République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Université Abderrahmane Mira Bejaïa

Faculté de Sciences Exactes

Département de recherche opérationnelle

Mémoire de Master en Mathématiques Appliquées

Option : Optimisation et Fiabilité des Réseaux de Communication

Équilibrage de Charge dans un Environnement Cloud

Réalisé par : MAZOUZ Lina

Soutenu le 04/07/2024 devant le jury composé de :

Présidente	Pr LEKADIR Ouiza	U. A/MIRA BÉJAÏA
Examinatrice	Pr BOULEFKHAR Samra	U. A/MIRA BÉJAÏA
Examinatrice	Dr BERNINE Nassima	U. A/MIRA BÉJAÏA
Encadreur	Dr KHIMOUM Noureddine	U. A/MIRA BÉJAÏA
Co-Encadreur	Dr EL-SAKAAN Nadim	U. A/MIRA BÉJAÏA

2023 – 2024

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué à la réalisation de ce mémoire.

En premier lieu, je remercie sincèrement mes encadreurs, Monsieur Khimoum et Monsieur El-Sakaan, pour leurs conseils avisés et leur encadrement tout au long de ce travail. Leur expertise, leur patience et leur disponibilité ont été essentiels à l'aboutissement de ce projet. Merci pour votre accompagnement et votre confiance.

Je souhaite également remercier les membres du jury pour avoir accepté de juger ce travail. Votre disponibilité et vos remarques constructives seront très appréciées et contribueront à améliorer la qualité de ce mémoire.

Enfin, je remercie ma famille et mes amis pour leur soutien moral et leur encouragement tout au long de cette période.

Table des matières

Table des figures

Liste des tableaux

Introduction Générale

De nos jours, le cloud computing est devenu une véritable révolution. Il a transformé la manière dont les ressources informatiques sont fournies et utilisées. Plutôt qu'un simple effet de mode, le cloud computing offre une infrastructure flexible, évolutive et accessible via Internet. Cela permet aux entreprises, aux gouvernements et aux individus de gérer et de traiter des volumes de données sans précédent. Cette avancée technologique encourage l'innovation, améliore l'efficacité opérationnelle et réduit considérablement les coûts liés à l'achat et à la maintenance de l'infrastructure traditionnelle.

Cependant, avec l'adoption généralisée du cloud computing, de nouveaux défis apparaissent, notamment en ce qui concerne la gestion optimale des ressources disponibles. Parmi ces défis, l'équilibrage de charge est essentiel. Il vise à répartir uniformément les tâches et les demandes des utilisateurs sur plusieurs serveurs, évitant ainsi les surcharges et garantissant une performance optimale du système. Cela est particulièrement important dans un environnement de cloud computing où les demandes peuvent fluctuer de manière imprévisible, nécessitant des solutions dynamiques et adaptatives.

Ce mémoire se concentre spécifiquement sur le problème de l'équilibrage de charge dans le cloud computing. L'objectif est d'explorer les différentes méthodes et algorithmes développés pour assurer une distribution efficace des charges de travail, minimiser les temps d'attente et maximiser l'utilisation des ressources tout en minimisant les violations des contrats de qualité de service (QoS). Nous aborderons également les implications de ces techniques sur la qualité de service et la satisfaction des utilisateurs finaux.

Le premier chapitre de ce mémoire offre une vue d'ensemble du cloud computing, en couvrant son historique, ses caractéristiques, ainsi que les différents modèles de service et de déploiement. Nous analyserons les avantages et les inconvénients de cette technologie et présenterons des exemples notables de services de cloud computing. Le second chapitre se concentrera sur les mécanismes de gestion des ressources, avec une attention particulière sur l'équilibrage de charge, la planification des tâches et la tolérance aux pannes. Nous explorerons diverses approches d'ordonnancement et leurs applications dans des scénarios de cloud computing. Enfin, le troisième chapitre présentera une modélisation détaillée du problème d'équilibrage de charge et proposera un algorithme de résolution basé sur MATLAB, accompagné d'analyses comparatives et de tableaux de performance.

En conclusion, ce mémoire vise à fournir une compréhension approfondie des défis et des solutions liés à l'équilibrage de charge dans le cloud computing, tout en mettant en lumière les innovations actuelles et les perspectives futures dans ce domaine en plein essor.

Chapitre 1

Généralités sur le cloud computing

1.1 Introduction

Dans ce qui suit, nous donnerons quelques concepts de base et notions fondamentales sur le Cloud Computing (CC) et notamment une terminologie qui nous servira pour l'étude et l'analyse du problème posé. Ce chapitre explore les origines, l'évolution et les caractéristiques clés du cloud computing, ainsi que ses modèles de déploiement et les défis liés à son adoption.

1.2 Qu'est-ce que le Cloud Computing ?

Le cloud computing (CC), ou informatique en nuage, est un modèle informatique révolutionnaire qui a transformé les technologies de l'information modernes. En offrant un accès flexible et évolutif à des ressources partagées via Internet telles que le stockage de données, le traitement de données et l'exécution d'applications, il permet aux entreprises, aux gouvernements et aux individus de gérer leurs données de manière plus efficace et économique. En résumé, cette technologie permet de réduire significativement les coûts liés à l'achat et à la maintenance de serveurs et de logiciels, tout en offrant la scalabilité nécessaire pour ajuster les ressources informatiques en réponse aux demandes fluctuantes, révolutionnant ainsi la façon dont les informations sont utilisées et gérées.

Nous avons retenu la définition suivante sur le (CC), émise par le NIST (NATIONAL INSTITUT OF STANDARDS AND TECHNOLOGY) :

Définition 1.1 *Le cloud computing est un modèle permettant un accès réseau omniprésent, pratique et à la demande à un pool partagé de ressources informatiques configurables (par exemple, réseaux, serveurs, stockage, applications et services) qui peuvent être rapidement mises à disposition et libérées avec un minimum d'efforts de gestion ou d'interaction avec le fournisseur de services.(NIST, 2014)*

1.3 Origine, évolution et virtualisation

L'origine du cloud remonte au concept de partage de temps apparu dans les débuts des années 1950, qui a permis à plusieurs utilisateurs d'accéder simultanément à un grand ordinateur central depuis divers terminaux, suivi du lancement de l'ARPANET, précurseur de l'internet en 1969, facilitant le partage de ressources informatiques à distance (Vouk, 2008).

1.3.1 Expansion et Virtualisation

Les années 1980 ont vu l'expansion du réseau vers un réseau global, augmentant la connectivité entre systèmes informatiques à travers le monde. En 1999, VMware révolutionne la gestion des serveurs avec la virtualisation, permettant à plusieurs systèmes d'exploitation de coexister sur un seul serveur physique.

1.3.2 Commercialisation et Croissance

Dans les années 2000, SALESFORCE a été le pionnier du modèle SaaS (Software as a Service) en offrant ses applications d'entreprise via internet. En 2006, AMAZON WEB SERVICES (AWS) a marqué un tournant en proposant des services d'infrastructure sur demande, établissant le modèle IaaS (Infrastructure as a Service). L'entrée de Google et Microsoft avec leurs propres services cloud, GOOGLE CLOUD et MICROSOFT AZURE, a stimulé la concurrence et l'innovation dans l'industrie.

1.3.3 Diversification et Innovation

Depuis 2010, le cloud s'est diversifié au-delà de l'infrastructure pour inclure des plateformes de développement (PAAS) et des services innovants. L'adoption généralisée a mis l'accent sur la sécurité et la réglementation. Le cloud a facilité des avancées dans l'intelligence artificielle, l'analyse de données et l'Internet des objets (IoT).

1.3.4 Caractéristiques

Les caractéristiques essentielles et fondamentales du Cloud Computing sont les propriétés suivantes, considérées d'ailleurs comme les propriétés qu'une solution cloud devait posséder. Si une solution ne possédait pas les cinq caractéristiques suivantes, elle n'était pas considérée comme une solution cloud (Lisdorf, 2021).

Service à la demande Les utilisateurs peuvent provisionner des capacités informatiques sans intervention humaine, comme le temps serveur et le stockage réseau, selon leurs besoins.

Accès au réseau étendu Les capacités sont disponibles sur le réseau et accessibles via des mécanismes standards, facilitant leur utilisation sur diverses plates-formes clientes.

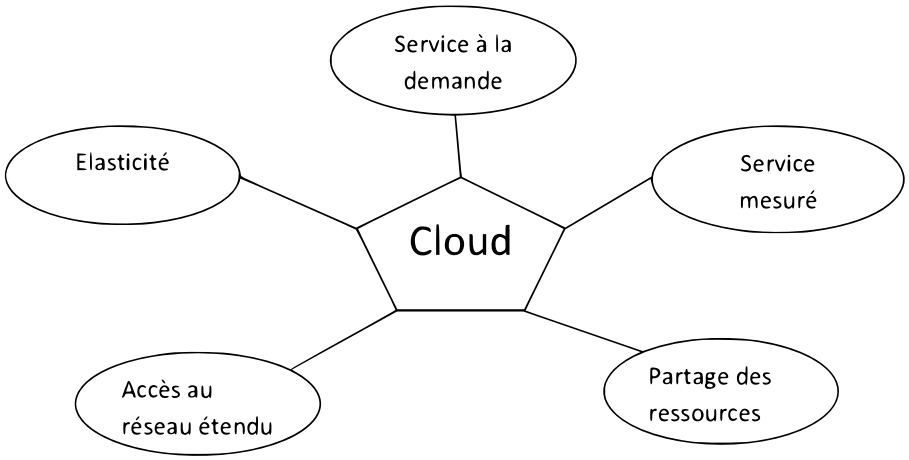

FIGURE 1.1 – Caractéristiques du cloud computing

Mise en commun des ressources Les ressources informatiques du fournisseur sont regroupées pour servir plusieurs utilisateurs, avec une assignation dynamique basée sur la demande, offrant une indépendance vis-à-vis de l'emplacement précis des ressources.

Élasticité rapide Les capacités peuvent être provisionnées et libérées de manière élastique, s'adaptant rapidement à la demande, offrant une apparence d'illimité pour l'utilisateur.

Service mesuré Les systèmes cloud contrôlent et optimisent l'utilisation des ressources grâce à une capacité de mesure, fournissant transparence et contrôle sur l'utilisation des services.

1.4 Modèles de service

On distingue les trois principaux types de services du Cloud Computing (Production, 2010) :

1.4.1 IaaS (Infrastructure as a Service)

Le matériel est fourni au client, qui est chargé d'installer tous les composants de la pile logicielle sur le matériel : systèmes d'exploitation, intergiciels, environnements d'exécution et applications. Le fournisseur de cloud est uniquement responsable de la gestion de la partie matérielle.

1.4.2 PaaS (Platform as a Service)

La responsabilité du fournisseur est déplacée un peu plus haut dans la pile. il sera chargé de l'installation et de la gestion des systèmes d'exploitation, des intergiciels et de tous les environnements d'exécution requis.

1.4.3 SaaS (Software as a Service)

Dans ce modèle, le client interagit avec les services cloud via une interface graphique (GUI). En effet, tous les services requis sont livrés sous forme d'application prête à l'emploi et le fournisseur est responsable de l'ensemble de la pile logiciel.

1.5 Modèles de Déploiement du Cloud

Une fois que le service de Cloud Computing est sélectionné par un client, il a le choix entre trois principaux modèles de déploiement, qui diffèrent selon les niveaux

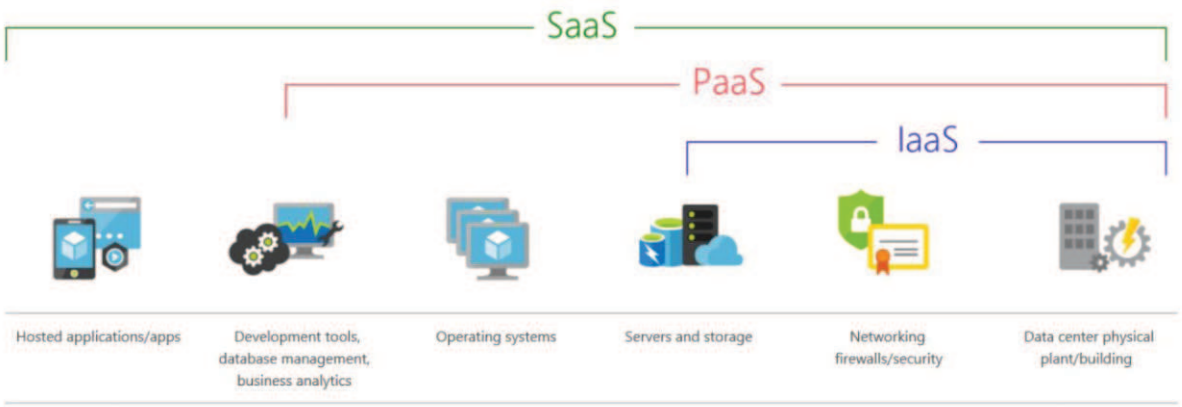

FIGURE 1.2 – Modèles de Services Cloud

de sécurité et de gestion. Chaque type requiert un niveau de gestion différent du client et fournit un niveau de sécurité différent (Chopra, 2018).

1.5.1 Privé

Le cloud privé est un modèle de déploiement où l'infrastructure cloud est provisionnée pour une utilisation exclusive par une seule organisation, comprenant plusieurs consommateurs tels que des unités commerciales. Cette infrastructure peut être possédée, gérée et exploitée par l'organisation elle-même, un tiers, ou une combinaison des deux, et elle peut être située sur site ou à distance.

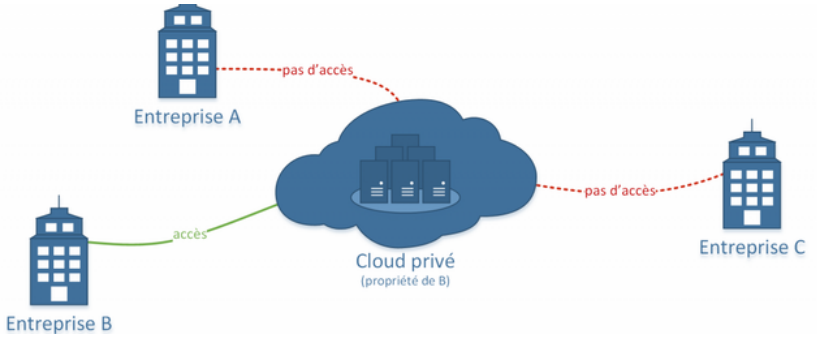

FIGURE 1.3 – modèles de déploiement d'un cloud privé

1.5.2 Communautaire

Le cloud communautaire est un modèle où l'infrastructure cloud est provisionnée pour une utilisation exclusive par une communauté spécifique de consommateurs provenant d'organisations ayant des préoccupations partagées telles que la mission,

les exigences en matière de sécurité, les politiques et les considérations de conformité. Cette infrastructure peut être possédée, gérée et exploitée par une ou plusieurs des organisations de la communauté, un tiers, ou une combinaison des deux, et elle peut être située sur site ou à distance.

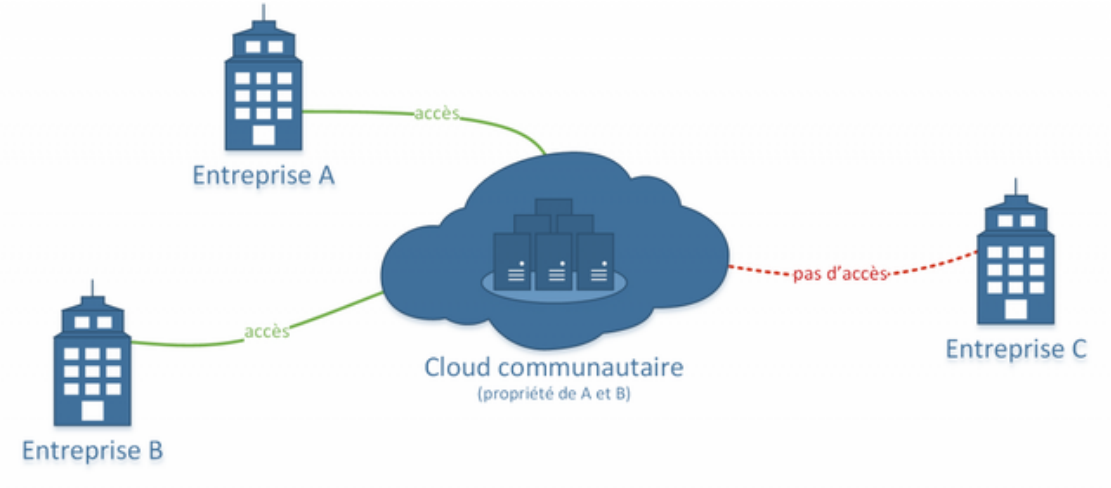

FIGURE 1.4 – modèles de déploiement d'un cloud communautaire

1.5.3 Public

Le cloud public est un modèle de déploiement où l'infrastructure cloud est provisionnée pour une utilisation ouverte par le grand public. Cette infrastructure peut être possédée, gérée et exploitée par une entreprise, une institution académique, une organisation gouvernementale, ou une combinaison de celles-ci, et elle est généralement située sur site chez le fournisseur de cloud.

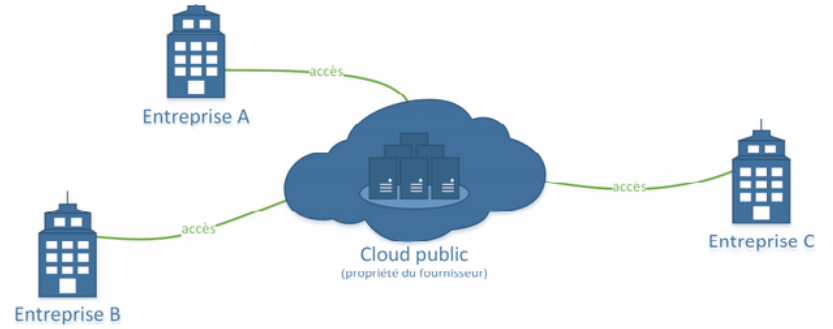

FIGURE 1.5 – modèles de déploiement d'un cloud public

1.5.4 Hybride

Le cloud hybride est une composition de deux infrastructures cloud ou plus (privée, communautaire, ou publique) qui restent des entités distinctes mais sont liées par une technologie standardisée ou propriétaire permettant la portabilité des données et des applications. Cela inclut par exemple la capacité à équilibrer la charge entre différents clouds via le "cloud bursting".

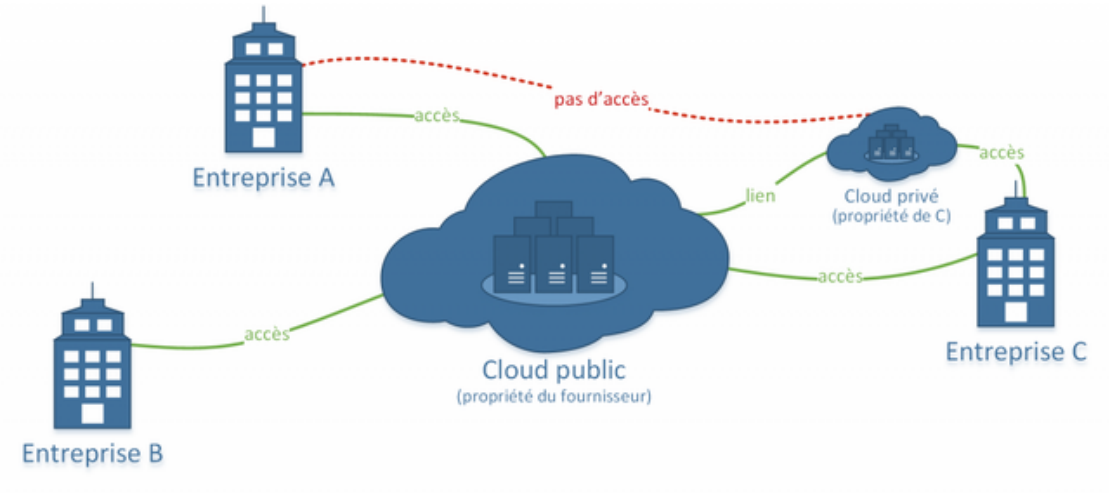

FIGURE 1.6 – modèles de déploiement d'un cloud hybride

1.6 Fournisseurs de Services Cloud

Google Cloud Platform Service de Google pour le stockage et la synchronisation de fichiers, intégrant des outils de productivité comme Google Docs. Idéal pour le travail collaboratif et accessible depuis tout dispositif connecté.

Apple iCloud Intégré aux produits Apple, iCloud stocke photos, vidéos et sauvegardes d'iOS, facilitant la synchronisation entre iPhone, iPad et Mac.

Amazon Web Services Collection de services cloud d'Amazon offrant stockage, calcul et gestion de bases de données, utilisée par des entreprises pour des applications variées, de l'analytique à l'hébergement d'applications.

Microsoft Azure Azure offre des services de calcul, d'analytique, de stockage et de mise en réseau. Elle est utilisée par les entreprises pour développer, déployer et gérer des applications.

IBM Cloud Connu pour ses solutions hybrides et multi-cloud, avec un accent sur les services d'intelligence artificielle, d'analytique avancée et de sécurité.

1.7 Avantages et Inconvénients du CC

Le cloud computing a révolutionné la gestion des ressources informatiques, cependant, comme toute technologie, il présente de nombreux avantages mais aussi des inconvénients que nous allons explorer dans cette partie.

1.7.1 Avantages

Parmi les nombreux avantages du Cloud Computing, nous énumérons pour en citer que quelques uns (Armbrust et al., 2010), les avantages suivants :

Réduction des coûts

Le cloud computing permet de réduire les coûts totaux d'investissement et des dépenses opérationnelles substantiels. L'absence d'infrastructure sur site réduit également les coûts opérationnels associés tels que les frais d'électricité, de climatisation et d'administration.

Flexibilité et Scalabilité

L'élasticité du cloud permet une évolutivité rapide des ressources informatiques en fonction des besoins, ce qui permet de répondre aux pics de charge sans investissements préalables importants.

Mise à jour et maintenance simplifiées

Les fournisseurs de services cloud se chargent de la mise à jour et de la maintenance de l'infrastructure, garantissant ainsi que les systèmes sont toujours à jour et fonctionnent de manière optimale. Les clients bénéficient d'une interface utilisateur web simple pour accéder aux logiciels, aux applications et aux services, sans les posséder.

Sauvegarde et récupération des données

Le cloud computing offre une sauvegarde sécurisée et rapide des données, réduisant ainsi les risques de perte ou de corruption. Il permet également des sauvegardes automatiques régulières sans intervention manuelle, et un accès facile aux données à tout moment et depuis n'importe quel appareil connecté à Internet.

1.7.2 Inconvénients

Malgré ses nombreux avantages, le Cloud Computing présente également des aspects négatifs qui peuvent impacter les organisations (Vacca, 2016). Voici les principaux côtés négatifs :

Dépendance à la connectivité Internet

Cette technologie repose entièrement sur une connexion Internet fiable et performante. En cas de problèmes de connexion ou de perte de connectivité, les utilisateurs peuvent rencontrer des difficultés pour accéder à leurs données stockées dans le cloud ou pour effectuer des opérations en ligne. Cela peut entraver leur productivité et causer des retards dans les projets.

Risques de sécurité

Bien que les fournisseurs de services cloud offrent généralement des mesures de sécurité robustes, la confiance dans la sécurité des données peut être compromise par le manque de visibilité sur l'emplacement physique du stockage et par les préoccupations concernant la confidentialité des données.

Tout n'est pas adapté au cloud

Les organisations peuvent avoir des applications développées pour des besoins spécifiques. Ces applications ne s'adaptent pas à l'externalisation. Les fournisseurs de services en cloud n'ont généralement aucun avantage à accepter une seule application, car leur modèle repose plutôt sur le développement d'une application unique vendue plusieurs fois.

1.8 Mécanismes de Gestion des Ressources

L'équilibrage de charge répartit la charge de travail sur les nœuds d'un système pour optimiser l'utilisation des ressources et améliorer les temps de réponse. Il évite que certains nœuds soient surchargés tandis que d'autres restent sous-utilisés. Un algorithme dynamique s'adapte en fonction de l'état actuel du système pour garantir une répartition efficace des tâches.Nous reprenons la définition suivante tirée de (Zhang, Cheng, & Boutaba, 2010).

Définition 1.2 *L'équilibrage de charge est un processus clé dans la gestion des ressources informatiques, particulièrement dans les environnements de cloud computing. Il vise à répartir de manière optimale la charge de travail entre plusieurs ressources (serveurs, processeurs, etc.) pour éviter les surcharges et garantir une utilisation efficace des capacités disponibles.*

Objectifs de l'Équilibrage de Charge

Les principaux objectifs de l'équilibrage de charge sont :

1. Améliorer la Performance : Réduire les temps de réponse en évitant que certains serveurs soient surchargés tandis que d'autres restent sous-utilisés.
2. Assurer la Disponibilité : Garantir que les ressources sont toujours disponibles pour traiter les requêtes, même en cas de pannes de certaines unités.
3. Optimiser l'Utilisation des Ressources : Maximiser l'utilisation des capacités des serveurs pour améliorer l'efficacité globale du système.
4. Réduire les Coûts : Minimiser les coûts opérationnels en utilisant de manière optimale les ressources disponibles et en évitant les surcharges qui pourraient nécessiter des interventions coûteuses.

Principe de l'Équilibrage de Charge

Le principe de base de l'équilibrage de charge est de distribuer les tâches de manière équilibrée sur plusieurs ressources. Cela peut se faire de manière statique, où la répartition est décidée à l'avance, ou de manière dynamique, où la répartition s'adapte en temps réel en fonction de l'état actuel des ressources et des tâches

à traiter. Afin d'appliquer ce processus, il est important de prendre en compte certains éléments lors de l'application du processus comme : l'estimation de la charge, la comparaison de la charge, la stabilité des différents systèmes, les performances du système . . . etc.

Exemple 1.1 *Dans l'exemple qui suit, les auteurs de (Xue, Li, & Nahrstedt, 2001) ont utilisé un processus d'équilibrage de charge dans les réseaux de capteurs sans fil (Wireless Sensor Networks). Ces réseaux sont constitués de nombreux capteurs qui doivent traiter et transmettre des données sans surcharger certains capteurs tout en laissant d'autres inactifs. Cette application montre comment l'équilibrage de charge est réalisé dans ces réseaux ,la charge de travail doit être équilibrée entre les capteurs pour garantir une performance optimale du réseau.*

Fonctionnement :

- **Équilibrage Dynamique** :
  - Les capteurs surveillent la quantité de données qu'ils traitent.
  - Si un capteur devient surchargé, il redirige une partie de ses données vers des voisins moins chargés.
- **Adaptation en Temps Réel** : Le système ajuste les chemins de routage en fonction de l'état des capteurs. Les décisions sont prises en temps réel pour s'adapter rapidement aux variations de la charge.

Bien que cette méthode s'adapte en temps réel aux variations de la charge, optimise l'utilisation des ressources et prolonge la durée de vie du réseau, elle présente des défis tels que la complexité de mise en œuvre, la nécessité d'une surveillance continue, et une consommation d'énergie importante due aux ajustements fréquents.

1.8.1 Planification des Tâches

La planification des tâches est cruciale dans le déploiement et la gestion des services cloud. Elle organise et alloue efficacement les ressources pour exécuter les tâches des applications hébergées (Xu & Li, 2013). Ce processus inclut la définition des priorités, la gestion des dépendances, l'affectation des ressources, l'estimation des temps d'exécution, et l'optimisation des ressources. Une bonne planification améliore l'efficacité et les performances des applications, garantissant une utilisation optimale des ressources et minimisant les temps d'exécution.

Les algorithmes de planification des tâches se classent en trois catégories :

– Algorithmes statiques : Les décisions sont prises avant l'exécution et restent fixes.
– Algorithmes dynamiques : Les décisions sont prises en temps réel en réponse aux changements.
– Algorithmes hybrides : Ils combinent les approches statiques et dynamiques.

1.8.2 Tolérance aux Pannes

La tolérance aux pannes est une exigence importante dans le domaine du cloud computing. Elle fait référence à la capacité d'un système à continuer à fonctionner et à fournir des services malgré des défaillances matérielles ou logicielles. Dans un environnement en nuage, où de nombreux appareils sont connectés, les pannes sont inévitables. C'est pourquoi les fournisseurs de services en nuage utilisent des mécanismes de tolérance aux pannes pour s'assurer que l'interruption des services est réduite au minimum. Ces mécanismes comprennent la redondance, la réplication des données, la détection automatique des erreurs et la récupération. L'objectif final est la disponibilité continue des services en nuage et la satisfaction des utilisateurs finaux (Garg, Versteeg, & Buyya, 2013).

1.9 Problématique

Dans un contexte où les demandes des utilisateurs sont diverses et évolutives, l'équilibrage de charge dans le cloud computing revêt une importance cruciale. Ce processus complexe consiste à affecter judicieusement les ressources disponibles, telles que le processeur, la mémoire, le stockage et la bande passante, afin de garantir une utilisation optimale des capacités des serveurs tout en répondant aux exigences de performance et de disponibilité des applications. En outre, la gestion des ressources dans le cloud doit prendre en compte d'autres contraintes telles que la minimisation des coûts et la maximisation de l'efficacité énergétique. Cela soulève une question fondamentale : comment répartir de manière optimale les tâches et les ressources dans un environnement cloud pour garantir la satisfaction des exigences des utilisateurs, maintenir la performance des serveurs et minimiser le nombre de tâches rejetées, tout en respectant les contraintes de temps et de capacité ?

Chapitre 2

Ordonnancement des tâches dans le Cloud Computing

2.1 Introduction

L'ordonnancement des tâches est une discipline cruciale dans divers domaines de la recherche opérationnelle et de l'informatique . Il joue un rôle essentiel pour garantir des performances optimales et une utilisation efficace des ressources. Ses principes sont applicables à une multitude de contextes, allant de la production industrielle aux services informatiques. Dans chaque contexte, les tâches doivent être planifiées et exécutées de manière à atteindre certains objectifs.

2.2 Concepts et Définitions

Le problème d'ordonnancement consiste à organiser dans le temps la réalisation d'un ensemble de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînements, ...) et de contraintes portant sur l'utilisation et la disponibilité des ressources requises (Carlier & Chrétienne, 1988). Les différents paramètres d'un problème d'ordonnancement incluent les données suivantes :

2.2.1 Tâches

L'une des notions fondamentales qui intervient dans un problème d'ordonnancement est la notion de la tâche (opération), qui est le dénominateur commun des

problèmes d'ordonnancement.

Définition 2.1 (tâche) *Une tâche est un travail élémentaire dont la réalisation nécessite un certain nombre d'unités de temps (sa durée) et d'unités de chaque ressource*

2.2.2 Ressources

Définition 2.2 (Ressource) *Une ressource est un moyen, technique ou humain, dont la disponibilité limitée ou non est connue a priori.*

On distingue deux types de ressources :

Ressource Renouvelable : Une ressource est considérée comme renouvelable lorsqu'elle est remise à disposition pour d'autres tâches après avoir été attribuée. Les ressources renouvelables courantes comprennent les équipements, les processeurs, les données et les employés.

Ressources Consommables : Une ressource est considérée comme consommable lorsqu'elle n'est plus utilisable pour les tâches restantes après avoir été assignée à une entité. Cela comprend des ressources comme l'argent et les ressources naturelles

Que ce soit pour des ressources consommables ou renouvelables, il est possible que certaines ressources ne soient disponibles qu'à certaines périodes, avec une courbe de disponibilité préétablie.

2.2.3 Contraintes

Dans un problème d'ordonnancement, diverses contraintes influencent la manière dont les tâches sont planifiées et exécutées. Nous citons :

Contraintes de Temps

Ces restrictions portent sur les échéances fixées pour la réalisation des tâches, comprenant les dates de début et de fin, ainsi que les limites de temps maximum pour l'exécution. (ex. Il est impératif de finir une tâche avant une date limite déterminée).

Contraintes de Ressources

Les ressources nécessaires pour exécuter les tâches sont couvertes par ces contraintes. (ex. Pour lancer une tâche, on demande une certaine quantité de mémoire et de CPU).

Contraintes de Dépendance

Certaines tâches doivent être terminées avant que d'autres puissent commencer en raison de ces contraintes. (ex. Les données doivent être collectées avant de pouvoir commencer une tâche de traitement).

Contraintes Technologiques

Selon ces contraintes, l'exécution d'une tâche ne peut commencer qu'après la fin des autres tâches. (ex. Dans une chaîne de production, chaque produit doit terminer l'étape précédente avant de pouvoir passer à la suivante).

Contraintes Commerciales

En raison de contraintes commerciales, certaines tâches doivent être complétées avant une date fixe selon ces exigences. (ex. Il est impératif que le lancement d'un produit suive la date de sortie annoncée).

Contraintes Matérielles

Ces restrictions touchent à la faculté des machines de gérer les tâches. Exemple : Une machine ne peut effectuer qu'une seule tâche à la fois, tout comme une imprimante ne peut imprimer qu'un document à la fois.

Contraintes de Main-d'oeuvre

Ces contraintes sont liées à la disponibilité restreinte du personnel. (ex. Pour faire progresser un projet, il est nécessaire d'avoir suffisamment de personnel disponible pour effectuer les tâches requises).

Contraintes Financières

Ces limitations budgétaires sont relatives à l'exécution des tâches. (ex. Il est impératif de respecter le budget alloué lors de la réalisation d'un projet).

2.2.4 Objectifs

Les objectifs d'ordonnancement sont variés et visent à optimiser différents aspects du système pour assurer une performance optimale et une satisfaction accrue des utilisateurs. Voici quelques-uns des objectifs clés de l'ordonnancement :

Minimiser le Temps d'Exécution (Makespan)

Réduire le temps total requis pour exécuter toutes les tâches est l'un des principaux objectifs de l'ordonnancement. Cela permet d'améliorer le rendement global du système et de s'assurer que les tâches sont exécutées avec la plus grande célérité.

Maximiser l'Utilisation des Ressources

L'objectif de cette approche consiste à garantir une utilisation optimale de toutes les ressources disponibles, telles que le CPU, la mémoire, la bande passante, etc. En optimisant l'utilisation des ressources, on peut diminuer les périodes d'inactivité et accroître la productivité du système.

Minimiser les Coûts

Un autre objectif clé est de diminuer les dépenses liées à l'utilisation des ressources. Cela comprend les coûts liés à l'énergie, ceux relatifs à la location ou à l'achat de ressources et aussi les frais opérationnels. Le but est d'assurer des performances élevées tout en fournissant des services de manière rentable.

Améliorer la Qualité de Service (QoS)

Il est essentiel de garantir le respect des niveaux de service définis afin d'assurer la satisfaction des utilisateurs. Cela comprend des éléments tels que le délai de réponse, la disponibilité des services et leur fiabilité. La fidélisation des clients et la compétitivité du fournisseur de services cloud sont favorisées par une excellente qualité de service.

Réduire les Temps d'Attente

Un objectif crucial est de réduire les délais entre les tâches afin d'améliorer l'efficacité et la satisfaction des utilisateurs. La réduction des temps d'attente permet

aux utilisateurs d'obtenir des résultats plus rapidement, ce qui est crucial pour de nombreuses applications nécessitant une réponse rapide.

Eviter la Famine des Tâches

Veiller à ce que toutes les tâches soient effectuées, même celles qui ont une faible priorité. Il est crucial de maintenir l'équité et la justice dans le système en évitant que les tâches soient constamment repoussées jusqú'à ne jamais être exécutées.

2.2.5 Modélisation

Afin de pouvoir résoudre un problème d'ordonnancement, il est essentiel de le modéliser correctement. La modélisation permet de formaliser le problème en termes mathématiques, facilitant ainsi l'application de diverses techniques d'optimisation. Cette section présente les principaux éléments de la formulation mathématique du problème posé sous forme d'un ordonnancement, à savoir :

1. Définition des variables qui représentent les décisions à prendre, telles que l'affectation des tâches aux ressources et le moment de début des tâches.
2. Définition de la fonction à optimiser, qui peut inclure des objectifs tels que la minimisation du temps total d'exécution, la réduction des coûts ou la maximisation de l'utilisation des ressources.
3. Définition des contraintes qui doivent être respectées, comme les capacités des ressources, les délais des tâches et les dépendances entre les tâches.

2.3 Problème d'ordonnancement central

Le problème se présente sous différentes formes selon les contraintes et les objectifs spécifiques :

1. Nombre de tâches et de machines : Combien de tâches doivent être planifiées et combien de machines sont disponibles ?
2. Temps de traitement : Combien de temps chaque tâche prend pour être exécutée sur chaque machine ?

3. Contraintes : Quelles sont les contraintes spécifiques ? Par exemple, certaines tâches doivent être exécutées avant d'autres (contraintes de précédence), ou certaines machines ne peuvent exécuter que certaines tâches.

il s'agit d'ordonnancer, en une durée minimale des tâches soumises à des contraintes de succession et de localisation temporelle.

2.3.1 Représentation des problèmes d'ordonnancement

Il existe trois types de représentations possibles pour un problème d'ordonnancement : le diagramme de Gantt, le graphe Potentiel-Tâches et la méthode PERT.

Diagramme de Gantt

Inventé dans les années 1890 par le polonais Karol Adamiecki, le diagramme de Gantt a été popularisé par l'américain Henry Gantt quelques années plus tard, d'où son nom.Cet outil permet de visualiser les tâches d'un projet sur une ligne de temps. Chaque tâche est représentée par une barre horizontale dont la longueur indique la durée de la tâche .

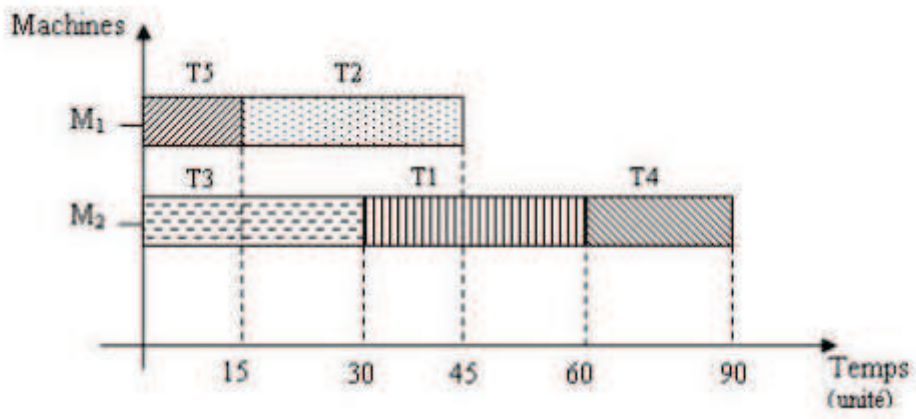

FIGURE 2.1 – Diagramme de Gantt d'un ordonnancement

Graphe Potentiel-Tâches

Le graphe Potentiel-Tâches est une représentation graphique des tâches et de leurs dépendances. Les tâches sont représentées par des nœuds, et les arcs entre les nœuds indiquent les contraintes de précédence. Le graphe de la figure (2.2) aide à

identifier les chemins critiques et les tâches qui doivent être priorisées.

Exemple il est nécessaire de définir :

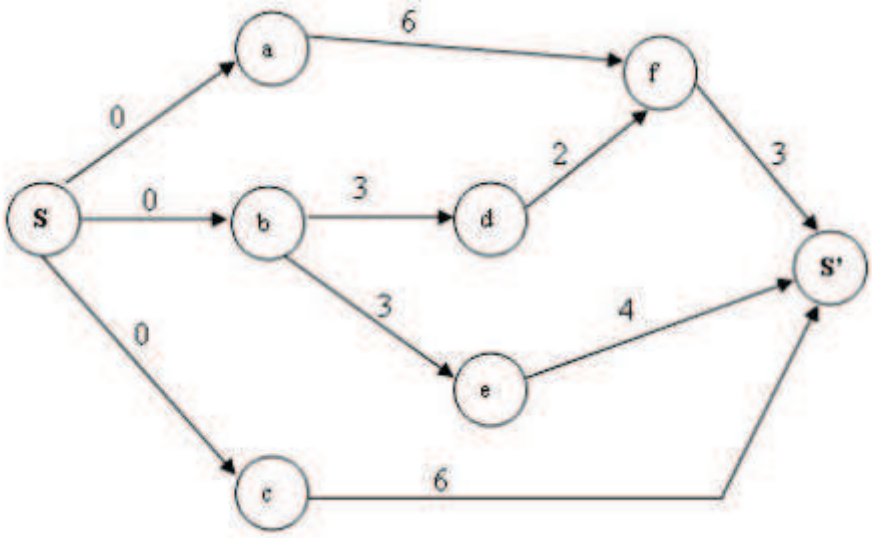

FIGURE 2.2 – Graphe Potentiel-Tâches d'un ordonnancement

- **La date au plus tôt de la tâche** : Cela correspond à la date de début, au plus tôt, de l'exécution de la tâche.
  - En se référant au graphe : La tâche f ne peut s'exécuter que si a et d ont été réalisées. La tâche a prend 6 mois et la tâche d prend 5 mois (2 + 3 mois). La tâche f ne pourra donc commencer qu'au plus tôt 6 mois après le début du projet, correspondant au plus long chemin entre a et f.
- **La durée du projet** : Cela correspond au plus long chemin entre S (tâche de début du projet) et S' (tâche de fin du projet).

Méthode PERT((Program Evaluation and Review Technique))

Cette technique utilise un graphe dirigé pour représenter les tâches et leurs dépendances. Chaque tâche est représentée par un arc, et les nœuds indiquent les événements ou les jalons du projet. PERT permet de calculer les temps optimistes, pessimistes et les temps moyens pour la réalisation des tâches.

Programme Linéaire en 0 et 1

Le problème d'ordonnancement central peut être modélisé en programme linéaire en 0 et 1 :

$$\min \sum_{i,j} c_{ij} x_{ij} \tag{2.1}$$

$$\sum_{i} d_i x_{ij} \leq h_j, \ \forall j \tag{2.2}$$

$$\sum_{j} x_{ij} = 1, \quad \forall i \tag{2.3}$$

$$x_{ij} \in \{0, 1\}, \ \forall i, j \tag{2.4}$$

où

$$x_{ij} = \begin{cases} 1, & \text{si la tâche } j \text{ est exécutée sur la machine } i\,; \\ 0, & \text{sinon.} \end{cases}$$

La contrainte (2.2) signifie que la durée totale des tâches sur chaque ressource ne doit pas dépasser les heures disponibles. La contrainte (2.3) signifie que chaque tâche doit être exécutée exactement une fois. La relation (2.1) exprime l'objectif spécifique du problème, (ex. minimiser les conflits horaires ou maximiser l'utilisation des ressources,...

Exemple 2.1 *Prenons l'exemple de (Pinedo, 2016), où il a supposé avoir une structure similaire pour l'ordonnancement des cours universitaires en ayant un certain nombre de cours à assigner à des salles de classe disponibles. Chaque cours a des besoins spécifiques en termes de durées et de créneaux horaires disponibles.*

Données du Problème

- ***Cours*** *:* (C_1, C_2, C_3, C_4)
- ***Salles*** *:* (S_1, S_2)
- ***Durée des cours*** *:*
  - C_1 *: 2 heures*
  - C_2 *: 3 heures*
  - C_3 *: 1 heure*
  - C_4 *: 2 heures*
- ***Créneaux horaires disponibles*** *:*
  - S_1 *: 8h-10h, 10h-12h, 14h-16h*

– S_2 *: 8h-10h, 10h-12h, 13h-15h*

Objectif *Maximiser l'utilisation des salles.*

Modélisation

– ***Variables de décision*** *:* x_{ij} *qui vaut 1 si le cours i est assigné à la salle j à un créneau horaire donné, 0 sinon.*

– ***Fonction objectif*** *: Maximiser l'utilisation des salles.*

$$\textit{Minimiser} \sum_{i,j} c_{ij} x_{ij}$$

Sous les contraintes :

$$\sum_{j} x_{ij} = 1, \quad \forall i \tag{2.5}$$

- Chaque cours doit être assigné exactement une fois.

$$\sum_{i} d_i x_{ij} \leq h_j, \quad \forall j \tag{2.6}$$

- La durée totale des cours dans chaque salle ne doit pas dépasser les heures disponibles.

$$x_{ij} \in \{0, 1\}, \quad \forall i, j \tag{2.7}$$

Solution

Cours	**Salle**	**Créneau Horaire**
C_1	S_1	8h-10h
C_2	S_2	8h-11h
C_3	S_1	14h-15h
C_4	S_2	13h-15h

TABLE 2.1 – Solution optimale pour l'ordonnancement des cours universitaires

Ainsi, les cours sont planifiés de manière à minimiser les conflits et maximiser l'utilisation des salles disponibles.

2.4 Ordonnancement dans le Cloud

Le type d'ordonnancement dans le cloud computing diffère selon les besoins des tâches et des ressources.Dans cette partie , nous allons explorer ces différents types :

2.4.1 Ordonnancement en Ligne :

Les décisions d'ordonnancement sont prises au fur et à mesure que les tâches arrivent, nécessitant des algorithmes réactifs capables de traiter les informations en temps réel, souvent sans avoir une connaissance complète des tâches à l'avance. Ce type d'ordonnancement est couramment utilisé dans des environnements où les tâches arrivent de manière imprévisible, tels que les systèmes de streaming vidéo ou les applications de traitement en temps réel. Les algorithmes en ligne doivent s'adapter rapidement aux changements et prioriser les tâches de manière dynamique pour maintenir la performance du système (Gupta & Maravelias, 2019) .

2.4.2 Ordonnancement Hors Ligne :

Dans ce type d'ordonnancement, toutes les tâches sont connues à l'avance, permettant ainsi une planification globale des décisions d'ordonnancement avant l'exécution. Cela offre l'avantage d'optimiser plus efficacement l'utilisation des ressources et les temps de traitement. L'ordonnancement hors ligne est particulièrement adapté aux situations où les tâches peuvent être planifiées à l'avance, comme les traitements batch dans les centres de données ou les projets de construction (Pinedo, 2016). Les algorithmes hors ligne peuvent exploiter cette connaissance préalable pour élaborer des plannings optimaux qui minimisent les temps d'attente et maximisent l'utilisation des ressources.

Certains travaux antérieurs sur l'ordonnancement en cloud computing ont exploré diverses méthodes pour optimiser l'efficacité et la performance des systèmes. Nous examinerons certaines de ces approches.

2.4.3 Ordonnancement basé sur les priorités

Dans cette approche, les tâches sont classées par ordre de priorité en fonction de leur importance et les ressources sont allouées de manière à maximiser l'effi-

cacité des tâches les plus importantes (Ghanbari & Othman, 2012). Les priorités peuvent être basées sur la criticité des tâches, les délais ou d'autres critères spécifiques à l'application. Par exemple, dans un système de cloud computing, les tâches critiques telles que les requêtes en temps réel pour les applications médicales ou financières peuvent être traitées avant les tâches moins importantes telles que les tâches de sauvegarde de données non urgentes.

2.4.4 Ordonnancement dynamique

L'ordonnancement dynamique prend des décisions en temps réel sur la base de l'état actuel du système. Cette approche est essentielle dans les environnements où les charges de travail changent de manière imprévisible et nécessitent souvent une réaffectation des ressources. Par exemple, dans les services de streaming vidéo ou de réseaux sociaux, le nombre d'utilisateurs actifs peut fluctuer de manière significative, ce qui nécessite des ajustements constants des ressources pour maintenir la qualité du service (Priore, Gomez, Pino, & Rosillo, 2014).

2.4.5 Ordonnancement multi-objectifs

L'ordonnancement multi-objectifs vise à optimiser simultanément plusieurs critères, tels que la minimisation des temps d'exécution, la réduction des coûts et la maximisation de l'utilisation des ressources (Deb, Pratap, Agarwal, & Meyarivan, 2001). Ces problèmes sont complexes car il faut souvent trouver un équilibre entre des objectifs qui souvent sont contradictoires. Par exemple, dans un environnement cloud, il peut être nécessaire de minimiser les coûts tout en maximisant les performances, ce qui nécessite des compromis entre la consommation de ressources et le temps d'exécution des tâches.

2.5 Étude comparative des approches d'équilibrage de charge dans le cloud computing

L'équilibrage de charge joue un rôle déterminant dans la qualité de service, la maîtrise de la consommation énergétique et la capacité des infrastructures cloud à passer à l'échelle (Zhou, Tian, Buyya, et al., 2024). Nous présentons ci-dessous une

sélection d'articles représentatifs : pour chacun, nous rappelons le contexte étudié, l'objectif poursuivi, l'approche proposée et les principaux résultats, avant de procéder à une comparaison.

Article 1 — Consolidation énergie (consolidation de machines virtuelles) (Beloglazov & Buyya, 2012). Dans des centres de données virtualisés, les auteurs regroupent la charge sur un sous-ensemble d'hôtes afin d'éteindre ceux qui restent peu sollicités, à l'aide de seuils d'utilisation adaptatifs qui déclenchent des migrations planifiées. L'objectif est de réduire la consommation énergétique et le coût d'exploitation tout en maintenant la qualité de service. Les expériences de simulation mettent en évidence une baisse nette de l'énergie et des dépenses, une utilisation plus homogène et une durée totale d'exécution stable ; des élévations ponctuelles du temps de réponse apparaissent toutefois durant les fenêtres de migration, ce qui plaide pour un bornage temporel strict et une orchestration fine des déplacements de machines virtuelles.

Article 2 — Équilibrage inspiré des abeilles (rééquilibrage local) (Babu & Krishna, 2013). La contribution transpose des mécanismes de butinage à l'infrastructure en cloud : chaque nœud observe son voisinage et réaffecte localement les tâches pour lisser les files. L'orientation décentralisée est particulièrement efficace lorsque la charge varie rapidement, avec des gains sensibles en temps de réponse, en débit soutenu et en utilisation des ressources. La démarche limite la coordination globale coûteuse et s'ajuste au contexte en temps quasi réel ; une procédure de calibration publique des paramètres renforcerait sa stabilité dans des environnements hétérogènes.

Article 3 — Colonies de fourmis (placements et migrations) (Dasgupta, Mandal, Dam, et al., 2014). L'allocation et la migration de machines virtuelles sont modélisées comme une exploration distribuée de l'espace des placements, guidée par renforcement indirect et mémoire de chemin. On observe une homogénéisation de l'utilisation des hôtes et une baisse de la consommation sans dégradation notable de la réactivité en régime nominal. La force du dispositif est sa capacité à parcourir des topologies complexes sans supervision centrale forte ; la montée en charge faisant croître la complexité, des heuristiques de filtrage et des critères d'arrêt explicites permettent d'en borner le coût.

Article 4 — Allocation sensible à l'énergie (provisionnement) (Panagiotou et al., 2015). Plutôt que de traiter séparément efficacité d'utilisation et sobriété, l'étude internalise un coût énergétique dès la décision d'allocation et de dimensionnement. Les résultats montrent des gains réguliers et pragmatiques sur l'utilisation des hôtes et la consommation, avec une durée totale d'exécution maîtrisée et une intégration crédible dans une chaîne d'exploitation. Pour élargir la portée, une validation croisée sur plusieurs hyperviseurs et profils de charge, assortie d'une publication détaillée des réglages, consoliderait la généralisabilité.

Article 5 — Réseau défini par logiciel (redistribution de flots) (Kanagavelu et al., 2016). Plutôt que d'ordonner les tâches, l'accent est mis sur les flots : détection des flots volumineux et réacheminement dynamique pour résorber les congestions au sein du centre de données. Les campagnes sur banc d'essai attestent d'une diminution du temps de réponse côté réseau et d'une hausse du débit soutenu en charge. Ce levier, orthogonal à l'ordonnancement de calcul, constitue un complément pertinent lorsque l'étranglement provient des échanges ; des validations sur plusieurs contrôleurs et topologies renforceraient la robustesse externe des conclusions.

Article 6 — Optimisation multi-objectif (compromis énergie–durée) (Kousalya et al., 2018). La décision d'ordonnancement est posée en compromis explicite : minimiser simultanément la consommation énergétique et la durée totale d'exécution, puis sélectionner une solution non dominée selon les priorités. Les fronts obtenus clarifient l'échange « sobriété contre performance » et facilitent un pilotage informé. Si l'approche est opérationnellement attractive, son coût de calcul et sa sensibilité aux opérateurs incitent à des analyses de sensibilité systématiques et à l'annonce de budgets de temps réalistes pour les grandes instances.

Article 7 — Co-optimisation pragmatique (modélisation coût–temps) (Parasha & Soni, 2018). L'ambition est d'obtenir des bénéfices concrets via une modélisation pragmatique de l'ordonnancement (programmation linéaire) qui arbitre explicitement coût et délais, avec des décisions locales faciles à implémenter. Les améliorations sont modestes mais constantes (coût et consommation en baisse, temps de réponse mieux tenu), compatibles avec une intégration industrielle. Une description plus détaillée des paramètres du modèle et des hypothèses de coût renforcerait la réplicabilité et la comparabilité entre bancs d'essai.

Article 8 — Logique floue + fourmis (robustesse à l'incertitude) (Ragmani et al., 2019). Lorsque les mesures de charge sont bruitées ou tardives, l'inférence floue stabilise l'estimation d'état et la stigmergie répartit la charge de manière distribuée. La combinaison tient bien face aux variations rapides : temps de réponse, durée totale d'exécution et utilisation demeurent plus réguliers qu'avec des méthodes plus rigides. À grande échelle, la coordination et la communication pèsent davantage ; réduire la périodicité de coordination ou agréger les messages aide à contenir ce surcoût.

Article 9 — Hybride essaim particulaire–génétique (stabiliser la convergence) (Agarwal & Srivastava, 2018). L'association d'un essaim particulaire, pour préserver la diversité, et d'un algorithme génétique, pour affiner les solutions, accélère et stabilise la recherche de placements de qualité. Les expériences révèlent des diminutions robustes de la durée totale d'exécution et une utilisation mieux lissée par rapport aux versions « pures ». La contrepartie est un paramétrage plus dense et un coût de calcul supérieur ; un protocole de calibration léger et des études de sensibilité publiques en faciliteraient l'adoption.

Article 10 — Étude comparative de référence (bases de comparaison) (Shahid et al., 2023). L'étude cartographie des schémas de base — répartition tournante, stratégies à seuils, heuristiques simples — sur des profils de charge variés afin de fournir un repère neutre. Elle met au jour des zones de fragilité (par exemple, le comportement de certaines politiques face aux charges en rafales) et aide à positionner équitablement de nouvelles propositions. La normalisation des scénarios, la publication des graines et le partage des artefacts renforceraient la comparaison inter-études.

Article 11 — Regroupement flou (hétérogénéité locale) (Huang et al., 2018). Dans des environnements hétérogènes, le regroupement flou sert de boussole locale pour acheminer la charge vers les hôtes les plus adaptés et prévenir la formation de points chauds. Les gains se manifestent surtout par une réactivité locale stabilisée et une utilisation plus régulière, avec une bonne insertion dans des architectures hiérarchiques. Le surcoût de synchronisation entre domaines étant non négligeable, une coordination hiérarchisée ou par fenêtres temporelles en atténue l'impact.

Article 12 — Apprentissage par renforcement tabulaire (décision edge–cloud en temps réel) (Du et al., 2023). Dans un contexte réparti entre périphérie et cloud, les auteurs conçoivent une politique d'allocation fondée sur l'apprentissage par ren-

forcement (Q-learning) afin de décider, à partir d'indicateurs opérationnels (qualité de lien, files d'attente, capacité disponible), où exécuter les requêtes temps réel. Le but est de réduire simultanément le temps de réponse de bout en bout et la saturation des nœuds tout en exploitant au mieux les ressources des deux côtés. Les évaluations montrent une baisse cohérente de la latence et un lissage de l'utilisation par rapport aux heuristiques statiques ; l'efficacité dépend toutefois de la qualité de la télémétrie et d'un apprentissage suffisamment stabilisé pour suivre les changements de trafic.

Article 13 — Décision de déport périphérie–cloud (latence et énergie) (Arcas et al., 2024). Sur le continuum périphérie–cloud, l'algorithme choisit le lieu d'exécution en fonction de la qualité des liaisons et des budgets énergétiques, dans l'objectif de réduire à la fois le temps de réponse de bout en bout et la consommation côté périphérie. Les résultats rapportent des gains conjoints significatifs lorsque les liaisons demeurent stables ; la sensibilité aux variations rapides de qualité de lien invite à agréger plusieurs exécutions et à affiner la télémétrie pour sécuriser la décision.

Article 14 — cloud avec dépendances (compromis réalisables) (Karami, Azizi, & Ahmadizar, 2024). Lorsque les tâches forment des graphes d'exécution avec dépendances et coûts de communication significatifs, l'orchestration doit articuler durée totale des workflows, consommation et échanges. La combinaison d'une heuristique semi-gloutonne et d'une recherche d'ensembles non dominés révèle des compromis réalistes et exploitables pour des applications réparties entre périphérie et cloud. La montée en charge restant l'écueil principal, des décompositions hiérarchiques et des approximations contrôlées aident à tenir des tailles industrielles.

Article 15 — Architecture à agents (montée en charge et résilience) (Swarnakar et al., 2023). Des agents coopératifs répartissent la charge et orchestrent la reprise après incident à travers plusieurs clouds, ce qui accroît la capacité soutenable et la tolérance aux pannes tout en rapprochant la décision des ressources. L'efficacité est notable en contexte multi-fournisseurs ; comme tout système distribué, le coût de coordination (volume de messages, latences de consensus) doit être contenu, par exemple en calibrant la périodicité des décisions et en limitant le trafic de coordination.

Article 16 — Apprentissage par renforcement profond (adaptation à la non-stationnarité) (Swarup et al., 2021). Les auteurs entraînent un agent à partir de

l'état courant du système et de retours de performance afin qu'il s'ajuste lorsque les profils de charge évoluent ; on dépasse ainsi les limites des heuristiques statiques. Les gains en temps de réponse et en utilisation sont probants dès lors que l'entraînement est stabilisé et que la mesure est fiable. Pour faciliter l'évaluation croisée, la publication des hyperparamètres, des graines et d'analyses d'ablation et de sensibilité constitue un atout décisif.

Article 17 — Revue méthodologique (bonnes pratiques d'évaluation) (Zhou et al., 2024). La revue ne propose pas un nouvel algorithme, mais un cadre d'évaluation exigeant : métriques normalisées, bases de référence canoniques, publication des réglages et protocoles de sensibilité. L'objectif est de rendre les comparaisons crédibles et reproductibles, en particulier pour les approches apprenantes. La mise à disposition de bancs de test ouverts et d'artefacts complets (code, traces, scripts) en prolongerait les bénéfices pour la communauté.

Article 18 — Politiques de chemins réseau (complément au calcul) (Shi et al., 2020). Dans un réseau de centre de données piloté par logiciel, l'observation continue de l'état des liens alimente des politiques de chemin qui redistribuent les flots afin d'éviter les goulots. En période de congestion, le temps de réponse côté réseau diminue et le débit soutenu augmente, ce qui profite in fine au service applicatif. L'effet sur la durée totale d'exécution restant indirect, une intégration conjointe avec l'ordonnancement de calcul et des validations à grande échelle sur des topologies variées en consolideraient l'impact.

2.5.1 Comparaison

Les stratégies de consolidation orientées vers l'efficacité énergétique réduisent nettement la consommation et les coûts, tout en préservant une durée totale d'exécution globalement stable ; leur limite récurrente tient toutefois aux pics transitoires de temps de réponse lors des migrations et à la sensibilité au réglage des seuils (Beloglazov & Buyya, 2012 ; Panagiotou et al., 2015). Les méta-heuristiques bio-inspirées (abeilles, fourmis, essaims, hybrides PSO–GA) procurent des gains réguliers sous charges hétérogènes, mais leur performance dépend d'un paramétrage rigoureux et leur coût de calcul croît avec l'échelle (Babu & Krishna, 2013 ; Ragmani et al., 2019 ; Dasgupta et al., 2014 ; Agarwal & Srivastava, 2018). Les approches

apprenantes (réseaux de neurones, renforcement et variantes profondes) sont particulièrement pertinentes en contexte non stationnaire : elles améliorent le temps de réponse et l'utilisation dès lors que la télémétrie est fiable et que l'entraînement est stabilisé, au prix d'un investissement matériel plus élevé (Swarup et al., 2021 ; Zhou et al., 2024). À l'échelle du centre de données, les politiques réseau dans les environnements définis par logiciel agissent surtout sur la réactivité en période de congestion et constituent un complément naturel aux ordonnanceurs côté calcul (Kanagavelu et al., 2016 ; Shi et al., 2020). Enfin, les formulations multi-objectifs (par exemple fondées sur NSGA-II) rendent explicites les arbitrages entre consommation énergétique et durée totale d'exécution, et facilitent une décision alignée sur les priorités (SLA, budget, contraintes opérationnelles) (Kousalya et al., 2018 ; Karami et al., 2024).

Limites méthodologiques et validité. Les protocoles d'évaluation demeurent hétérogènes (simulateurs, traces, jeux de baselines), ce qui borne la comparabilité stricte des résultats. Il est recommandé de normaliser les métriques, de publier les graines d'aléa et de quantifier l'overhead opérationnel (télémétrie, migrations, entraînement) (Shahid et al., 2023 ; Zhou et al., 2024). Nous en tenons compte dans (Tab. 2.2) .

TABLE 2.2 – Comparaison synthétique des approches d'équilibrage de charge

Art.	**Approche**	**Profil de performance (délai / durée globale / efficacité)**	**Coûts**	**Complexité**
1	Consolidation à seuils dynamiques	Temps de réponse globalement stable, mais pics possibles pendant les migrations de VM ; durée totale d'exécution neutre à légèrement réduite ; très efficace pour réduire la consommation énergétique lorsque la charge moyenne reste modulée par des périodes de creux.	I	Moyenne
2	Comportement d'abeilles (HBB-LB)	Temps de réponse plus court en régime de charge fluctuante ; durée totale en baisse grâce au rééquilibrage local des files ; bon désengorgement sans coordination centrale lourde, mais nécessite un réglage fin des paramètres.	III	Moyenne

Suite à la page suivante

Art.	Approche	Profil de performance (délai / durée globale / efficacité)	Coûts	Complexité
3	Colonies de fourmis (consolidation)	Temps de réponse peu affecté ; durée légèrement améliorée via une utilisation plus homogène des hôtes ; décision distribuée adaptée aux topologies complexes, mais la stratégie devient coûteuse à grande échelle.	II	Élevée
4	Logique floue	Temps de réponse plus régulier sous mesures bruitées ; durée réduite de manière modérée ; bonne stabilité quand l'information d'état est imparfaite ; reste raisonnablement intégrable.	III	Moyenne
5	Hybride floue + fourmis	Amélioration notable du temps de réponse et de la durée globale, même avec une télémétrie incertaine ; robuste aux variations rapides, mais la coordination distribuée devient lourde à très grande échelle.	III	Élevée
6	Essaim particulaire (PSO)	Gains légers mais réels sur le délai et la durée ; comportement sensible aux hyperparamètres initiaux, surtout sous charge hétérogène.	II	Moyenne
7	PSO + génétique (hybride)	Temps de réponse et durée mieux tenus que PSO seul ; convergence plus stable et solutions plus régulières ; contrepartie : coût de calcul plus élevé.	II	Élevée
8	Optimisation par baleines (edge/cloud)	Réduction du délai de bout en bout lorsque l'offloading edge/cloud est pertinent ; durée globale améliorée si les liens réseau restent stables ; dépend fortement de la qualité de liaison.	II	Moyenne
9	Optimiseur loup gris (GWO)	Temps de réponse raccourci ; durée totale en baisse ; convergence rapide mais risque d'optima locaux, à surveiller lorsque la charge devient très irrégulière.	II	Moyenne
10	Algorithme des chauves-souris	Améliorations légères et variables sur le temps de réponse et la durée ; comportement fortement dépendant du réglage interne de l'algorithme.	III	Moyenne

Suite à la page suivante

Art.	Approche	Profil de performance (délai / durée globale / efficacité)	Coûts	Complexité
11	Recherche du coucou	Bons résultats surtout en charge irrégulière ; durée globale légèrement réduite et répartition plus équilibrée ; coût de calcul qui peut varier selon le scénario.	III	Moyenne
12	Min–Min amélioré	Temps de réponse réduit ; durée fortement réduite pour les lots de tâches courtes ; très efficace mais peut défavoriser les tâches longues si aucune correction d'équité n'est prévue.	III	Faible
13	Max–Min amélioré	Durée améliorée pour les charges dominées par des tâches longues ; utile quand le workload est composé de gros jobs ; délai moyen comparable au standard.	III	Faible
14	Réseaux de neurones (équilibrage)	Temps de réponse réduit après apprentissage ; durée globale en baisse grâce à une allocation mieux anticipée ; profite des motifs de charge récurrents, mais dépend d'un entraînement coûteux.	IIII	Moyenne
15	Renforcement (Q-learning)	Temps de réponse et durée améliorés dans un contexte non stationnaire ; s'adapte en ligne en fonction de l'état du système ; nécessite une exploration bien contrôlée et une récompense bien définie.	IIII	Moyenne
16	Renforcement profond	Bons gains si l'entraînement est stabilisé et la télémétrie riche ; durée globale réduite et comportement robuste sous variations rapides de charge ; tuning délicat.	IIII	Élevée
17	NSGA-II (multi-objectif)	Délai souvent neutre ; durée totale optimisée suivant le compromis choisi ; permet d'arbitrer explicitement entre performance et consommation énergétique selon les priorités opérationnelles.	II	Élevée

Suite à la page suivante

Art.	Approche	Profil de performance (délai / durée globale / efficacité)	Coûts	Complexité
18	Politique de chemins réseau (SDN)	Temps de réponse réseau réduit en situation de congestion et débit soutenu plus élevé ; la durée applicative reste globalement inchangée, ce qui en fait surtout un complément utile de l'ordonnancement côté calcul.	III	Moyenne

Coûts : I = très faible ; II = faible à modéré ; III = modéré ; IIII = élevé).

2.6 Conclusion

Dans ce chapitre, nous avons exploré le rôle central de l'ordonnancement dans le cloud computing. L'ordonnancement des tâches ne consiste pas seulement à décider où exécuter une requête : il vise à utiliser les ressources de manière efficace, à réduire les temps d'attente et à maintenir un niveau de performance stable malgré la variation de la charge. Nous avons présenté les principaux types d'ordonnancement, leurs principes et leurs objectifs, puis étudié différentes approches d'équilibrage de charge proposées dans la littérature. Chaque méthode cherche un compromis spécifique entre qualité de service, consommation de ressources et montée en charge du système. Les enseignements tirés ici serviront de base au chapitre suivant, qui précisera le modèle retenu et la stratégie d'ordonnancement que nous adoptons.

Chapitre 3

Modélisation et Résolution du Problème

Introduction

Nous avons des tâches à accomplir et des serveurs prêts à les réaliser. Chaque tâche a une date limite à laquelle elle doit être terminée et une taille qui représente le temps estimé pour sa réalisation. Chaque serveur a ses propres caractéristiques , comme la vitesse de calcul, la mémoire et la bande passante, mais dans ce cas, nous allons les considérer égales. Notre objectif est de décider quelles tâches assigner à quels serveurs afin de terminer le maximum de tâches à effectuer dans les délais et en utilisant efficacement les serveurs, c'est à dire qu'il n'y ait pas de perte de temps d'attente trop long entre les affectations. Nous devons nous assurer que chaque tâche est assignée à un seul serveur et que ce serveur peut la compléter avant la date limite.Ce problème se rapproche plus d'un problème d'ordonnancement

3.1 Modélisation mathématique

L'objectif de cette phase de modélisation est de donner une abstraction mathématique du problème posé. Pour cela, nous identifions les objets à manipuler (ressources, périodes de temps, . . .), les données associées aux objets (capacité des ressources, . . .), les variables de décision à fixer par rapport aux objets afin de proposer une solution au problème, les contraintes à satisfaire par les décisions afin de définir des solutions réalisables, et enfin la fonction objectif qui permet d'évaluer les solutions réalisables,

et de sélectionner la solution optimale parmi les solutions réalisables.
Sans perte de généralité, on supposera que le nombre n de tâches à réaliser est strictement supérieur au nombre m de serveurs disponibles, et que tous les serveurs possèdent les mêmes capacité de traitement (CPU), de mémoire RAM, et de bande passante (BW).

Notations

Considérons les notations suivantes :

$t = \{t_1, t_2, \ldots, t_n\}$: l'ensemble des tâches à réaliser.

$s = \{s_1, s_2, \ldots, s_m\}$: l'ensemble des serveurs.

$l_i \geq 0$: la taille de la tâche t_i, $i = 1, \ldots, n \times h$.

$d_i \geq 0$: la deadline de la tâche t_i, $i = 1, \ldots, n \times h$.

$T_j = \sum_{i=1}^{n} l_i x_{ij}$, $j = 1, \ldots, m$: le temps nécessaire au serveur s_j pour effectuer les tâches qui lui sont assignées.

h : le nombre d'histoires (une histoire, sous entend une affectation des m premières taches parmi les n taches disponibles, aux m serveurs).

Notons :

$$x_{ij} = \begin{cases} 1, & \text{si la tâche } t_i \text{ est affectée au serveur } s_j \text{ ;} \\ 0, & \text{sinon.} \end{cases} \tag{3.1}$$

Contraintes

Étant données que les tâches ne sont pas de type disjonctives (les tâches t_i et t_j peuvent être réalisées simultanément), chaque tâche doit être complétée avant sa deadline, et doit être assignée à un seul serveur, on a :

$$l_i + s_j \leq d_i \; i = \overline{1, n}, \; j = \overline{1, m} \tag{3.2}$$

$$\sum_{j=1}^{m} x_{ij} = 1, \quad \forall i = 1, \ldots, n \tag{3.3}$$

fonction objectif

$$\min \quad var(T = (\frac{1}{m} \sum_{i=1}^{m} T_j) \tag{3.4}$$

Le problème posé se formule ainsi comme un programme linéaire en nombres entiers et mixtes suivant :

$$\min \quad var(T = (\frac{1}{m}\sum_{i=1}^{m} T_j) \tag{3.5}$$

$$l_i + s_j \leq d_i, \quad i = \overline{1,n},\ j = \overline{1,m} \tag{3.6}$$

$$\sum_{j=1}^{m} x_{ij} = 1, \quad \forall i = 1, \ldots, n \tag{3.7}$$

$$x_{ij} \in \{0,1\},\ l_i \geq 0,\ d_i \geq 0. \tag{3.8}$$

Le problème abordé est à la fois complexe et complet. Bien que les méthodes exactes soient très efficaces pour résoudre des petites instances, elles deviennent impraticables pour des problèmes de grande taille en raison de leur complexité . C'est pourquoi nous nous tournons vers des heuristiques. Certes, les heuristiques ne garantissent pas toujours des solutions optimales, mais elles permettent d'obtenir des résultats proches de l'optimum en un temps raisonnable. Cela nous permet de gérer efficacement l'ordonnancement des tâches, même dans des environnements à grande échelle, où la rapidité et l'efficacité sont cruciales.

3.2 Schéma et Implémentation de l'Algorithme

3.2.1 Etape 1 :Initialiser les serveurs

Chaque serveur a un numéro unique, une capacité initiale de traitement, une occupation initiale, et une liste de tâches affectées.

```
% Initialisation des serveurs
s = table((1:m)', zeros(m, 1), zeros(m, 1), cell(m, 1), ...
'VariableNames', {'NumS', 'S', 'Occ', 'Tasks'});
```

FIGURE 3.1 – Initialisation des Serveurs

m : Nombre de serveurs.

La table s de la figure (3.1) est initialisée avec les colonnes suivantes :

– $NumS$:Numéro du serveur.

– T :le temps nécessaire au serveur j pour effectuer les tâches qui lui sont assignées. (initialisé à zéro).
– Occ : Indicateur d'occupation (initialisé à zéro).
– $Tasks$: Liste des tâches affectées (initialisée vide).

3.2.2 Etape 2 : Générer les Tâches

Nous avons utilisé une fonction une fonction $GenT(m, k)$ qui prend en paramètres le nombre de tâches(m) par histoire(h) et le numéro de l'histoire afin de générer les tâches pour une histoire donnée. Chaque tâche a un numéro, une durée aléatoire et une deadline aléatoire basée sur sa durée.

```
function t = GenT(m, k)

    b = k * m + 1;
    e = (k + 1) * m;
     numT = (b:e)';
     l = randi([20000, 100000], m, 1) /80000;
     d = l + randi([0, 50], m, 1);
     t = table(numT, l, d, 'VariableNames', {'numT', 'l', 'd'});
end
```

FIGURE 3.2 – Génération des tâches

Paramètres :

– m : Nombre de tâches par histoire.
– k : Numéro de l'histoire.
– n : Nombre total tâches $n = h * m$.

Variables :

– $numT$: Numéros des tâches (de 1 à n).
– l : Durées des tâches (exprimée en secondes).
– d : Deadlines des tâches (durées plus un décalage aléatoire entre 0 et 100).

Les tailles des tâches sont exprimées en millions d'instructions(**MI**). La durée d'une tâche t_i, notée l correspond au rapport de sa taille sur la capacité du serveur c_i auquel elle est affectée. Dans notre cas, nous avons pris des serveurs identiques avec une capacité de 80 000 millions d'instructions par seconde (**MIPS**). Pour obtenir la durée des tâches en secondes, il suffit de diviser la taille des tâches par la capacité

des serveurs :

$$l_i = \frac{taille(t_i)}{c_i}.$$

3.2.3 Etape 3 : Ordonner les Tâches et les Serveurs

pour l'ordonnancement nous avons utilisé la fonction $ORD(t, s)$ qui permet de trier les tâches par durée croissante et les serveurs par ordre décroissante.

```
function[t, s] = ORD(t ,s)
t=sortrows(t,'l','ascend');
s=sortrows(s,'S','descend');
end
```

FIGURE 3.3 – Ordonnancement des tâches et Serveurs

- Les tâches sont triées par ordre croissant de durée (l).
- Les serveurs sont triés par ordre décroissant de capacité (T).

3.2.4 Etape 4 : Affecter les Tâches aux Serveurs

Pour cette étape,La fonction $Affecte(m, k, s)$ assigne les tâches aux serveurs en respectant les contraintes de temps et de capacité.La fonction prend en paramètres le nombre de tâches(m), le numéro de l'histoire (h), et les serveurs (s).

```
function s = Affecte(m,k,s)
    t = GenT(m, k);
    [t, s] = ORD(t ,s);
       for i = 1:m
        for j = 1:m
            if t.l(i) + s.S(j) <= t.d(i) && s.Occ(j) == 0
                s.S(j) = s.S(j) + t.l(i);
                s.Occ(j) = 1;
                s.Tasks{j} = [s.Tasks{j}, t.numT(i)];
                break;
            end
        end
       end
        s.Occ(:) = 0;
end
```

FIGURE 3.4 – Procédure d'affectation

Pour chaque tâche i et chaque serveur j :

– Si la somme de la durée de la tâche $(t.l(i))$ et de la capacité actuelle du serveur $(s.S(j))$ est inférieure ou égale à la deadline de la tâche $(t.d(i))$, et que le serveur n'est pas déjà occupé $(S.Occ(j) == 0)$, alors :
  – La capacité du serveur est mise à jour $(s.T(j) = s.T(j) + t.l(i))$.
  – Le serveur est marqué comme occupé $(s.Occ(j) = 1)$.
  – La tâche est ajoutée à la liste des tâches du serveur $(s.Tasksj = [s.Tasksj, t.numT(i)])$.
  – La boucle se termine pour cette tâche.

3.2.5 Etape 5 : Résultats

Sur le script on initialise les serveurs, exécute l'affectation des tâches pour chaque histoire, et affiche les résultats finaux.

```
function main()
    % Initialisation des serveurs
s = table((1:m)', zeros(m, 1), zeros(m, 1), cell(m, 1), ...
    'VariableNames', {'NumS', 'S', 'Occ', 'Tasks'});

    % Boucle pour chaque histoire
      for k = 0:(h-1)
     % Appeler la fonction d'affectation qui génère et affecte les tâches
        s = Affecte(m, k, s);
      end
    s=sortrows(s,'NumS','ascend');
    final_table = s(:, {'NumS', 'Tasks','S'});
    disp('Tableau final des serveurs et des tâches affectées :');
    disp(final_table);
end
```

FIGURE 3.5 – Affichage des résultats

– La fonction initialise m serveurs.
– Pour chaque histoire k (de 0 à $h-1$), elle génère et affecte les tâches en utilisant la fonction Affecte.
– Après avoir traité toutes les histoires, les serveurs sont triés par numéro $(NumS)$ et le tableau final des serveurs et des tâches affectées est affiché.

3.3 Application

Dans ce qui suit, nous allons présenter l'application de l'algorithme pour un certain nombre de tâches. Nous avons exécuté l'algorithme sur différentes instances

afin de tester son efficacité et sa performance dans la gestion de l'ordonnancement des tâches. Les résultats obtenus sont résumés dans le tableau ci-dessous :

m	h	$t-aff$	n	$T-moy-aff$	$Variance$	$t-execution-algo$
5	3	15	15	2.6165	0.10165	0.006756
	5	24	25	3.5025	0.34367	0.007609
	10	49	50	6.4999	0.39013	0.012871
10	3	30	30	2.33	0.02723	0.008132
	5	49	50	3.6999	0.056359	0.010941
	10	96	100	7.4204	0.085014	0.018576
50	3	150	150	2.234	0.056855	0.04809
	5	242	250	3.6045	0.14232	0.076971
	10	480	500	7.0825	0.93634	0.14164

TABLE 3.1 – Variance en fonction du nombre de tâches et d'histoires

Le tableau (3.1) présente une comparaison des performances du serveur en fonction du nombre m de tâches par histoire et du nombre h d'histoires. Les colonnes affichent le temps moyen (T) nécessaire pour exécuter toutes les tâches attribuées aux serveurs, le nombre de tâches affectées $(t-aff)$, le nombre total n de tâches, ainsi que la variance et le temps d'exécution de l'algorithme $(t-execution-algo)$.

Résultats et Discussion

En faisant varier le nombre m de serveurs :

1. Pour $(m=5)$, on remarque que le nombre total de tâches augmente proportionnellement avec le nombre d'histoires. Le temps moyen d'exécution $(T-moy-aff)$ montre une augmentation significative, particulièrement notable pour $(h=10)$.La variance augmente également, indiquant une dispersion croissante des temps d'exécution. Le temps d'exécution de l'algorithme augmente
2. Pour le cas $(m=10)$, nous observons une augmentation progressive de la variance à mesure que h et le nombre de tâches affectées augmente. Le temps

d'exécution de l'algorithme suit une tendance à la hausse, devenant plus long pour $(h = 10)$.

3. Pour $(m = 50)$,L'augmentation du nombre d'histoires (h) entraîne une augmentation significative du nombre de tâches affectées et du temps moyen d'exécution. La variance montre une augmentation importante, indiquant une grande dispersion des temps d'exécution. Le temps d'exécution de l'algorithme devient également beaucoup plus long.

Après discussion des résultats, nous retenons les remarques suivantes :

Remarque 3.3.1 *Variance Globale : Les fluctuations dans la variance des temps d'exécution en fonction du nombre de tâches et d'histoires indiquent que la complexité de la gestion des tâches augmente avec leur nombre. Cela souligne l'importance de mécanismes efficaces de répartition des charges.*

Remarque 3.3.2 *Stabilité et Efficacité : Une faible variance est souvent un signe de stabilité et d'efficacité dans le traitement des tâches, ce qui est essentiel pour maintenir une qualité de service élevée.*

Remarque 3.3.3 *Temps d'Exécution : Les temps d'exécution de l'algorithme , bien que variant selon les configurations, tendent à augmenter avec le nombre d'histoires, surtout pour un nombre important de tâches de* (n) *. Cela montre que la gestion des tâches devient plus complexe et nécessite plus de temps à mesure que la charge de travail augmente.*

Remarque 3.3.4 *Le tableau (3.1) a mis en évidence l'importance de la gestion efficace de la charge de travail, en particulier lorsque le nombre de tâches et d'histoires augmente, afin de minimiser la variabilité dans les temps de traitement et d'optimiser les performances globales du système.*

Variance en fonction du nombre de serveurs

La figure (3.6) illustre la variance des temps d'exécution en fonction du nombre de serveurs (m) avec $(h = 5)$. On observe que la variance présente des fluctuations

marquées à mesure que le nombre de serveurs augmente.

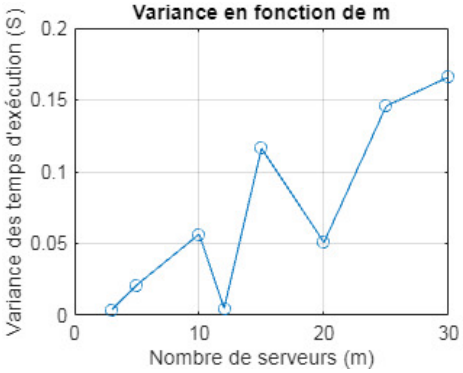

FIGURE 3.6 – Variance en fonction du nombre de serveurs m.

Commentaire 1 *Les fluctuations de la variance en fonction du nombre de serveurs montrent la capacité de l'algorithme à s'adapter à différents niveaux de charge. Les variations observées indiquent des points de transition où l'algorithme peut être encore optimisé pour améliorer l'allocation des ressources. Cette adaptabilité met en lumière la flexibilité et la robustesse de l'algorithme dans la gestion des tâches.*

Commentaire 2 *Ces variations révèlent le potentiel de l'algorithme à maintenir une performance stable même en présence de fluctuations. Elles indiquent également des opportunités pour des optimisations futures, renforçant ainsi la capacité du système à gérer efficacement des environnements de cloud computing complexes. L'algorithme démontre une aptitude remarquable à ajuster la répartition des tâches pour maximiser l'efficacité et la performance.*

Variance en fonction du nombre d'histoires

La figure (3.7) montre la variance en fonction du nombre d'histoires (h) avec m fixé à 15. On observe une tendance générale à l'augmentation de la variance avec l'augmentation du nombre d'histoires.

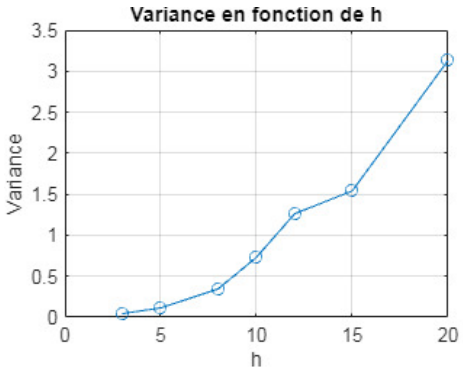

FIGURE 3.7 – Variance en fonction du nombre d'histoires h.

Commentaire 3 *La variance augmente progressivement avec le nombre d'histoires, ce qui montre que l'algorithme peut s'ajuster de manière efficace même à mesure que la complexité des tâches augmente. La montée graduelle de la variance reflète une capacité à gérer une charge croissante tout en maintenant une stabilité relative jusqu'à des niveaux très élevés d'histoires.*

Commentaire 4 *L'augmentation de la variance avec le nombre d'histoires met en évidence la résilience et la capacité de l'algorithme à optimiser la gestion des tâches dans des situations de complexité croissante. Ces résultats indiquent non seulement une gestion efficace des ressources mais aussi un potentiel important pour des optimisations supplémentaires, garantissant une amélioration continue de la performance et de la fiabilité du système.*

Conclusion

Ce troisième chapitre a permis d'explorer en détail la modélisation et la résolution du problème d'ordonnancement des tâches dans le cloud computing. Grâce à un algorithme spécifique, nous avons démontré comment les tâches peuvent être efficacement attribuées aux serveurs pour optimiser l'utilisation des ressources et respecter les délais. Les étapes de l'algorithme ont été clairement définies et implémentées en MATLAB, offrant ainsi une solution pratique et adaptable à divers scénarios de cloud computing. Les résultats obtenus confirment la viabilité de notre approche, qui peut être ajustée selon les besoins spécifiques des environnements de cloud. Ces avancées permettent non seulement de maximiser l'efficacité opérationnelle, mais aussi de garantir une meilleure qualité de service pour les utilisateurs finaux.

Conclusion Générale

En résumé, l'ordonnancement dans le cloud computing est un domaine dynamique nécessitant une compréhension approfondie et une application judicieuse des diverses méthodes disponibles. Des solutions efficaces d'ordonnancement sont cruciales pour améliorer la performance des systèmes cloud et répondre aux besoins croissants des utilisateurs.

Ce mémoire a exploré l'équilibrage de charge dans le cloud computing, en mettant en lumière les défis et les solutions innovantes pour optimiser la répartition des tâches et améliorer la performance des systèmes cloud. Nous avons commencé par une analyse exhaustive du cloud computing, de ses origines, de son évolution et de ses caractéristiques essentielles. Les modèles de service et de déploiement ont été présentés pour fournir un cadre conceptuel clair et compréhensible.

Le deuxième chapitre s'est concentré sur les mécanismes de gestion des ressources, en particulier sur l'ordonnancement des tâches, crucial pour maintenir l'efficacité et la performance dans un environnement de cloud computing. Nous y avons comparé plusieurs travaux de la littérature et étudié différentes approches d'équilibrage de charge, chacune recherchant un compromis spécifique entre qualité de service, consommation de ressources et montée en charge du système.

Enfin, le troisième chapitre a présenté une modélisation mathématique détaillée du problème d'équilibrage de charge et a proposé un algorithme de résolution implémenté sous MATLAB. Cette approche a été validée par des analyses comparatives et des tableaux de performance, montrant l'efficacité de l'algorithme dans divers scénarios de charge de travail.

L'ensemble de ce travail souligne l'importance de l'équilibrage de charge pour une utilisation optimale des ressources, la minimisation des temps d'attente et la garantie d'une performance stable dans le cloud computing. Les techniques et algorithmes présentés offrent des solutions pratiques et efficaces aux défis actuels, tout en ou-

vrant la voie à des recherches futures dans ce domaine en constante évolution.

En conclusion, ce mémoire apporte une contribution significative à la compréhension des défis et des solutions en matière d'équilibrage de charge dans le cloud computing. Il fournit un cadre pour l'amélioration continue des systèmes cloud, en mettant en lumière les innovations actuelles et les perspectives futures.

Bibliographie

Agarwal, M., & Srivastava, G. M. S. (2018). Genetic algorithm-enabled particle swarm optimization (PSOGA)-based task scheduling in cloud computing environment. *International Journal of Information Technology & Decision Making*, *17*(05), 1589–1611.

Arcas, G. I., et al. (2024). Whale optimization for cloud–edge offloading decision. *Biomimetics*.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., et al. (2010). A view of cloud computing. *Communications of the ACM*, *53*(4), 50–58.

Babu, L. D. D., & Krishna, P. V. (2013). Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied Soft Computing*, *13*(5), 2292–2303.

Beloglazov, A., & Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, *28*(5), 755–768.

Carlier, J., & Chrétienne, P. (1988). *Problèmes d'ordonnancement : Modelisation, complexité, algorithmes*. Paris : Masson.

Chopra, R. (2018). *Cloud computing : A self-teaching introduction*. Boston, Massachusetts : Mercury Learning and Information.

Dasgupta, K., Mandal, B., Dam, S., et al. (2014). A genetic/ant colony based scheduling for load balancing in cloud environments. In *Proceedings of iccct / icccs*.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2001). Multi-objective optimization using evolutionary algorithms. *International Journal of Approximate Reasoning*, *2*(3), 345–365.

Du, Z., et al. (2023). A Q-learning-based load balancing method for real-time service between edge and cloud. *Electronics*, *12*(15), 3254.

Garg, S. K., Versteeg, S., & Buyya, R. (2013). Survey on fault tolerance techniques

in cloud computing. *Journal of Network and Computer Applications*.

Ghanbari, S., & Othman, M. (2012). A priority based job scheduling algorithm in cloud computing. *Procedia Engineering*, *50*, 778–785.

Gupta, D., & Maravelias, C. T. (2019). Online scheduling : Understanding the impact of uncertainty. *IFAC-PapersOnLine*, *52*(1), 727–732.

Huang, W., et al. (2018). Fuzzy clustering with feature weight preferences for load balancing in cloud. *International Journal of Information Technology & Decision Making*, *17*(03), 897–924.

Kanagavelu, R., et al. (2016). Software-defined load balancer in cloud data centers. In *Iccip*.

Karami, S., Azizi, S., & Ahmadizar, F. (2024). A bi-objective workflow scheduling in virtualized fog–cloud computing using NSGA-II with semi-greedy initialization. *Applied Soft Computing*, *150*, 111086.

Kousalya, E., et al. (2018). Multi-objective task scheduling to minimize energy and makespan using NSGA-II. *Journal of Network and Systems Management*.

Lisdorf, A. (2021). *Cloud computing basics : A non-technical introduction*. Copenhagen, Denmark : Apress.

NIST. (2014). *The NIST definition of cloud computing* (Special Publication N° SP-800 145). National Institute of Standards and Technology.

Panagiotou, N., et al. (2015). Energy-aware management of virtual machines in cloud data centers. In *Pci 2015*.

Parasha, A., & Soni, M. (2018). Cost-effective task scheduling for cloud computing using linear programming. *Journal of Cloud Computing*, *7*(1), 1–15.

Pinedo, M. L. (2016). *Scheduling : Theory, algorithms, and systems*. Springer.

Priore, P., Gomez, A., Pino, R., & Rosillo, R. (2014). Dynamic scheduling of manufacturing systems using machine learning : An updated review. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *28*(1), 83–97.

Production, C. de Recherche de l'École de. (2010). *Cloud computing : Définitions et concepts, enquête et analyse des tendances*. France : C. d. R. d. É. de Production.

Ragmani, A., et al. (2019). An improved hybrid fuzzy–ant colony algorithm for cloud load balancing. *Procedia Computer Science*, *151*, 1047–1054.

Shahid, M. A., et al. (2023). Performance evaluation of load-balancing algorithms with cloud analyst. *Applied Sciences*, *13*(3), 1586.

Shi, X., et al. (2020). An openflow-based load balancing strategy in software-defined networks. *Computers, Materials & Continua*, *63*(2), 845–861.

Swarnakar, S., et al. (2023). A multi-agent-based VM migration for dynamic load balancing in cloud data centers. *Journal of Cloud Computing*.

Swarup, S., et al. (2021). Task scheduling in cloud using deep reinforcement learning. In *Procedia computer science* (Vol. 193, pp. 173–182). (Double DQN pour lordonnancement)

Vacca, J. R. (2016). *Cloud computing security : Foundations and challenges*. Boca Raton, FL : CRC Press.

Vouk, M. A. (2008). Cloud computing—issues, research and implementations. *Journal of Computing and Information Technology*, *16*(4), 235–246.

Xu, Q., & Li, T. (2013). Task scheduling in cloud computing : A review. *Journal of Supercomputing*, *66*(3), 1709–1732.

Xue, Y., Li, B., & Nahrstedt, K. (2001). A scalable location management scheme in mobile ad-hoc networks. In *Ieee international conference on communications*.

Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing : state-of-the-art and research challenges. *Journal of Internet Services and Applications*.

Zhou, G., Tian, W., Buyya, R., et al. (2024). Deep reinforcement learning-based methods for resource scheduling in cloud computing : A review. *Journal of Artificial Intelligence Research*.

Résumé

Le cloud computing a révolutionné la gestion des ressources informatiques, offrant une infrastructure flexible, évolutive et accessible via Internet. Ce mémoire examine le problème de l'équilibrage de charge dans le cloud computing, en explorant diverses méthodes et algorithmes pour assurer une distribution efficace des charges de travail. L'objectif est de minimiser les temps d'attente, maximiser l'utilisation des ressources et garantir la qualité de service. Nous avons analysé les avantages et les inconvénients des modèles de service et de déploiement du cloud, ainsi que les approches d'ordonnancement et leurs applications pratiques. Une modélisation mathématique détaillée et un algorithme de résolution implémenté sous MATLAB sont également présentés, accompagnés d'analyses comparatives et de tableaux de performance.

Abstract

Cloud computing has revolutionized the management of computing resources by offering a flexible, scalable, and Internet-accessible infrastructure. This thesis examines the problem of load balancing in cloud computing, exploring various methods and algorithms to ensure efficient workload distribution. The goal is to minimize waiting times, maximize resource utilization, and ensure quality of service. We analyze the advantages and disadvantages of cloud service and deployment models, as well as scheduling approaches and their practical applications. A detailed mathematical model and a MATLAB based solution algorithm are also presented, accompanied by comparative analyses and performance tables.