété sur plusieurs générations jusqu'à obtenir une solution optimale ou quasi optimale. L'objectif principal est d'améliorer la répartition des tâches tout en minimisant le temps d'exécution, l'énergie consommée et le taux de retard dans le système. Les paramètres de l' algorithme génétique sont présentés dans le tableau III.6:

Paramètre	**Valeur**
Taille de la population	10
Nombre de générations	8
Taux de mutation	0.4
Taille de l'élite	2
Probabilité de croisement	0.5

Tableau III.6 : Paramètres d'algorithme génétique

L'exemple ci-dessous illustre la représentation d'une solution. La figure III.2 présente un exemple d'affectation des tâches aux différentes ressources de calcul de l'environnement Cloud/Fog/Edge.

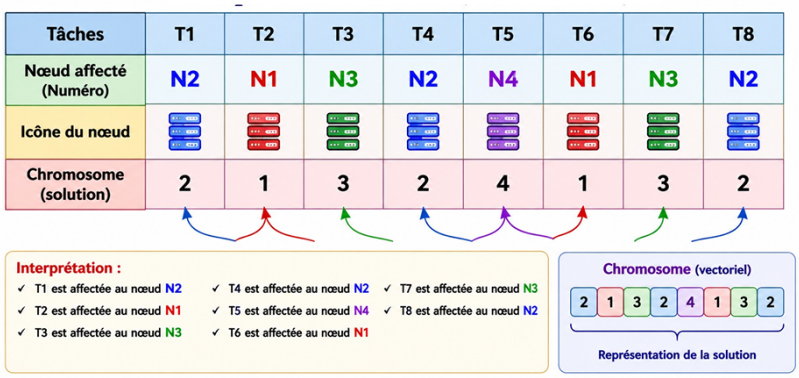


Figure III.2 : Représentation d'une solution .

III.3.2.3 Présentation de l'algorithme Q-Learning :

III.3.2.3.1 Espace d'état (State Space) :

L'état du système est représenté sous une forme discrète afin de simplifier et d'améliorer le processus d'apprentissage. Il regroupe quatre informations principales décrivant la situation actuelle du système : l'état du processeur (occupé ou libre), le niveau de mémoire RAM

disponible, la taille de la file d'attente ainsi que la taille de la tâche à exécuter. Cette représentation permet à l'agent de percevoir l'état global du système à un instant donné et de prendre des décisions adaptées.

L'exemple ci-dessous illustre la représentation d'un état. La figure III.3 présente un exemple d'état décrivant la situation du système à un instant donné.

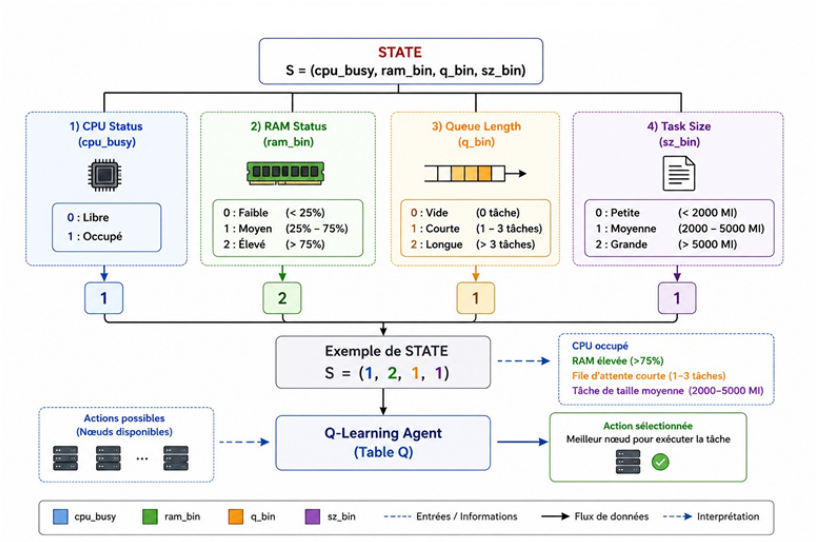


Figure III.3 : Représentation d'un state de Q-Learning.

III.3.2.3.2 Espace d'actions (Action Space) :

L'espace d'actions correspond à l'ensemble des nœuds disponibles dans l'environnement, qu'il s'agisse de Node dans le Fog ou de VM dans le Cloud. Chaque action représente le choix d'un VM/node spécifique vers lequel une tâche sera envoyée pour son exécution. Ainsi, l'agent apprend à sélectionner, pour chaque état, le VM/node le plus approprié afin d'optimiser les performances globales du système.

III.3.2.3.3 Fonction de récompense (Reward Function) :

La fonction de récompense est conçue pour guider l'apprentissage vers des décisions optimales. Elle prend en compte plusieurs critères de performance, notamment le temps d'exécution, la consommation énergétique et le retard des tâches. Une pénalité importante est appliquée lorsque les contraintes de délai ne sont pas respectées. Ainsi, l'agent est encouragé à privilégier les décisions qui améliorent l'efficacité globale du système tout en respectant les contraintes temporelles.

Les paramètres de l' algorithme **Q-Learning** sont présentés dans le tableau III.7:

Paramètre	**Valeur**
Taux d'apprentissage (Learning rate (α))	0.12
Facteur de réduction (Discount factor (γ))	0.92
Taux d'exploration initial (Epsilon initial (ϵ))	1.0
Décroissance de l'exploration	0.95
Taux d'exploration minimal	0.05

Tableau III.7 : Paramètres d'algorithme du Q-Learning

III.3.2.3.4 Mise à jour des Q-values (Q-Learning) :

Dans notre algorithme, la mise à jour de la Q-value est réalisée selon la règle de Q-Learning :

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

où :

- $Q(s,a)$: valeur Q actuelle de l'état s et de l'action a ;
- α : taux d'apprentissage ;
- r : récompense obtenue ;
- γ : facteur de réduction ;
- s' : état suivant ;
- $Q(s',a')$: valeur future estimée ;

III.3.2.4 Présentation de l'algorithme DDQN :

Nous présentons les composants principaux du modèle DDQN. Le problème d'ordonnancement est modélisé comme un processus de prise de décision séquentielle, dans lequel

un agent intelligent apprend, à travers ses interactions avec l'environnement, à sélectionner le VM/Node le plus approprié (Fog ou Cloud) pour l'exécution des tâches. L'objectif est d'optimiser les différents indicateurs de performance du système, tels que le temps d'exécution, la consommation énergétique et le respect des contraintes temporelles (deadlines), afin d'améliorer l'efficacité globale du système.

III.3.2.4.1 Espace d'état (State Space) :

L'espace d'état représente l'ensemble des informations observées par l'agent DDQN à chaque instant afin de prendre une décision d'ordonnancement appropriée. Dans notre système basé sur l'architecture Fog/Cloud IoT, chaque état correspond à la configuration actuelle des nœuds disponibles ainsi qu'aux caractéristiques de la tâche à exécuter.

Le scheduler est un composant essentiel chargé de collecter un ensemble de paramètres liés aux ressources du système. Ces paramètres incluent notamment le taux d'utilisation du processeur, la mémoire disponible, la longueur de la file d'attente, la capacité de calcul, le nombre de cœurs CPU ainsi que la consommation énergétique. Les informations relatives à la tâche, telles que sa taille et sa deadline, sont également intégrées dans l'état.

Ces caractéristiques sont normalisées afin d'améliorer la stabilité de l'apprentissage du réseau de neurones. Ainsi, l'état global du système est obtenu par la concaténation des vecteurs de caractéristiques de l'ensemble des nœuds Fog et Cloud disponibles. Grâce à cette ordonnancement , l'agent DDQN dispose d'une vision complète de l'environnement, ce qui lui permet de sélectionner le VM/Node le plus approprié pour optimiser les performances du système.

III.3.2.4.2 Espace d'actions (Action Space) :

L'espace des actions représente l'ensemble des décisions que l'agent DDQN peut effectuer durant le processus d'ordonnancement. Dans notre système Fog/Cloud IoT, une action correspond au choix d'un VM/Node de calcul capable d'exécuter la tâche reçue. Ces nodes peuvent être des Fog Nodes situés à proximité des appareils IoT ou des VMs dans le Cloud. À chaque arrivée d'une tâche, le scheduler DDQN analyse l'état actuel de l'environnement puis sélectionne une action parmi les VMs/nodes disponibles. Le choix est réalisé selon la politique ε-greedy, qui combine exploration et exploitation afin d'améliorer progressivement les décisions d'ordonnancement. Ainsi, l'espace des actions est défini comme l'ensemble des VMs/nodes accessibles dans le système, où chaque action représente l'affectation de la tâche

à un VM/ node spécifique pour son exécution.La figure III.4 présente l'espace des actions du modèle DDQN, où chaque action correspond à l'affectation d'une tâche à une ressource de calcul (VM ou nœud Fog) disponible dans l'environnement.

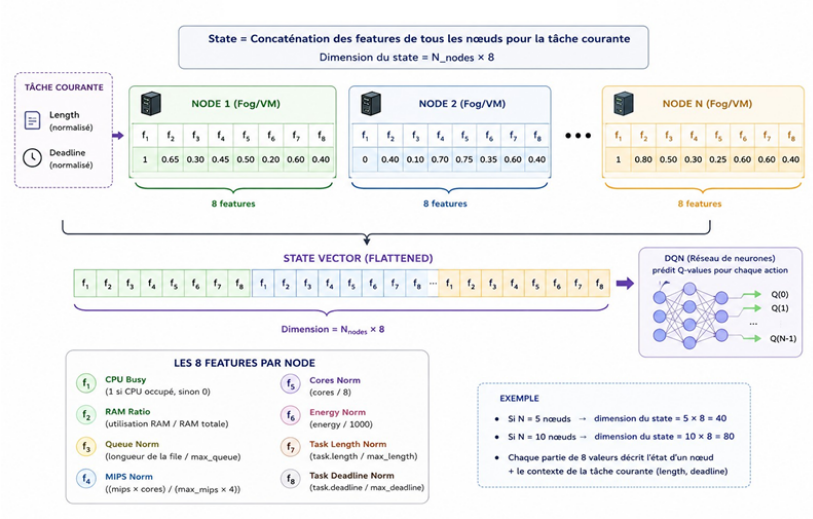


Figure III.4 : Représentation d'un state de DDQN.

III.3.2.4.3 Fonction de récompense (Reward Function) : La récompense mesure la qualité de la décision prise par le scheduler.

L'objectif est de :

- réduire le temps d'exécution,
- minimiser la consommation énergétique,
- diminuer le délai,

III.3.2.4.4 Politique d'apprentissage :

Le scheduler utilise la stratégie ε-greedy.

III.3.2.4.5 Exploration :

L'agent choisit une action aléatoire afin de découvrir de nouvelles solutions.

III.3.2.4.6 Exploitation :

L'agent sélectionne l'action ayant la meilleure Q-value.

III.3.2.4.7 Mise à jour des Q-values :

Le modèle DDQN utilise l'équation suivante :

$$Q(s,a) = r + \gamma\, Q_{\text{target}}\,(s', \text{argmax}_{a'}\, Q_{\text{online}}(s', a'))$$

où :

- $Q(s,a)$: valeur de l'action,
- r : récompense (Reward),
- γ : facteur de réduction,
- s' : nouvel état,
- $Q_{\text{online}}(s,a)$: réseau online.

Les paramètres de l'algorithme DDQN sont présentés dans le tableau III.8:

Paramètre	**Valeur**
Learning Rate	0.003
Gamma	0.97
Epsilon Max	1.0
Epsilon Min	0.05
Epsilon Decay	0.97
Coefficient de mise à jour douce $(\tau\dot{\epsilon})$	0.005
Batch Size	64
Buffer Size	10000
Gradient Clip	1.0
Hidden Layer 1	256 Neurones
Hidden Layer 2	128 Neurones
Hidden Layer 3	64 Neurones
Activation Function	ReLU
Optimizer	Adam
Loss Function	Huber Loss
Replay Method	Experience Replay
DQN Type	Double DQN

Tableau III.8 : Paramètres de l'algorithme DDQN

III.3.2.5 Métriques de performance :

III.3.2.5.1 Temps d'exécution :

Dans ce travail, le temps d'exécution d'une tâche est estimé en fonction de sa taille (length de la tâche) et de la capacité de traitement de VM/node. Nous supposons que la VM/node exécute la tâche de manière parallèle en utilisant l'ensemble de ses cœurs disponibles.

$$T_{\text{exec}} = \frac{\text{Length}_{\text{task}}}{\text{MIPS} \times \text{Cores}}$$

où :

- $\text{Length}_{\text{task}}$: représente la charge de calcul de la tâche (en Million Instructions),
- MIPS : est la capacité de traitement d'un seul cœur (Million Instructions Par Second),
- Cores : désigne le nombre de cœurs disponibles dans la VM/node.

Ce modèle reflète l'effet du parallélisme, puisque l'augmentation du nombre de cœurs ou de la puissance de calcul (MIPS) permet de réduire le temps d'exécution. Il permet ainsi de simuler de manière simple mais réaliste le comportement des systèmes multi-cœurs dans un environnement Fog/Cloud.

III.3.2.5.2 Makespan :

Il correspond au temps de terminaison de la dernière tâche exécutée dans le système.

$$\text{Makespan} = \max(\text{Finish Time})$$

III.3.2.5.3 Consommation énergétique : La consommation énergétique est modélisée au niveau des VMs ou des nœuds en fonction de leur état d'exécution. Contrairement à un modèle où la consommation est continue, nous considérons que l'énergie est principalement consommée lorsque la VM /node exécute une tâche.
Ainsi, une VM/Node consomme de l'énergie uniquement pendant la période d'exécution des tâches qui lui sont affectées. Lorsque la VM/node est inactive (aucune tâche en cours), elle est considérée en état idle, et sa consommation énergétique est supposée minimale.
L'énergie consommée par une VM/node est donc calculée uniquement pendant les périodes d'activité, selon la relation :

$$E = P \times t$$

où :

- **P** : représente la puissance consommée par la VM/Node pendant l'exécution,

- **t** :le temps total d'exécution des tâches.

La puissance dépend de l'état de la VM et pour les nœuds de fog :

- **u = 1** lorsque la VM/Node exécute une tâche (busy state)
- **u = 0** lorsque la VM/Node est inactive (idle state)

La puissance est modélisée comme suit :

$$P(u) = P_{\text{idle}} + (P_{\text{busy}} - P_{\text{idle}}) \times u$$

Cependant, dans ce modèle, l'énergie est principalement accumulée uniquement pendant les périodes d'exécution des tâches, ce qui signifie que la consommation totale est la somme des énergies générées par chaque tâche exécutée sur la VM/Node.

III.3.2.5.4 Tâches en retard :

Les tâches en retard sont utilisées comme une métrique importante pour évaluer la qualité de service (QoS) et les performances de l'algorithme d'ordonnancement dans l'environnement Fog/Cloud.
Chaque tâche possède une contrainte temporelle appelée *deadline*, représentant le temps maximal autorisé pour terminer son exécution. Après l'exécution, le système calcule le temps total passé par la tâche dans le système, appelé *turnaround time*.
Le ***turnaround time*** est calculé comme suit :

$$T_{\text{turnaround}} = T_{\text{finish}} - T_{\text{arrival}}$$

où :

- T_{finish} : représente le temps de fin d'exécution de la tâche,
- T_{arrival} : représente le temps d'arrivée de la tâche dans le système.

Ensuite, le temps de rotation est comparé à la *deadline* de la tâche :

$$\text{Status} = \begin{cases} \text{Taux de rejet} & \text{si } T_{\text{turnaround}} > \text{Deadline} \\ \text{Taux de réussite} & \text{sinon} \end{cases}$$

- Si le temps de rotation dépasse la *deadline*, la tâche est considérée comme retardée (rejet).
- Sinon, elle est considérée comme exécutée dans les délais (réussite).

Cette modélisation permet d'évaluer la capacité du système à respecter les contraintes temporelles des applications IoT et à minimiser le nombre de tâches retardées.

III.3.2.6 Analyse des résultats :

Dans cette partie, nous procédons à la présentation et à l'analyse des résultats issus des simulations effectuées selon différents scénarios de charge (100, 200, 300, 500 tâches). Une étude comparative est menée entre plusieurs algorithmes d'ordonnancement, notamment les algorithmes classiques (FCFS,Min-Min et Max-Min), l'algorithme méta-heuristique (GA) ainsi que les algorithmes basées sur le Reinforcement Learning (Q-Learning et DDQN).
Cette analyse a pour objectif d'évaluer les performances de ces algorithmes en termes de taux de complétion des tâches, de respect des délais, de makespan et de consommation énergétique, en fonction de la variation du nombre de tâches.

III.3.2.6.1 Analyse des performances des algorithmes pour 100 tâches :

Les résultats de simulation, obtenus à partir d'un ensemble de 100 tâches générées aléatoirement, sont présentés ci-après afin d'analyser et d'évaluer les performances des différents algorithmes d'ordonnancement.

ALGO	Taux de réussite (%)	Taux de rejet (%)	Taux à temps (%)	Taux en retard (%)	MAKESPAN (S)	Consommation d'énergie (J)
FCFS	100.00	0.00	67.33	32.67	30.30	3533.86
Min-Min	100.00	0.00	90.33	9.67	9.47	2069.96
Max-Min	100.00	0.00	66.33	33.67	75.42	5834.45
GA	100.00	0.00	91.67	8.33	8.77	2069.64
Q-Learning	100.00	0.00	95.67	4.33	6.36	2715.66
DDQN	100.00	0.00	98.33	1.67	4.82	2301.99

Tableau III.9 : Résultats expérimentaux obtenus pour 100 tâches.

Analyse des résultats

Les résultats présentés dans le Tableau III.9, obtenus pour 100 tâches, permettent d'évaluer les performances des différents algorithmes d'ordonnancement dans un environnement Cloud/-Fog. Tous les algorithmes affichent un taux de réussite de 100 % sans rejet de tâches, ce qui confirme la capacité du système à traiter l'ensemble de la charge. L'analyse des performances temporelles montre une nette supériorité des approches basées sur l'apprentissage par renforcement, en particulier DDQN avec 98,33 % de tâches exécutées à temps et un makespan minimal de 4,82 s, suivi par Q-Learning (95,67 % et 6,36 s). Les algorithmes heuristiques tels que GA et Min-Min présentent des performances intermédiaires, tandis que FCFS et Max-Min obtiennent les résultats les plus faibles, notamment en termes de taux de tâches à temps et de makespan. Sur le plan énergétique, GA et Min-Min offrent les meilleures consommations ($\approx$ 2069 J), alors que Max-Min est le plus énergivore.

Discussion des résultats

La discussion des résultats présentés dans le Tableau III.9 montre que les algorithmes basés sur l'apprentissage par renforcement, notamment DDQN et Q-Learning, parviennent à établir un meilleur équilibre entre plusieurs objectifs de performance. En effet, ces approches assurent simultanément un taux élevé de tâches exécutées dans les délais, une réduction significative du makespan et une consommation énergétique raisonnable. DDQN se distingue particulièrement en offrant le meilleur compromis global entre qualité de service et efficacité énergétique, tandis que Q-Learning obtient également des résultats très satisfaisants. À l'inverse, les algorithmes heuristiques tels que GA et Min-Min privilégient davantage l'optimisation énergétique au détriment des performances temporelles, alors que FCFS et Max-Min présentent des limitations importantes sur la plupart des indicateurs évalués. Ces résultats confirment ainsi l'efficacité des méthodes d'apprentissage par renforcement pour résoudre des problèmes d'ordonnancement multi-objectifs dans les environnements Cloud/Fog modernes.

III.3.2.6.2 Analyse des performances des algorithmes pour 200 tâches :

Les résultats de simulation, obtenus à partir d'un ensemble de 200 tâches générées aléatoirement, sont présentés ci-après afin d'analyser et d'évaluer les performances des différents algorithmes d'ordonnancement.

ALGO	Taux de réussite (%)	Taux de rejet (%)	Taux à temps (%)	Taux en retard (%)	MAKESPAN (S)	Consommation d'énergie (J)
FCFS	100.00	0.00	63.17	36.38	56.38	7174.15
Min-Min	100.00	0.00	89.67	10.33	17.63	4150.83
Max-Min	100.00	0.00	67.00	33.00	147.96	11429.92
GA	100.00	0.00	91.00	9.00	16.27	4151.23
Q-Learning	100.00	0.00	97.50	2.50	11.11	5145.81
DDQN	100.00	0.00	99.33	0.67	9.66	4662.78

Tableau III.10 : Résultats expérimentaux obtenus pour 200 tâches.

Analyse des résultats

Les résultats présentés dans le Tableau III.10, obtenus pour un ensemble de 200 tâches, permettent d'évaluer le comportement des différents algorithmes face à une charge de travail plus importante. Tous les algorithmes conservent un taux de réussite de 100 % et un taux de rejet nul, ce qui démontre leur capacité à traiter l'ensemble des tâches générées. Cependant, des différences significatives apparaissent au niveau des performances temporelles. L'algorithme DDQN obtient les meilleurs résultats avec un taux de tâches exécutées à temps de 99,33 % et un *makespan* minimal de 9,66 s, suivi par Q-Learning avec 97,50 % de tâches à temps et un *makespan* de 11,11 s. Les algorithmes GA et Min-Min présentent des performances intermédiaires avec respectivement 91,00 % et 89,67 % de tâches exécutées à temps. À l'inverse, FCFS et Max-Min affichent les taux de retard les plus élevés ainsi que des *makespans* nettement supérieurs.En termes de consommation énergétique,Min-Min et GA demeurent les plus économes ($\approx$ 4150 J), tandis que Max-Min présente la consommation la plus élevée avec 11429,92 J.

Discussion des résultats

L'augmentation du nombre de tâches de 100 à 200 met davantage en évidence les différences de performance entre les approches étudiées. Les résultats montrent que les méthodes basées sur l'apprentissage par renforcement, notamment DDQN, conservent une excellente capacité d'adaptation même lorsque la charge du système augmente. Le taux élevé de tâches exécutées à temps et la réduction du *makespan* indiquent une meilleure utilisation des ressources Cloud/Fog ainsi qu'une prise de décision plus efficace lors de l'ordonnancement. Bien que la consommation énergétique de DDQN soit légèrement supérieure à celle de GA et Min-Min, elle reste raisonnable au regard des gains obtenus en matière de délai d'exécution. À l'inverse, les algorithmes traditionnels FCFS et Max-Min voient leurs performances se dégrader

fortement lorsque la charge augmente, ce qui confirme leurs limites dans les environnements distribués. Ces résultats soulignent ainsi l'intérêt des techniques d'intelligence artificielle, en particulier du *Deep Reinforcement Learning*, pour répondre aux exigences de performance des environnements de *Cloud/Fog Computing*.

III.3.2.6.3 Analyse des performances des algorithmes pour 300 tâches :

Les résultats de simulation, obtenus à partir d'un ensemble de 300 tâches générées aléatoirement, sont présentés ci-après afin d'analyser et d'évaluer les performances des différents algorithmes d'ordonnancement.

ALGO	Taux de réussite (%)	Taux de rejet (%)	Taux à temps (%)	Taux en retard (%)	MAKESPAN (S)	Consommation d'énergie (J)
FCFS	100.00	0.00	65.22	34.78	85.59	10447.75
Min-Min	100.00	0.00	85.33	14.67	26.75	6079.79
Max-Min	100.00	0.00	67.33	32.67	218.55	16900.90
GA	100.00	0.00	87.56	12.44	24.70	6082
Q-Learning	100.00	0.00	97.11	2.89	15.56	7924.58
DDQN	100.00	0.00	98.00	2.00	14.90	6746.94

Tableau III.11 : Résultats expérimentaux obtenus pour 300 tâches.

Analyse des résultats

Les résultats présentés dans le Tableau III.11 illustrent les performances des différentes stratégies d'ordonnancement pour une charge de travail plus importante, composée de 300 tâches. Tous les algorithmes maintiennent un taux de réussite de 100 % et un taux de rejet nul, démontrant leur capacité à traiter l'ensemble des tâches soumises. Toutefois, des écarts significatifs apparaissent au niveau des indicateurs de performance. L'algorithme DDQN obtient les meilleurs résultats avec un taux de tâches exécutées à temps de 98,00 % et le plus faible *makespan* (14,90 s), suivi de près par Q-Learning avec 97,11 % de tâches à temps et un *makespan* de 15,56 s. Les algorithmes GA et Min-Min affichent des performances intermédiaires avec respectivement 87,56 % et 85,33 % de tâches exécutées dans les délais.En revanche, FCFS et Max-Min présentent les taux de retard les plus élevés ainsi que des *makespans* nettement plus importants, atteignant respectivement 85,59 s et 218,55 s. Concernant la consommation énergétique, Min-Min et GA demeurent les solutions les plus économes ($\approx$ 6080 J), tandis que Max-Min affiche la consommation la plus élevée avec 16900,90 J

Discussion des résultats

L'augmentation de la charge de travail à 300 tâches confirme les tendances observées pour les scénarios précédents et met davantage en évidence la robustesse des approches intelligentes. Les algorithmes basés sur l'apprentissage par renforcement, notamment DDQN et Q-Learning, conservent des performances élevées malgré la croissance du nombre de tâches, ce qui traduit leur capacité à adapter dynamiquement les décisions d'ordonnancement aux conditions du système. DDQN se distingue particulièrement en offrant le meilleur compromis entre respect des délais, réduction du *makespan* et maîtrise de la consommation énergétique. Bien que GA et Min-Min soient plus efficaces sur le plan énergétique, leurs performances temporelles restent inférieures à celles des méthodes d'apprentissage. À l'inverse, les approches traditionnelles FCFS et Max-Min montrent une forte dégradation du temps d'exécution et de la consommation énergétique lorsque la charge augmente, ce qui limite leur efficacité dans des environnements Cloud/Fog à grande échelle. Ces résultats confirment ainsi que les algorithmes de *Reinforcement Learning* constituent une solution particulièrement adaptée pour optimiser l'ordonnancement dans des systèmes distribués modernes soumis à des charges importantes.

III.3.2.6.4 Analyse des performances des algorithmes pour 500 tâches :

Les résultats de simulation, obtenus à partir d'un ensemble de 500 tâches générées aléatoirement, sont présentés ci-après afin d'analyser et d'évaluer les performances des différents algorithmes d'ordonnancement.

ALGO	Taux de réussite (%)	Taux de rejet (%)	Taux à temps (%)	Taux en retard (%)	MAKESPAN (S)	Consommation d'énergie (J)
FCFS	100.00	0.00	65.20	34.80	142.9	17386.25
Min-Min	100.00	0.00	79.33	20.67	44.67	10122.50
Max-Min	100.00	0.00	65.67	34.33	336.62	27899.18
GA	100.00	0.00	79.53	20.47	40.67	10125.32
Q-Learning	100.00	0.00	86.13	13.87	33.05	13646.64
DDQN	100.00	0.00	80.93	19.07	44.36	13135.00

Tableau III.12 : Résultats expérimentaux obtenus pour 500 tâches.

Analyse des résultats

Les résultats présentés dans le Tableau III.12 , obtenus pour un ensemble de 500 tâches, permettent d'évaluer les performances des différents algorithmes d'ordonnancement dans un scénario de forte charge. Tous les algorithmes maintiennent un taux de réussite de 100 % et

un taux de rejet nul, ce qui confirme leur capacité à traiter l'ensemble des tâches générées. Toutefois, des différences importantes apparaissent au niveau des performances temporelles. Les algorithmes Q-Learning et DDQN présentent les meilleures performances avec respectivement 86,13 % et 80,93 % de tâches exécutées à temps, ainsi que des valeurs de makespan relativement faibles (33,05 s et 44,36 s). Les approches heuristiques GA et Min-Min affichent des résultats intermédiaires avec 79,58 % et 79,33 % de tâches dans les délais. En revanche, FCFS et surtout Max-Min présentent les performances les plus faibles avec des taux élevés de retard et des makespans très importants, atteignant 142,9 s et 336,62 s. Concernant la consommation énergétique, Min-Min et GA restent les plus efficaces ($\approx$ 10122.50 J), tandis que Max-Min enregistre la consommation la plus élevée ((27899.18 J).

Discussion des résultats

La discussion des résultats montre que les algorithmes basés sur l'apprentissage par renforcement, notamment Q-Learning et DDQN, parviennent à maintenir un bon équilibre entre plusieurs objectifs de performance. En effet, ces approches assurent simultanément une bonne qualité de service en termes de respect des délais, une réduction du makespan, ainsi qu'une consommation énergétique relativement maîtrisée, ce qui confirme leur capacité à optimiser plusieurs critères de manière conjointe dans des environnements Cloud/Fog complexes et fortement sollicités. En revanche, les algorithmes heuristiques tels que GA et Min-Min présentent une bonne efficacité, en particulier sur le plan énergétique, mais restent moins performants en termes de compromis global entre les différents objectifs. Enfin, l'algorithme classique FCFS affiche les performances les plus faibles, notamment en termes de makespan et de respect des délais, ce qui limite son efficacité dans les systèmes distribués modernes.

III.3.2.6.5 Consommation d'énergie selon le nombre de tâches

Dans le but d'évaluer l'efficacité énergétique des différentes stratégies d'ordonnancement, nous avons analysé l'évolution de la consommation d'énergie en fonction du nombre de tâches traitées. Le graphique suivant illustre la quantité totale d'énergie consommée par chaque algorithme pour différentes charges de travail dans l'environnement Cloud/Fog/IoT.

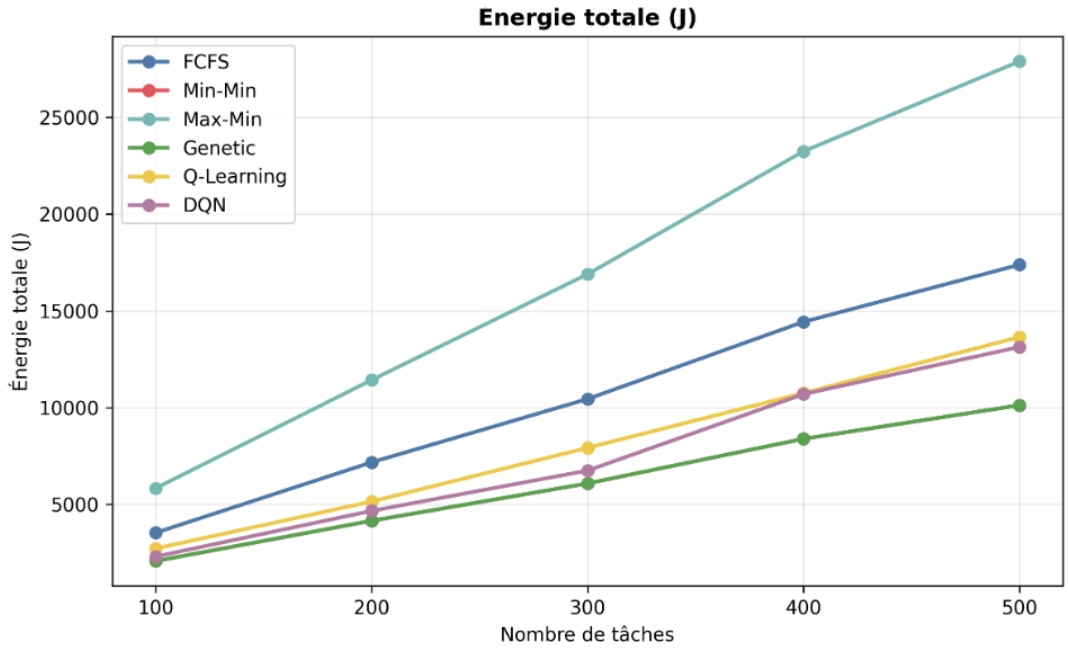


Figure III.5 : Énergie totale consommée en fonction du nombre de tâches.

La Figure III.5 illustre l'évolution de la consommation énergétique totale (en Joules) des différents algorithmes d'ordonnancement en fonction du nombre de tâches (de 100 à 500). Les résultats montrent une augmentation progressive et quasi linéaire de l'énergie consommée pour l'ensemble des méthodes. Toutefois, des écarts significatifs apparaissent entre les algorithmes : les approches basées sur l'intelligence artificielle et l'optimisation (notamment l'algorithme génétique, le DDQN et le Q-Learning) présentent les meilleures performances énergétiques, tandis que les méthodes classiques comme FCFS et Max-Min affichent une consommation nettement plus élevée.

III.3.2.6.6 Makespan selon le nombre de tâches :

Afin d'évaluer l'efficacité des différentes stratégies d'ordonnancement en termes de temps d'exécution global, nous avons analysé l'évolution du Makespan en fonction du nombre de tâches à traiter. Le graphique suivant présente les performances des algorithmes étudiés pour différentes charges de travail. Cette métrique permet de mesurer la durée totale nécessaire à l'exécution de l'ensemble des tâches soumises au système.

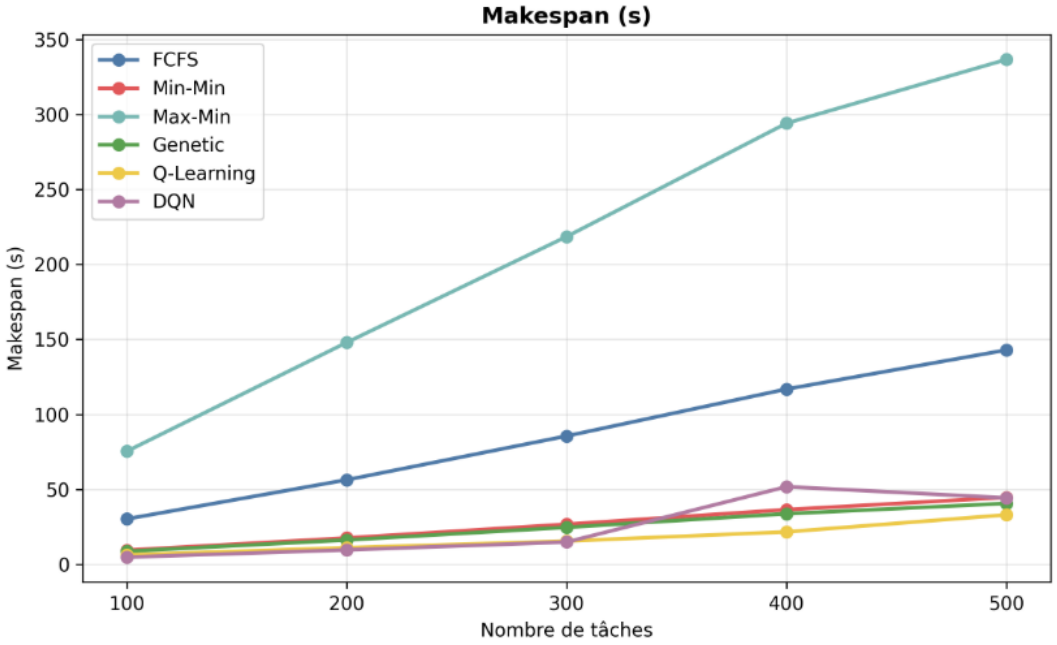


Figure III.6 : l'évolution de Makespanen en fonction du nombre de tâches.

La Figure III.6 présente l'évolution du Makespan (temps total d'exécution en secondes) en fonction du nombre de tâches (100 à 500) pour différents algorithmes d'ordonnancement. Les résultats montrent une augmentation globale du temps d'exécution avec la charge de travail, mais avec des écarts importants entre les méthodes. Les algorithmes traditionnels comme FCFS et Max-Min affichent les plus mauvaises performances avec des temps élevés, tandis que les approches optimisées, notamment Q-Learning, DDQN, Min-Min et l'algorithme génétique, permettent de réduire significativement le Makespan. Parmi eux, DDQN, Q-Learning se distinguent comme les algorithmes les plus efficaces en termes de rapidité, suivi de près par Min-Min et l'algorithme génétique, ce qui confirme l'intérêt des approches intelligentes pour améliorer les performances temporelles des systèmes d'ordonnancement.

III.3.2.6.7 Respect des délais selon le nombre de tâches :

Le respect des délais constitue un critère essentiel dans les systèmes IoT à contraintes temporelles strictes. Afin d'évaluer la capacité des algorithmes à exécuter les tâches dans les délais impartis, nous avons analysé le pourcentage de tâches terminées à temps pour différentes charges de travail. Le graphique suivant présente l'évolution de cette métrique en fonction du nombre de tâches.

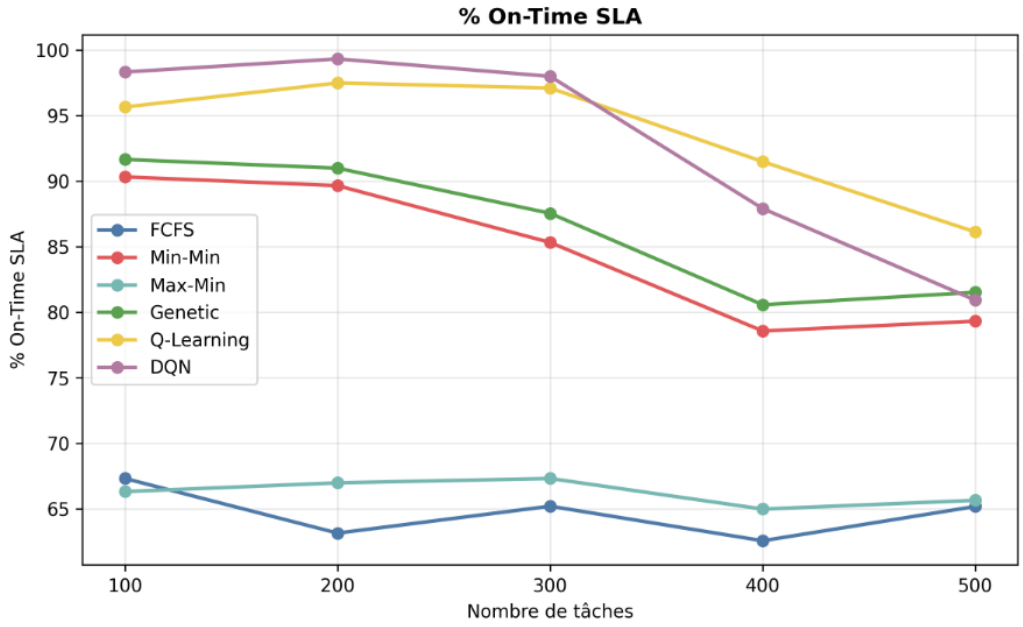


Figure III.7 : pourcentage des tâches livrées a temps en fonction du nombre de tâches.

La Figure III.7 illustre l'évolution du pourcentage de respect des délais (On-Time) en fonction du nombre de tâches (de 100 à 500) pour différents algorithmes d'ordonnancement. Les résultats montrent une diminution globale du taux de réussite à mesure que la charge de travail augmente, ce qui reflète la difficulté croissante à respecter les délais dans des environnements fortement sollicités. Les algorithmes FCFS et Max-Min présentent les performances les plus faibles, tandis que les approches basées sur l'intelligence artificielle, notamment Q-Learning et DDQN, atteignent des taux de réussite très élevés pour les charges faibles et moyennes. Cependant, Q-Learning et DDQN se distinguent par leur stabilité et leur robustesse à grande échelle.

III.3.2.6.8 Synthèse :

Les résultats obtenus dans ce travail montrent que les algorithmes basés sur le Reinforcement Learning (Q-Learning et DDQN) offrent une meilleure optimisation globale du système Fog/-Cloud par rapport aux méthodes classiques et heuristiques.

L'objectif principal de ces algorithmes est de trouver un équilibre entre plusieurs métriques de performance, notamment la réduction du makespan, l'amélioration de la consommation énergétique, et le respect des délais. Grâce à leur capacité d'apprentissage et d'adaptation, ils prennent des décisions dynamiques en fonction de l'état du système.

Contrairement aux approches classiques (FCFS,Min-Min, Max-Min) qui suivent des règles

fixes, et à l'algorithme génétique qui reste coûteux et parfois instable, les méthodes de Reinforcement Learning apprennent progressivement à améliorer la qualité des décisions.
Ainsi, le DDQN et le Q-Learning permettent un meilleur ordonnancement des tâches, en assurant une réduction du temps d'exécution, une meilleure efficacité énergétique et un respect plus important des contraintes de délai.
En conclusion, les approches basées sur le Reinforcement Learning se distinguent comme les solutions les plus adaptées pour les environnements IoT dynamiques et complexes.

III.4. Conclusion :

Ce chapitre a présenté l'implémentation et la simulation du système proposé pour l'ordonnancement des tâches dans un environnement Cloud/Fog/IoT. Nous avons décrit l'architecture générale du système ainsi que les différentes stratégies d'ordonnancement implémentées, incluant les algorithmes heuristiques, l'algorithme métaheuristique Genetic Algorithm (GA) et les algorithmes intelligents basés sur le Reinforcement Learning, à savoir le Q-Learning et le DDQN.
Par la suite, les paramètres de simulation et les métriques de performance utilisées pour l'évaluation ont été définis. Une étude comparative a ensuite été menée à travers plusieurs scénarios de simulation afin d'évaluer les performances des différents algorithmes en termes de consommation énergétique, de Makespan et de respect des délais d'exécution.
Les résultats obtenus montrent que les approches intelligentes basées sur le Reinforcement Learning, notamment le Q-Learning et le DDQN, offrent de meilleures performances que les méthodes classiques. Grâce à leur capacité d'adaptation et d'optimisation dynamique, ces approches permettent de réduire la consommation énergétique, d'améliorer le Makespan et de mieux respecter les contraintes temporelles dans des environnements Cloud/Fog hétérogènes et distribués.

Conclusion générale

CONCLUSION GÉNÉRALE

Le Cloud et le Fog Computing sont aujourd'hui des technologies indispensables pour répondre à la croissance des besoins des applications IoT et des systèmes distribués modernes. Grâce à leurs capacités de traitement, de stockage et de gestion des ressources, ces environnements permettent de supporter un grand nombre d'applications nécessitant flexibilité, évolutivité et rapidité d'exécution. Cependant, l'augmentation continue du volume des données et du nombre de tâches engendre plusieurs défis majeurs, notamment la consommation énergétique élevée, la latence et la gestion efficace des ressources.

Dans le cadre de notre mémoire, nous avons mené une étude sur l'ordonnancement des tâches IoT dans un environnement Fog et Cloud Computing. Notre objectif principal était l'optimisation de la consommation énergétique et du temps d'exécution, tout en garantissant le respect des délais et une qualité de service satisfaisante. Pour cela, nous avons étudié les principaux algorithmes d'ordonnancement, incluant les algorithmes heuristiques (FCFS,Min-Min et Max-Min), les algorithmes métaheuristiques (GA) et les algorithmes basés sur le Reinforcement Learning (Q-Learning et DDQN). Ensuite, nous avons développé et simulé plusieurs algorithmes avec SimPy et Python afin d'évaluer leurs performances dans divers scénarios de charge de travail. Les résultats obtenus montrent que les algorithmes intelligents basés sur le Reinforcement Learning (Q-Learning et DDQN) permettent d'obtenir un meilleur équilibre entre la consommation énergétique, le makespan et l'exécution des tâches dans les délais imposés. Grâce à leurs capacités d'apprentissage et d'adaptation dynamique, ces algorithmes améliorent significativement l'efficacité globale du système par rapport aux approches classiques. En particulier, le DDQN a fait la preuve de ses performances dans les environnements complexes et hétérogènes caractérisés par un grand nombre de tâches et de ressources distribuées. Cette démarche favorise une prise de décision plus éclairée et une optimisation dynamique des ressources.

Néanmoins, ce travail présente certaines limites. Tout d'abord, la mobilité des nœuds n'a pas été prise en compte dans les simulations, alors qu'elle constitue une caractéristique importante des environnements IoT réels. De plus, les expériences ont été réalisées avec un nombre limité de VMs et de nœuds Fog, ce qui restreint l'analyse de la scalabilité des approches proposées dans des environnements à grande échelle.

Enfin, le modèle de consommation énergétique repose sur des estimations approximatives basées sur l'utilisation des ressources, sans mesure réelle de la consommation matérielle, ce qui peut affecter la précision des résultats. Comme perspectives futures, il serait intéressant d'intégrer la mobilité des nœuds afin de mieux représenter les scénarios IoT réels. Il

serait également pertinent de tester les algorithmes sur des infrastructures plus larges et plus complexes afin d'étudier leur capacité de passage à l'échelle (scalability). Enfin, l'utilisation de modèles énergétiques plus réalistes permettrait d'obtenir une évaluation plus précise de la consommation énergétique et de valider davantage les performances des approches proposées.

RÉFÉRENCES BIBLIOGRAPHIQUES

Bibliographie

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, Jan. 2009, doi : 10.1145/1496091.1496100.

[2] G. Reese, *Cloud Application Architectures : Building Applications and Infrastructure in the Cloud*, 1st ed. Sebastopol, CA, USA : O'Reilly Media, 2009.

[3] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," National Institute of Standards and Technology, Gaithersburg, MD, USA, Spec. Publ. 800-145, Sep. 2011.

[4] A. Bouzara, "Sécurité dans le Cloud Computing," Ph.D. dissertation, Univ. Ibn Khaldoun, Tiaret, Algeria, 2018.

[5] J.-P. Figer, "Cloud computing — informatique en nuage," *Gestion de contenus numériques*, [Online], 2012.

[6] W. Voorsluys, J. Broberg, and R. Buyya, "Introduction to cloud computing," in *Cloud Computing : Principles and Paradigms*, Hoboken, NJ, USA : Wiley, 2011, pp. 1–41.

[7] BAP Software, "What is Cloud Computing ? Definition, Benefits, and Types," BAP Software Knowledge Base, 2024. [Online]. Available : `https://bap-software.net/en/knowledge/what-is-cloud-computing/`. [Accessed : Feb. 26, 2026].

[8] S. Singh and I. Chana, "A survey on resource scheduling in cloud computing : Issues and challenges," *J. Grid Comput.*, vol. 14, no. 2, pp. 217–264, Jun. 2016.

[9] N. M. Lucas, V. Kherbache, M. Mohamed, K. Yannick, and L. Allan, "Cloud Computing," Course Material, IUT Nancy-Charlemagne, Licence Professionnelle : Administration de systèmes, réseaux et applications à base de logiciels libres, p. 39, n.d.

[10] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing : State-of-the-art and research challenges," *J. Internet Serv. Appl.*, vol. 1, no. 1, pp. 7–18, May 2010, doi : 10.1007/s13174-010-0007-6.

[11] A. Gupta, S. Bandyopadhyay, and S. S. Thakur, "Cloud computing : Its characteristics, security issues and challenges," *Rev. Comput. Eng. Stud.*, vol. 4, no. 2, pp. 76–81, 2017.

[12] H. Alzahrani, "A brief survey of cloud computing," *Global J. Comput. Sci. Technol.*, vol. 1, no. 1, 2016.

[13] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[14] R. Panarin, "Everything You Need to Know About Edge Computing," Mad Devs Blog, 2024. [Online]. Available : `https://maddevs.io/blog/everything-you-need-to-know-about-edge-computing/`. [Accessed : Feb. 26, 2026].

[15] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[16] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for Internet of Things : a primer," *Digit. Commun. Netw.*, vol. 4, no. 2, pp. 77–86, 2018.

[17] S. Yi, C. Li, and Q. Li, "Fog computing : Platform and applications," in *Proc. 3rd IEEE Workshop Hot Topics Web Syst. Technol. (HotWeb)*, 2015, pp. 73–78.

[18] M. Mukherjee *et al.*, "Security and privacy in fog computing : Challenges," *IEEE Access*, vol. 5, pp. 19293–19304, 2017.

[19] G. Peralta *et al.*, "Fog computing based efficient IoT scheme for the Industry 4.0," in *Proc. IEEE Int. Workshop Electron. Control Meas. Signals Appl. Mechatronics (ECMSM)*, 2017, pp. 1–6.

[20] Cisco, "Fog Computing and the Internet of Things : Extend the Cloud to Where the Things Are," White Paper, Cisco Systems, San Jose, CA, USA, 2016. [Online]. Available : `https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf`. [Accessed : Apr. 10, 2026].

[21] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing : Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[22] P. Garcia Lopez *et al.*, "Edge-centric computing : Vision and challenges," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sep. 2015.

[23] M. Tseng, T. E. Canaran, and L. Canaran, "Introduction to Edge Computing in IIoT," Technical Report, Industrial Internet Consortium, 2018.

[24] D. Linthicum, "Edge computing vs. fog computing : definitions and enterprise uses," Cisco Technical Articles, 2018. [Online]. Available : `https://www.cisco.com/`. [Accessed : Apr. 10, 2026].

[25] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017.

[26] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu, "Data security and privacy preserving in edge computing paradigm : survey and open issues," *IEEE Access*, vol. 6, pp. 18209–18237, Mar. 2018.

[27] Edge Computing Consortium & Alliance of Industrial Internet, "Edge Computing Reference Architecture 2.0," Technical Report, 2017.

[28] S. Singh, "Optimize cloud computations using edge computing," in *Proc. Int. Conf. Big Data, IoT and Data Science (BID)*, Pune, India, 2017, pp. 49–53, doi : 10.1109/BID.2017.8335128.

[29] A. Mebrek, "Fog Computing pour l'Internet des objets," Ph.D. dissertation, Université de Technologie de Troyes, Troyes, France, 2020. [Online]. Available : `https://theses.hal.science/tel-03808714`. [Accessed : Apr. 10, 2026].

[30] M. E. K. Fareh, "Une approche basée agents pour l'allocation des ressources dans le Cloud Computing," Mémoire de magistère, Université Mohamed Khider de Biskra, Algeria, 2015.

[31] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf, "NIST Cloud Computing Reference Architecture," National Institute of Standards and Technology, Gaithersburg, MD, USA, Spec. Publ. 500-292, 2011.

[32] S. Yassa, "Allocation optimale multicontraintes des workflows aux ressources d'un environnement Cloud Computing," Thèse de doctorat, Université de Cergy-Pontoise, France, 2014.

[33] E. Daubert, "Adaptation et Cloud Computing : un besoin d'abstraction pour une gestion transverse," Thèse de doctorat en informatique, Institut National des Sciences Appliquées de Rennes, France, 2013.

[34] B. P. Rimal, E. Choi, and I. Lumb, "A Taxonomy and Survey of Cloud Computing Systems," in *Proc. 5th Int. Joint Conf. INC, IMS and IDC (NCM)*, 2009, pp. 44–51.

[35] I. P. Egwutuoha, S. Chen, D. Levy, B. Selic, and R. Calvo, "A proactive fault tolerance approach to High Performance Computing (HPC) in the cloud," in *Proc. 2nd Int. Conf. Cloud and Green Computing (CGC)*, Nov. 2012, pp. 268–273.

[36] BareMetalCloud, "BareMetalCloud," [Online]. Available : `http://baremetalcloud.com/index.php/en/`. [Accessed : Apr. 15, 2026].

[37] Grid5000, "Grid5000 Platform," [Online]. Available : `http://www.grid5000.fr`. [Accessed : Apr. 15, 2026].

[38] SoftLayer, "SoftLayer Cloud Infrastructure," [Online]. Available : `http://www.softlayer.com/`. [Accessed : Apr. 15, 2026].

[39] National Institute of Standards and Technology, "Fog Computing Conceptual Model," NIST Special Publication 500-325, 2018.

[40] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing : A literature survey," *Future Gener. Comput. Syst.*, vol. 91, pp. 407–415, Feb. 2019, doi : 10.1016/j.future.2018.09.014.

[41] T. Aladwani, "Types of Task Scheduling Algorithms in Cloud Computing," in *Scheduling Problems : New Applications and Trends*, 2020, pp. 131–142.

[42] N. Tripathy, S. Sahoo, N. S. Alghamdi, W. Viriyasitavat, and G. Dhiman, "Energy and makespan optimised task mapping in fog enabled IoT application : A hybrid approach," *Sci. Rep.*, 2026. [Online]. Available : `https://www.nature.com/articles/s41598-025-35065-9`.

[43] J. L. De Souza Toniolli and B. Jaumard, "Resource Allocation for Multiple Workflows in Cloud-Fog Computing Systems," in *Proc. 12th IEEE/ACM Int. Conf. Utility and Cloud Computing Companion (UCC Companion)*, 2019, pp. 77–84, doi : 10.1145/3368235.3368846.

[44] F. Bensassi and S. Arraria, "Ordonnancement temps réel des tâches indépendantes sur environnements de cloud computing," Master's thesis, Université Ibn Khaldoun, Tiaret, Algeria, 2020. [Online]. Available : `http://dspace.univ-tiaret.dz/handle/123456789/5363`.

[45] M. Ghofiri Amina and O. Guettaia Oussama, "Utilisation des mécanismes d'ordonnancement pour la gestion des ressources dans un réseau à base de Fog Computing," Master's thesis, Université de Tlemcen, Algeria, 2023–2024.

[46] R. Belmahdi, D. Mechta, and S. Harous, "A Survey on Various Methods and Algorithms of Scheduling in Fog Computing," *Ingénierie des Systèmes d'Information*, vol. 26, no. 2, pp. 211–224, 2021, doi : 10.18280/isi.260208.

[47] M. Trabelsi, "Energy and Cost Aware Real Time Task Scheduling with Deadline Constraints in Fog Computing Environments," in *Proc. Int. Conf. on Cloud and Fog Computing*, SciTePress, 2024.

[48] L. Mall, P. Vashisht, and A. Narang, "Quality of service based on scheduling in cloud/-fog environment," *Int. J. Innovative Research Eng. Manag.*, vol. 11, no. 3, pp. 65–70, 2024.

[49] H. Suleiman, "A cost-aware framework for QoS-based and energy-efficient scheduling in cloud–fog computing," *Future Internet*, vol. 14, no. 11, p. 333, 2022.

[50] M. S. Kumar and G. R. Karri, "EEOA : Cost and energy efficient task scheduling in a cloud-fog framework," *Sensors*, vol. 23, no. 5, p. 2445, 2023.

[51] A. Chergui, A. N. Dahmani, and A. Adda Abbou, "Ordonnancement d'un flow-shop par métaheuristique hybride," Master's thesis, Université de Tlemcen, Algeria, Jun. 2017.

[52] F. K. Karim, S. Ghorashi, S. Alkhalaf, S. H. A. Hamza, A. Ben Ishak, and S. Abdel-Khalek, "Optimizing makespan and resource utilization in cloud computing environment via evolutionary scheduling approach," *PLoS ONE*, vol. 19, no. 11, p. e0311814, 2024.

[53] H. Nouhas, A. Belangour, and M. Nassar, "Heuristic-Based Workload Scheduling Approaches in Edge-Cloud Environments : A Review," Preprint, 2024.

[54] A. Muthusamy and R. K. Dhanaraj, "Dynamic Q-Learning-Based Optimized Load Balancing Technique in Cloud," *Mobile Inf. Syst.*, vol. 2023, Art. no. 7250267, 2023.

[55] S. Radhika, S. K. Swain, S. Adinarayana, and B. R. Babu, "Efficient Task Scheduling in Cloud Using Double Deep Q-Network," *Int. J. Comput. Digit. Syst.*, vol. 17, no. 1, pp. 1–11, 2025.

[56] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[57] N. R. Alharbe, "Fuzzy clustering based scheduling algorithm for minimizing the tasks completion time in cloud computing environment," *Sci. Rep.*, vol. 15, no. 1, p. 19505, Jun. 2025, doi : 10.1038/s41598-025-02654-z.

[58] S. Santra, "A review on task-scheduling algorithms in the cloud computing towards energy efficiency," *J. Emerg. Trends Comput. Sci. Appl.*, vol. 1, no. 2, pp. 8–16, 2025.

[59] D. Saxena and D. R. K. Chauhan, "Shortest-Job First With Fair Priority and Energy Awareness Scheduling in Green Cloud Computing," in *Proc. Int. Conf. Comput., Commun. and Autom. (ICCCA)*, 2016.

[60] A. Khan, A. Abbas, H. A. Khattak, F. Rehman, I. U. Din, and S. Ali, "Effective Task Scheduling in Critical Fog Applications," *Sci. Program.*, vol. 2022, Art. no. 9208066, 2022, doi : 10.1155/2022/9208066.

[61] I. Moukhtari, R. Bakiri, and S. Cherbal, "PS-DDAF : Priority-based Scheduling and Deadline/Distance-based Allocation for Performance Optimization in Fog Computing," *Algerian J. Sci.*, vol. 2, no. 1, pp. 1–14, 2025.

[62] A. Rahman *et al.*, "EcoTaskSched : A hybrid machine learning approach for energy-efficient task scheduling in IoT-based fog-cloud environments," *Sci. Rep.*, vol. 15, no. 1, p. 9786, 2025.

[63] M. Mokni, S. Yassa, J. E. Hajlaoui, M. N. Omri, and R. Chelouah, "Multi-objective fuzzy approach to scheduling and offloading workflow tasks in fog–cloud computing," *Simul. Model. Pract. Theory*, vol. 123, p. 102687, Feb. 2023.

[64] M. Fatima, "An Efficient Energy Aware Task Scheduling in Cloud Environment," in *Proc. EAI Int. Conf. Sustainability, Innovation and Science (SIS)*, 2023.

[65] S. Ijaz and M. A. Munir, "Energy makespan optimization of workflow scheduling in fog cloud computing," *Future Gener. Comput. Syst.*, vol. 123, pp. 145–158, 2024.

[66] N. Thakur *et al.*, "Deadline aware and energy efficient IoT task scheduling in fog computing systems : A semi greedy approach," *J. Netw. Comput. Appl.*, vol. 200, p. 103147, 2022.

[67] R. Sharma *et al.*, "Energy Aware Task Scheduling in IoT Based Fog Environment," in *Proc. IEEE Int. Conf. Cloud and Fog Comput. (ICCFog)*, 2023.

[68] R. Sing, S. K. Bhoi, N. Panigrahi, K. S. Sahoo, M. Bilal, and S. C. Shah, "EMCS : An Energy-Efficient Makespan Cost-Aware Scheduling Algorithm Using Evolutionary Learning Approach for Cloud-Fog-Based IoT Applications," *Sustainability*, vol. 14, no. 22, p. 15096, Nov. 2022, doi : 10.3390/su142215096.

[69] B. Jamil, H. Ijaz, M. Shojafar, K. Munir, and R. Buyya, "Resource Allocation and Task Scheduling in Fog Computing and Internet of Everything Environments : A Taxonomy, Review, and Future Directions," *ACM Comput. Surv.*, vol. 54, no. 11s, Art. no. 233, pp. 1–38, Sep. 2022, doi : 10.1145/3513002.

[70] M. Tigane, "Ordonnancement et Gestion de l'Énergie," Thèse de doctorat, Université des Sciences et de la Technologie Houari Boumedienne, Alger, Algeria, 2021.

[71] "Server Utilization — an overview," *ScienceDirect Topics*. [Online]. Available : `https://www.sciencedirect.com/topics/computer-science/server-utilization`. [Accessed : May 6, 2026].

[72] S. Gangwar, *Operating System : Scheduling Algorithm*, Department of Computer Applications, VBS Purvanchal University, Jaunpur, India, n.d.

[73] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, doi : 10.1038/nature14236.

[74] R. N. Calheiros *et al.*, "CloudSim : A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Softw.: Pract. Exp.*, vol. 41, no. 1, pp. 23–50, 2011.

[75] H. Gupta *et al.*, "iFogSim : A Toolkit for Modeling and Simulation of Resource Management Techniques in Internet of Things, Edge and Fog Computing Environments," *Softw.: Pract. Exp.*, vol. 47, no. 9, pp. 1275–1296, 2017.

[76] M. M. A. Khan *et al.*, "FogNetSim++ : A Toolkit for Modeling and Simulation of Distributed Fog Environment," in *Proc. Int. Conf. Adv. Comput., Commun. Inf. (ICACCI)*, 2017.

[77] PyFog Developers, *PyFog : Python-based Fog Computing Simulation Framework*. [Online]. Available : `https://github.com/pyfog`. [Accessed : May 2026].

[78] C. Guerrero *et al.*, "YAFS : A Simulator for IoT Scenarios in Fog Computing," *IEEE Access*, vol. 7, pp. 91745–91758, 2019.

[79] SimPy Developers, *SimPy Documentation*. [Online]. Available : `https://simpy.readthedocs.io/`. [Accessed : May 2026].

[80] NumPy Developers, *NumPy Documentation*. [Online]. Available : `https://numpy.org/doc/`. [Accessed : May 2026].

[81] Microsoft, "Visual Studio Code Documentation." [Online]. Available : `https://code.visualstudio.com/docs`. [Accessed : May 2026].

[82] HEIG-ARC, "Le module random — Le langage Python." [Online]. Available : `https://he-arc.github.io/livre-python/random/`. [Accessed : May 2026].

[83] Wild Code School, "Pourquoi et comment utiliser Matplotlib avec Python." [Online]. Available : `https://www.wildcodeschool.com/blog/pourquoi-et-comment-utiliser-matplotlib-avec-python`. [Accessed : May 2026].

[84] 01net, "Téléchargement Python." [Online]. Available : `https://www.01net.com/telecharger/programmation/creation/python.html`. [Accessed : May 2026].

[85] A. Paszke *et al.*, "PyTorch : An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[86] Python Software Foundation, “time — Time access and conversions.” [Online]. Available : `https://docs.python.org/3/library/time.html`. [Accessed : May 2026].

[87] Python Software Foundation, “csv — CSV File Reading and Writing.” [Online]. Available : `https://docs.python.org/3/library/csv.html`. [Accessed : May 2026].

[88] Python Software Foundation, “datetime — Basic date and time types.” [Online]. Available : `https://docs.python.org/3/library/datetime.html`. [Accessed : May 2026].

[89] A. Jayanetti, S. Halgamuge, and R. Buyya, “Deep reinforcement learning for energy and time optimized scheduling of precedence-constrained tasks in edge–cloud computing environments,” *Future Gener. Comput. Syst.*, vol. 137, pp. 14–30, 2022, doi : 10.1016/j.future.2022.06.012.

[90] G. S. S. Chalapathi, V. Chamola, A. Vaish, and R. Buyya, “Industrial Internet of Things (IIoT) applications of edge and fog computing : A review and future directions,” arXiv preprint arXiv :1912.00595, 2019.

[91] Y. Luo, W. Li, W. Yang, and G. Fortino, “A real-time edge scheduling and adjustment framework for highly customizable factories,” *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5625–5634, Aug. 2021.