Vers une approche intelligente de placement de données dans un cloud distribué basé sur un système de stockage hybride

Amina Chikhaoui

DOCTORAT BRETAGNE LOIRE / MATHSTIC

Université de Bretagne Occidentale

USTHB

THÈSE DE DOCTORAT EN COTUTELLE INTERNATIONALE

DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

AVEC L'UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE (ALGERIE)
Laboratoire des Systèmes Informatiques
Spécialité : Informatique

Par
Amina CHIKHAOUI

Vers une approche intelligente de placement de données dans un cloud distribué basé sur un système de stockage hybride

Thèse présentée et soutenue à USTHB, le 18 Avril 2022
Unité de recherche : LabSTICC / LSI

Rapporteurs avant soutenance :

Olivier BARAIS	Professeur à l'U. Rennes 1, IRISA
Abdelhafid ZITOUNI	Professeur à l'U. Constantine 2

Composition du Jury :

Président :	Madame Hassina NACER	Professeure à l'USTHB
Examinateurs :	Monsieur Olivier BARAIS	Professeur à l'U. Rennes 1, IRISA
	Monsieur Abdelhafid ZITOUNI	Professeur à l'U. Constantine 2
Dir. de thèse :	Monsieur Kamel BOUKHALFA Professeur à l'USTHB / Monsieur Jalil BOUKHOBZA Professeur à l'ENSTA-Bretagne	

Invité(s) :
Laurent LEMARCHAND Maître de conférence à l'UBO

REMERCIEMENTS

Je tiens à remercier tout d'abord Dieu de m'avoir donné la force et la patience d'accomplir ce modeste travail.

Ma sincère reconnaissance et mes chaleureux remerciements s'adressent à mes trois encadrants pour avoir accepté de m'encadrer, pour leur disponibilité, pour leur encouragement et patience, pour leur démarche fructueuse de travail, pour leur soutien aussi bien sur le plan scientifique que humain. Un grand merci à :

- Monsieur **Kamel BOUKHALFA**, mon directeur de thèse, pour son travail de suivi, pour ses remarques enrichissantes et ses précieuses orientations qui m'ont certainement permis de mener à bien mon travail de thèse.
- Monsieur **Jalil BOUKHOBZA**, mon co-directeur de thèse, pour m'avoir guidé tout au long de mon travail, pour sa rigueur et la pertinence de ses jugements qui ont été très constructives et pour ses précieux conseils. Merci pour son accueil au sein du laboratoire.
- Monsieur **Laurent LEMARCHAND**, mon co-encadrant de thèse, pour son appui scientifique, pour sa pédagogie et ses remarques très pointues qui ont contribué à améliorer la qualité de ce travail.

Je tiens à remercier très sincèrement Madame **Hassina NACER** pour m'avoir fait l'honneur d'accepter de présider le jury de cette thèse et accorder du temps pour le jugement de mon travail. Que Monsieur **Abdelhafid ZITOUNI** et Monsieur **Olivier BARAIS** trouve ici l'expression de mes vifs remerciement pour avoir accepter d'être examinateurs et consacrer un temps précieux à l'évaluation de mon manuscrit de thèse.

Je tiens à remercie mes amies et particulièrement **Djamila, Imene, Safia, Ryma, Zakia, Ibtissem, Dihia, Zeyneb et Yasmina** pour leurs soutiens et encouragements incessants.

Cette thèse est en particulier dédiée à la mémoire de **ma grand-mère**.

Enfin, j'adresse mes profonds remerciements à **ma famille** et particulièrement à **mes chers parents, mes frères et ma sœur** pour leur amour et soutien inconditionnels.

Titre : Vers une approche intelligente de placement de données dans un cloud distribué basé sur un système de stockage hybride

Mots clés : Informatique en nuages, Fédération de *Clouds*, Placement de données, Réplication de données, Optimisation, Matheuristique.

Résumé : a fédération de *Clouds* permet d'étendre de manière transparente les ressources des fournisseurs de services *Cloud* (Cloud Service Provider : CSP) afin de fournir une meilleure qualité de service (Quality of Service : OS) aux clients sans frais de déploiement supplémentaires.

Le stockage en tant que service (STorage as a Service : STaaS), constitue l'un des principaux services *Cloud* offerts aux clients. Pour un tel service, la performance des Entrée/Sortie (E/S) des supports de stockage et la latence du réseau sont parmi les métriques les plus importantes considérées par les clients. En effet, le système d'E/S prend environ 90% du temps d'exécution d'une transaction pour certaines requêtes de base de données. Afin de rassurer les clients, certaines sociétés de *Cloud* incluent déjà des garanties de latence dans leurs contrats de niveau de service (Service Level Agreement : SLA) et les clients peuvent payer des frais supplémentaires pour réduire davantage les latences.

Cette thèse traite du problème de placement de données pour un CSP faisant partie d'une fédération. En effet, offrir des services attractifs et peu coûteux est un grand défi pour les CSP. Notre but est de proposer des approches intelligentes pour un meilleur placement des données qui minimise le coût pour le fournisseur tout en satisfaisant les clients. Cette approche doit prendre en compte l'hétérogénéité des ressources de stockage interne et externe en terme de plusieurs paramètres (comme la capacité, les performances, la tarification) ainsi que les caractéristiques des clients et leurs exigences. En dépit du fait que de nombreuses stratégies de placement de données ont été proposées pour les systèmes de stockage hybrides, elles ne sont pas généralisables pour toutes les architectures. En effet, une stratégie de placement doit être conçue selon l'architecture du système pour laquelle elle est proposée et les objectifs visés.

Les stratégies d'optimisation dans les environnements *Cloud* sont, de manière générale, basées sur le coût. En effet, réduire les coûts opérationnels tout en maintenant des niveaux élevés de garantie de service pour les clients sont des facteurs importants pour que les fournisseurs augmentent leurs revenus et restent compétitifs. Ainsi, notre première contribution est relative à l'évaluation du coût de placement de données dans un *Cloud* fédéré. Le modèle proposé prend en compte différents facteurs modélisant les caractéristiques physiques et fonctionnelles du système fédéré.

Afin d'optimiser le placement de données, les coûts de stockage, de migration et de latence doivent être pris en compte. Ces coûts sont corrélés et s'avèrent, dans certains cas, contradictoires. Il convient alors de trouver un compromis entre ces coûts. Par conséquent, une optimisation multi-objectifs est nécessaire. Dans cette perspective, notre deuxième contribution consiste en la proposition d'une approche multi-objectifs basée sur la méta-heuristique évolutionnaire Non Dominated Sorting Genetic Algorithm version II (NSGAII). L'approche développée CDP-NSGAII$_{IR}$ *(a Constraint Data Placement matheuristic based on NSGAII with Injection and Repair functions)* est une matheuristique de placement de données qui consiste à hybrider une méthode exacte avec la méta-heuristique NSGAII. Un opérateur de réparation a été conçu pour rendre les solutions réalisables au regard des contraintes du système.

Enfin, la troisième contribution de ce travail de thèse concerne la réplication de données. En effet, fournir des services fiables avec une haute disponibilité des données et des performances adéquates sont des exigences clés qui doivent être satisfaites. Le concept de réplication est utilisé pour garantir de telles exigences. Pour résoudre le problème de placement et de sélection des répliques, nous avons proposé *StorNIR* (*a cost-efficient data object* ***Stor****ing scheme based on* ***N****SGAII upgraded with* ***I****njection and* ***R****eparation operators*), une stratégie de stockage d'objets basée sur NSGAII et mise à niveau avec des opérateurs d'injection et de réparation. Le même raisonnement de la deuxième contribution a été repris dans ce travail. En effet, nous avons utilisé la matheuristique combinant une méthode exacte avec NSGAII pour résoudre le problème de placement et de sélection des répliques. Il est à noter que nous avons mis à jour les opérateurs d'évolution de NSGAII pour les adapter au problème de réplica-

tion. De plus, lorsque les objets sont répliqués, la latence réseau des requêtes d'accès peut être optimisée. Par conséquent, dans l'approche matheuristique nous avons proposé un algorithme pour optimiser la latence des requêtes. Il consiste à affecter les requêtes à l'emplacement le plus proche qui satisfait les performances E/S de stockage.

Title : Towards an intelligent approach for data placement in a distributed Cloud based on a hybrid storage system

Keywords : Cloud computing, Cloud federation, Data placement, Data replication, Optimization, Matheuristic.

Abstract : *Cloud* federation makes it possible to seamlessly extend the resources of Cloud Service Provider (CSP) in order to provide a better Quality of Service (QoS) to customers without additional deployment costs.

STorage as a Service (STaaS), is one of the main *Cloud* services offered to customers. For such a service, storage Input/Output (I/O) performance and network latency are among the most important metrics considered by customers. In effect, transactions for some database queries spend 90% of the execution time in I/O operations. In order to satisfy customers, some *Cloud* companies already include latency guarantees in their Service Level Agreement (SLA) and customers can pay additional fees to further reduce latencies.

This thesis addresses the data placement problem for a CSP that is part of a federation. Indeed, offering attractive and inexpensive services is a big challenge for CSP. Our goal is to provide intelligent approaches for a better data placement that minimizes the cost of placement for the provider while satisfying the customers QoS requirements. This approach must take into account the heterogeneity of internal and external storage resources in terms of several parameters (such as capacity, performance, pricing) as well as customer characteristics and requirements. Despite the fact that many data placement strategies have been proposed for hybrid storage systems, they are not generalizable to every architecture. Indeed, a placement strategy must be designed according to the system architecture for which it is proposed and the target objectives.

Optimization strategies for *Cloud* environments are, in general, cost-based. Indeed, reducing operational costs while maintaining high levels of customer satisfaction are important factors for providers to increase their revenue and remain competitive. Thus, our first contribution relates to the evaluation of the data placement cost in a federated *Cloud*. The proposed model takes into account different factors modeling the physical and functional characteristics of the federated system.

In order to optimize data placement, storage, migration, and latency costs must be considered. These costs are correlated and, in some cases, conflicting. It is then necessary to operate a tarde-off between them. Therefore, multi-objective optimization is necessary. In this perspective, our second contribution consists in proposing a multi-objective approach based on the evolutionary metaheuristic NSGAII. The developed approach CDP-NSGAII$_{IR}$ *(a Constraint Data Placement matheuristic based on NSGAII with Injection and Repair functions)* is a data placement matheuristic which consists in hybridizing an exact method with the metaheuristic NSGAII. A repair operator has been designed to make the solutions feasible with regard to the system constraints.

Finally, the third contribution of this work concerns data replication. In fact, providing reliable services with high data availability and adequate performance are key requirements that must be met. The concept of replication is used to ensure such requirements. To solve the replica placement and selection problem, we proposed *StorNIR* (*a cost-efficient data object* ***Stor****ing scheme based on* ***N****SGAII upgraded with* ***I****njection and* ***R****eparation operators*). The same approach as for the second contribution was used in this work. Indeed, we used the matheuristic combining an exact method with NSGAII to solve the replica placement and selection problem. It should be noted that we have updated the evolution operators of NSGAII to adapt them to the replication problem. Additionally, when objects are replicated, the network latency of access requests can be optimized. Therefore, in the matheuristic approach we have proposed an algorithm to optimize the requests latency. It consists of assigning requests to the nearest location that satisfies the I/O storage performance.

TABLE DES MATIÈRES

TABLE DES FIGURES

Liste des tableaux

Liste des algorithmes

Acronymes

CAGR Compound Annual Growth Rate.

Capex capital expenditure.

CSP Cloud Service Provider.

DB DataBase.

E/S Entrée/Sortie.

EBS Elastic Block Store.

EC2 Amazon Elastic Compute Cloud.

FLA Federation Level Agreements.

FTL Flash Translation Layer.

GFS Google File System.

HDD Hard Disk Drives.

HDFS Hadoop Distributed File System.

HV Hypervolume.

I/O Input/Output.

IaaS Infrastructure as a Service.

IOPS Input/Output Operations Per Second.

MAPE-K Monitor, Analyze, Plan, Execute, Knowledge.

MILP Mixed Integer Linear Programming.

MOEA Multi-Objectif Evolutionary Algorithm.

MOO Multi-Objective Optimization.

MOOP Multi-Objectif Optimization Problem.

MOPSO Multi-Objective Particle Swarm Optimization.

NSGA Non Dominated Sorting Algorithm version II.

NSGAII Non Dominated Sorting Genetic Algorithm version II.

NVM Non Volatile Memory.

Opex Operational expenditure.

OS Operating System.

PaaS Platform as a Service.

PM Polynomial Mutation.

PM Physical Machine.

PSO Particle Swarm Optimization.

QoS Quality of Service.

RR Random Read.

RW Random Write.

SaaS Software as a Service.

SBX Simulated Binary Crossover.

SLA Service Level Agreement.

SOO Single-Objective Optimization.

SR Sequential Read.

SSD Solid State Drives.

STaaS STorage as a Service.

SW Sequential Write.

VM Virtual Machine.

VMM Virtual Machine Monitor.

YCSB Yahoo! Cloud Serving Benchmark.

Introduction

Chapitre 1

Introduction

Sommaire

1.1 Contexte scientifique

L'informatique en nuages, ou le *Cloud computing* (le terme *Cloud* est utilisé tout au long de ce manuscrit), [9, 10, 11, 12] est reconnu comme la norme *de facto* pour l'hébergement et la fourniture de services sur Internet. Ce concept informatique est basé sur le calcul parallèle, le calcul distribué, et le grid computing [13] (voir la section 2.1.2.3). L'élasticité, la multi-location, le libre-service, la répartition et le modèle de tarification selon l'usage (pay-as-you-go) sont ses principales caractéristiques [10, 11]. La qualité de service (QoS : Quality of Service) est garantie par un contrat, établi entre le client et le fournisseur de service (CSP pour *Cloud Service Provider*), appelé *Service Level Agreement* (SLA). Ce contrat comprend la description des services accordés par le CSP, les différents métriques des services, les garanties, et les pénalités à appliquer en cas de violation.

Bien qu'il y'a une vingtaine d'années, le *Cloud computing* était peu répandue, il alimente désormais tous les secteurs [14]. En effet, les particuliers utilisent des applications pour gérer leurs activités quotidiennes. Aussi, les petites entreprises, qui n'ont pas les moyens de créer leur propre centre de données, utilisent le *Cloud* en choisissant le modèle de service qui répond le mieux à leurs besoins. L'un des principaux facteurs de cette démocratisation est le grand avantage qu'offre les services de *Cloud* aux fournisseurs de services et aux clients. En effet, ces services permettent un déploiement rapide des applications pour les clients avec une facturation à l'usage. Pour les CSP, les services de *Cloud* représentent leur principale source de profit. Selon

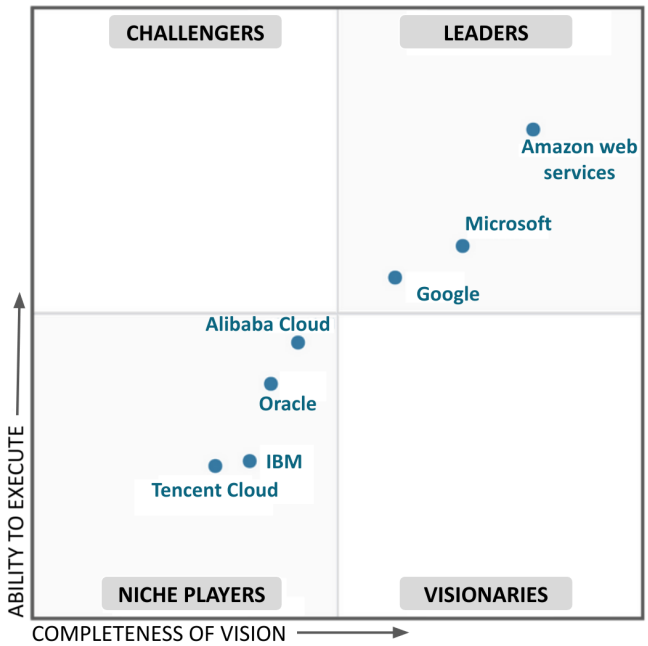


FIGURE 1.1 – Magic Quadrant de Gartner pour les services IaaS et PaaS [1]

[15], la taille du marché mondial du *Cloud Computing* devrait passer de 371,4 milliards USD en 2020 et atteindre jusqu'à 832,1 milliards USD d'ici 2025, avec un taux de croissance annuel composé (CAGR pour *Compound Annual Growth Rate*) de 17,5%.

Les modèles de services de *Cloud* peuvent être classés en trois catégories [16] : *(i) Infrastructure as a Service (IaaS) :* offre des ressources de calcul, de stockage et de réseau, *(ii) Platform as a Service (PaaS) :* fournit aux utilisateurs des outils qui facilitent le déploiement d'applications dans le *Cloud*, et *(iii) Software as a Service (SaaS) :* offre aux utilisateurs des applications prêtes à être utilisées sans configuration ni modification.

Plusieurs entreprises offrent des services *Cloud*. La figure 1.1 montre le *Magic Quadrant* de Gartner pour les fournisseurs de services IaaS et PaaS [1]. En effet, le *Magic Quadrant* [1, 17] est un outil de visualisation permettant de surveiller et d'évaluer les positions des entreprises sur un marché technologique spécifique. Les critères de Gartner lors de l'évaluation des entreprises sont la vision et la capacité d'exécuter la technologie en question. Selon ce schéma, Amazon, Microsoft et Google sont considérés comme des leaders ayant une excellente vision de la technologie et une bonne aptitude à la mettre en œuvre. Ces grandes entreprises ont une large clientèle et sont très visibles sur le marché. De plus, elles ont une grande influence sur le marché IaaS et PaaS et ont même la capacité d'influencer la direction globale du marché. D'autre part, Alibaba Cloud, Oracle, IBM et Tencent Cloud sont des marchés de niches ayant une vision du marché et une capacité à l'exécuter en retrait par rapport à leurs concurrents. Ils peuvent bien faire fonctionner la technologie considérée dans un segment particulier du marché mais ne peuvent pas surpasser

les plus grands fournisseurs.

Avec la démocratisation du *Cloud* et les exigences croissantes des clients, diverses limitations compromettant son fonctionnement ont émergées [18, 19, 20, 21, 22, 23], telles que la contention et le gaspillage de ressources, l'interruption de services et la dégradation de la qualité de service. Ces limitations peuvent affecter le potentiel de cette technologie, mettant en péril leur utilisation [20].

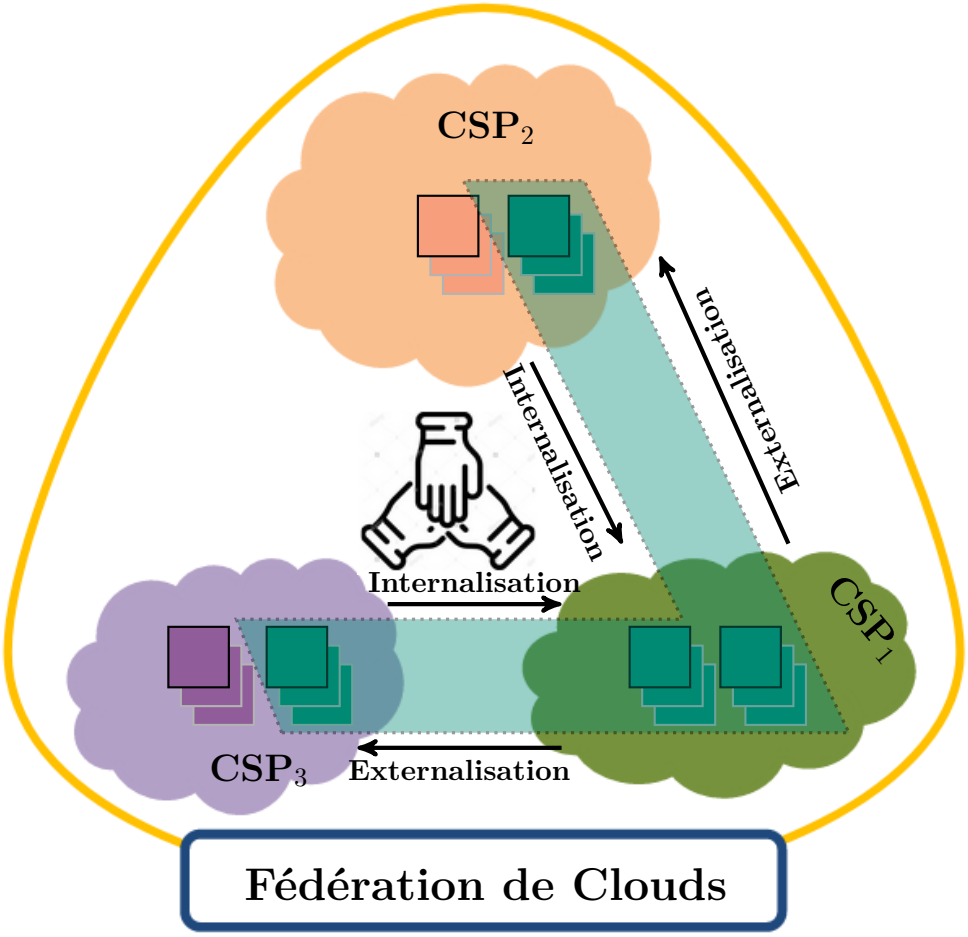


FIGURE 1.2 – Fédération de *Clouds*

Afin de surmonter ces limitations et satisfaire les utilisateurs, l'idée de l'hybridation des services en inter-connectant des *Clouds* hétérogènes et ainsi former un système fédéré a attiré l'attention des communautés scientifiques et industrielles. En effet, la fédération de *Clouds* [20, 18, 19] permet à différents CSP de travailler en collaboration à travers le partage de leurs ressources respectives afin d'ajuster dynamiquement leurs capacités d'hébergement en fonction de leurs charges de travail. Cette collaboration est régie par des accords au niveau de la Fédération (*Federation Level Agreements* : FLA). Ces accords définissent les règles et les conditions qui ré-

gissent la mise en commun et l'échange des ressources. Ce partage mutuel des ressources permet aux fournisseurs de surmonter leurs limitations et de fournir des services avancés en termes de performances, disponibilité et garanties de qualité de service. La figure 1.2 illustre les aspects de coopération au sein d'une fédération, à savoir l'*internalisation (insourcing)* et l'*externalisation (outsourcing)*. En fait, les CSP d'une fédération peuvent internaliser et externaliser les données et les charges de travail de leurs clients afin de fournir un service continu.

Le service de stockage de données dans le *Cloud*, communément connu sous le nom *STorage as a Service* : (STaaS), est considéré comme l'un des principaux composants du modèle IaaS [21, 24]. Il permet aux utilisateurs de stocker leurs données à distance avec le fournisseur de service de stockage plutôt que localement sur leur propres disques durs. Il en résulte que les données peuvent ensuite être consultées sur Internet de n'importe où en utilisant différents appareils tels que des ordinateurs de bureau, des ordinateurs portables, des tablettes et des smart-phones, ce qui offre une flexibilité et une accessibilité importante [25].

Le service STaaS gagne en popularité et de nombreuses entreprises transfèrent les données de leurs centres de données internes vers les fournisseurs de service de stockage. Depuis leur création, les services STaaS sont devenus une industrie lucrative [26]. En effet, selon un rapport publié par *Allied Market Research* [27], la taille du marché du stockage dans le *Cloud* a été évaluée à 46,12 milliards USD en 2019 et devrait atteindre 222,25 milliards USD d'ici 2027, avec un CAGR de 21,9% de 2020 à 2027.

Pour le service de stockage de données dans le *Cloud*, les performances des Entrée/Sortie (E/S) des supports de stockage et la latence du réseau sont parmi les métriques principales considérées par les clients [28, 29, 21]. De ce point de vue, placer efficacement les objets des clients pour un *Cloud* membre d'une fédération constitue un véritable défi. En effet, les E/S prennent environ 90% du temps d'exécution d'une transaction de base de données [22]. D'autre part, pour la latence réseau, Amazon a constaté, il y a une dizaine d'années, que pour chaque 100 ms supplémentaires de latence, les ventes diminuaient de 1% [30, 31]. Selon [32], une perte de 1% de revenus annuels pour Amazon en 2006 aurait été d'environ 107 millions de dollars. Aujourd'hui, ce serait environ 3,8 milliards de dollars. Ce qui signifie que garantir une bonne latence n'est plus une option mais un besoin.

Pour gérer les goulets d'étranglement des E/S, les infrastructures *Cloud* actuelles hébergent différentes classes de stockage ayant des modèles de performances complémentaires allant des disques durs (*Hard Disk Drives* ou HDD) aux disques électroniques représentés par les mémoires flash (*Solid State Drives* ou SSD), voire des mémoires non volatiles (*Non Volatile Memory* : NVM), comme les disques de la gamme Optane d'Intel [33, 34]. Pour gérer la latence réseau, les CSP essaient de rapprocher dynamiquement les données de leurs utilisateurs. Cela permet de réduire la latence ainsi que le trafic global entre les centres de données [35].

Cette thèse traite du problème de placement de données pour un CSP faisant

partie d'une fédération de *Clouds*. En effet, offrir des services attractifs et peu-coûteux est un grand défi pour les CSP. Bien que la fédération de *Clouds* permet de réduire les dépenses en capital (*capital expenditure* : Capex), les dépenses en exploitation (*Operational expenditure* : Opex) sont réduites par des stratégies d'optimisation. Notre but est de proposer des approches intelligentes pour un meilleur placement des données qui minimisent le coût de placement pour le fournisseur tout en satisfaisant les contraintes de QoS des clients.

Le reste de ce chapitre résume les principaux aspects de nos travaux de recherche et s'organise comme suit. La section 1.2 présente l'architecture globale du système considéré. Dans la section 1.3, la problématique est introduite. Les principales contributions scientifiques de cette thèse sont décrites dans la section 1.4. La section 1.5 décrit l'organisation de ce manuscrit et enfin, les publications liés aux travaux réalisés dans le cadre de cette thèse sont présentées dans la section 1.6.

1.2 Architecture générale du système considéré

Comme le montre la figure 1.3, nous considérons une fédération de *Clouds* composée de plusieurs fournisseurs, répartis sur différentes zones géographiques. Il peut y avoir un ou plusieurs CSP dans chaque zone avec des caractéristiques différentes.

On s'intéresse au placement des objets de données du point de vue d'un seul CSP (c'est le CSP_0 dans la figure). En effet, dans une fédération de *Clouds*, chaque CSP est libre d'appliquer ses propres techniques d'optimisation.

Le CSP_0 héberge une infrastructure de stockage hybride composée de plusieurs classes de stockage pour les objets de ses clients. Chaque classe est caractérisée par : sa capacité, ses performances en terme d'IOPS (*Input/Output Operations Per Second*), son prix d'achat et sa durée de vie. De plus, le CSP_0 peut exploiter les services de stockage proposés par des CSP partenaires en externalisant les objets de ses clients auprès de ces derniers, par exemple, lorsqu'il ne peut pas répondre aux exigences de QoS de ses clients ou lorsque le placement d'un objet dans un emplacement coûte moins cher que dans un autre. Les services de stockage externes proposés par les *Clouds* partenaires possèdent différentes caractéristiques en termes de capacité de stockages, performances d'E/S et coûts (occupation, IOPS et transfert réseau).

Le CSP_0 gère un ensemble de clients. Les clients peuvent être situés dans différentes zones géographiques. Ils peuvent être mobiles ou avoir des utilisateurs finaux dispersés géographiquement. Chaque client détient une application de données composée d'un ensemble d'objets. Il génère une charge de travail représentant les opérations d'E/S générées par l'ensemble des requêtes émises par le client. Le client est aussi caractérisé par un SLA décrivant les QoS requises et les pénalités associées en cas de non respect des exigences demandées.

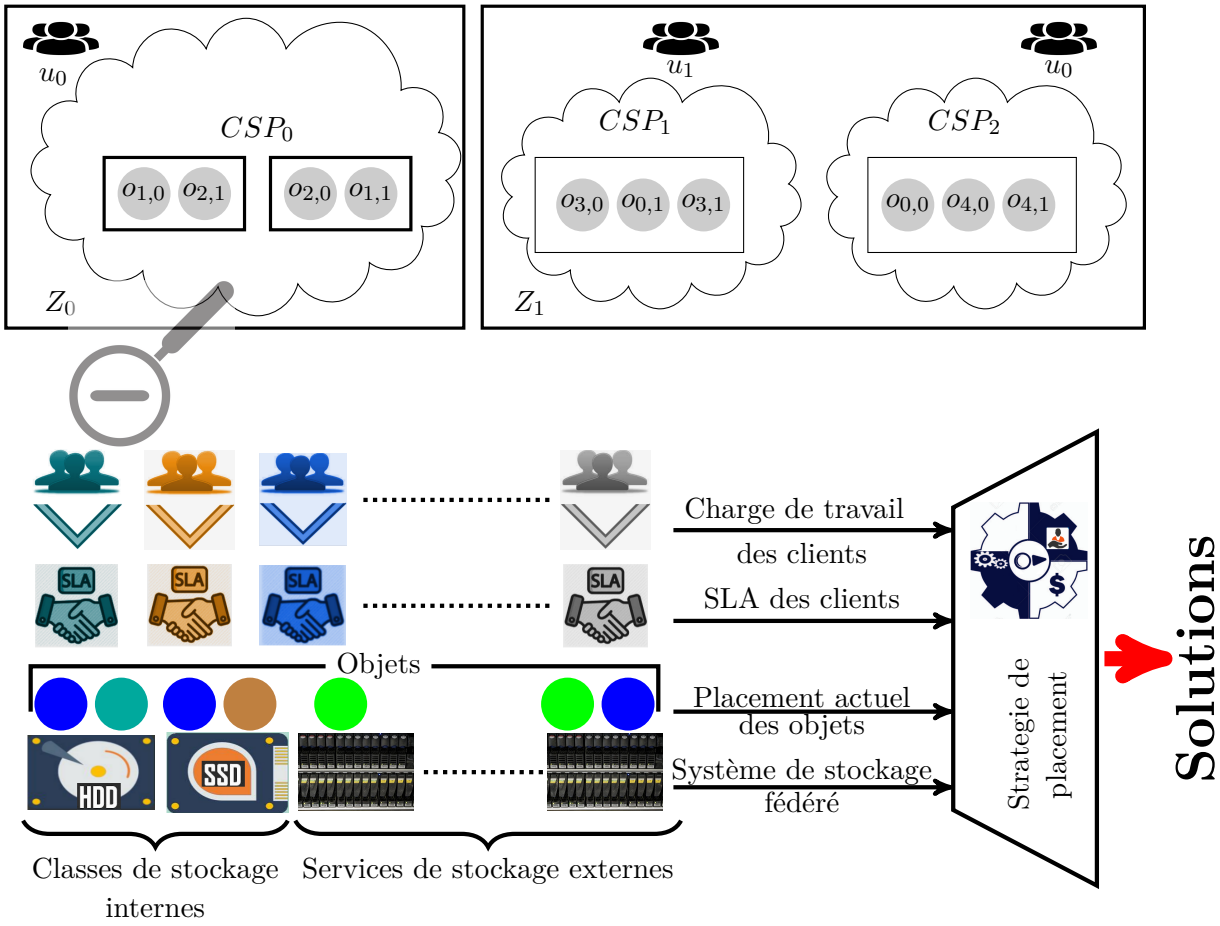


FIGURE 1.3 – Instance de modèle de système

Nous cherchons à trouver des plans de placement des objets des clients qui minimisent les coûts opérationnels du CSP_0 tout en satisfaisants les SLA des clients.

1.3 Prolématique

Dans cette thèse, nous nous intéressons au problème de placement de données des clients d'un CSP membre d'une fédération de *Clouds*. L'objectif est la satisfaction des clients en répondant à leurs exigences tout en minimisant le coût de placement pour le CSP afin d'augmenter son profit. Pour ce faire, nous étudions trois problématiques que nous présentons ci-dessous.

Problème 1 : Les méthodes d'optimisation, y compris les méthodes de placement de données, dans un environnement de *Cloud* sont généralement basées sur le coût pour les différents acteurs (clients et fournisseurs). Par conséquent, une mauvaise estimation du coût de placement des données dans un tel environnement peut entraîner une mauvaise configuration de placement

des objets des clients (violation des SLA des clients et/ou augmentation du coût de placement pour les CSP). Ainsi, la première problématique abordée dans cette thèse est : **comment évaluer le coût de placement de données dans un *Cloud* fédéré**, c'est à dire, quels sont les différents sous-coûts constituant le coût global de placement de données pour un fournisseur faisant partie d'une fédération de *Clouds* ?

Problème 2 : La deuxième problématique traitée dans ce travail est celle du placement des objets dans un système de stockage hybride (ressources internes hétérogènes : HDD et SSD) et fédéré (ressources externes : services de stockage des autres CSP). En effet, dans le contexte d'une fédération de *Clouds*, un CSP membre tente d'exploiter à la fois les services de stockage locaux et les services de stockage des autres CSP pour placer les données de ses clients. Le coût de placement global est composé des coûts de stockage, de migration et de latence. Ces coûts sont interdépendants et, dans certains cas, contradictoires. Par exemple, le placement des données localement peut être peu coûteux en termes de stockage et de migration, mais peut générer une latence importante du réseau pour les clients situés dans d'autres régions. La deuxième problématique adressée dans cette thèse est : **comment placer les objets des clients tout en minimisant au mieux le coût de placement et en satisfaisant les clients en termes de QoS exigées.** C'est à dire : quels objets doivent être sélectionnés pour la migration et où doivent-ils être placés ?

Problème 3 : Dans les *Clouds*, les données sont généralement répliquées pour plusieurs raisons comme l'amélioration de l'accessibilité et la robustesse du système. Pour un CSP, membre d'une fédération, un placement efficace des répliques des clients est important pour satisfaire les demandes de QoS. La conception d'une solution pour la réplication des données nécessite de répondre à trois questions principales : 1) Combien de répliques ? 2) Comment assurer la cohérence des données ? et 3) Où stocker les répliques ? Bien que la première et deuxième questions soient principalement liées aux exigences des applications, on s'intéresse particulièrement à la troisième question qui doit être résolue en ligne pour permettre un placement dynamique des données dans une infrastructure de fédération de *Clouds*. Ainsi, la troisième problématique adressée dans cette thèse est : **Comment placer les répliques des objets pour une meilleure accessibilité à moindre coût tout en respectant les contraintes en termes de SLA des clients ?**

1.4 Contributions

Cette partie présente nos contributions aux problèmes soulevés dans la section 1.3. Ces contributions s'articulent autour d'une méthodologie de modélisation pour un placement de données optimisé dans un *Cloud* fédéré. Cette méthodologie suit le modèle de gestion autonomique MAPE-K (*Monitor, Analyze, Plan, Execute, Knowledge*). Ce modèle, suggéré par IBM [5], se base sur la mise en œuvre d'une boucle de contrôle de rétroaction afin de définir une

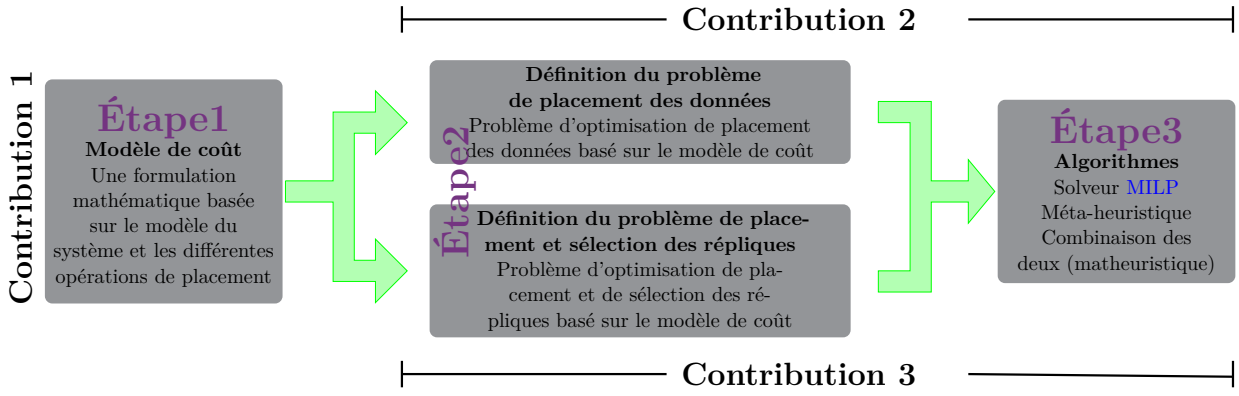


FIGURE 1.4 – Méthodologie de travail

gestion autonomique d'un système informatique (voir la section 2.1.2.4). Il comporte une séquence de quatre étapes principales avec une base de connaissance. Dans l'étape *Monitor*, des données sur le système considéré et de son environnement sont collectées à travers des capteurs préalablement installés. Une analyse de ces données pour vérifier si une adaptation du système est nécessaire est effectuée dans l'étape *Analyse*. Si c'est le cas, un nouveau plan est élaboré dans l'étape *Plan* composé d'un ensembles d'actions d'adaptation nécessaires pour atteindre les objectifs visés. Ces actions sont exécutées durant l'étape *Execute* à travers des actionneurs. La base de connaissances (*Knowledge*) conserve toutes les connaissances sur le système géré et l'historique des étapes précédentes.

En effet, notre travail est constituée de trois étapes principales intervenant dans deux phases du modèle MAPE-K. Tout d'abord (i) *un modèle de coût* est défini selon le modèle du système ainsi que les différentes opérations intervenant dans le processus de placement des données dans un *Cloud* fédéré. La deuxième étape concerne (ii) *la formalisation sous forme de problème d'optimisation des problèmes 2 et 3* (voir la section 1.3). La fonction objectif est basée sur le modèle de coût tout en prenant en compte les contraintes requises. Enfin, (iii) *les approches* proposées pour résoudre ces problèmes d'optimisation sont : l'utilisation d'un solveur MILP standard, une méta-heuristique évolutionnaire multi-objectifs, et une matheuristique (une combinaison des deux premières approches). L'étape (i) et l'étape (ii) font partie de la phase *(*Analyse) alors que l'étape (iii) intervient dans la phase *Plan*.

Le reste de cette section décrit brièvement les contributions réalisées dans le cadre de cette thèse.

1.4.1 Estimation du coût de placement des données

Afin d'offrir des services attractifs à des prix compétitifs, le CSP essaye de minimiser le coût de ses services tout en satisfaisant ses clients. Pour cette raison, un *Cloud* doit, tout d'abord,

estimer les sous-coûts constituant le coût global de placement de données de ses clients afin de l'exploiter dans les méthodes d'optimisation. Dans cette optique, nous avons défini mathématiquement un modèle de coût qui inclut tous les coûts induits par le placement des données dans un *Cloud* fédéré à base d'un système de stockage hybride. Notre modèle prend en compte différents facteurs modélisant les caractéristiques physiques et fonctionnelles du système fédéré.

Les résultats d'évaluation du modèle de coût proposé montrent la pertinence des coûts modélisés. Aussi, les évaluations indiquent que l'internalisation et l'externalisation sont des opérations complexes qui nécessitent de prendre en compte un grand nombre de paramètres. De plus, le modèle de coût peut être exploité dans plusieurs scénarios. En effet, il peut être appliqué par un *Cloud* hybride, un *Cloud* distribué composé de plusieurs centres de données appartenant à un seul fournisseur ainsi que par un *Cloud* ne possédant pas une infrastructure physique. Il permet d'évaluer plusieurs modèles de facturation de ressources fédérés.

Ce modèle de coût constitue une première phase de l'étape *Analyse* du processus MAPE-K.

1.4.2 Optimisation multi-objectifs de placement de données dans un *Cloud* fédéré

Après l'évaluation du coût de placement, nous nous sommes penchés sur la phase d'optimisation. En analysant les sous-coûts du placement des objets des clients internes, nous avons constaté qu'il est constitué de trois parties liées à trois opérations qui sont **1) le coût de stockage :** ce coût est en relation avec le placement futur des objets, **2) le coût de latence :** est lié au placement futur des objets et la localisation future des clients, et enfin **3) le coût de migration :** dépendant du placement actuel et futur des objets. Ces coûts sont interdépendants et dans certain cas contradictoires. Ainsi, des compromis sont inévitables ce qui nous a orienté vers une optimisation multi-objectifs.

Par conséquent, le coût de stockage, le coût de latence et le coût de migration, qui sont basés sur le modèle de coût proposé, constituent les fonctions objectifs. Le modèle est soumis à différentes contraintes liées à l'hétérogénéité des classes et services de stockage locaux et fédérés, les charges de travail des clients et leur SLA. Il est à noter que la définition du problème intervient dans l'étape *Analyse* du processus MAPE-K comme deuxième phase après celle de l'évaluation des coûts.

Dans l'optimisation multi-objectifs (*Multi-Objective Optimization* : MOO), il est rare qu'il existe une solution unique qui optimise simultanément toutes les fonctions objectifs [36]. En effet, il existe un ensemble de solutions, appelées solutions non dominées ou ensemble de Pareto. Ces solutions définissent des compromis entre les différents objectifs. La sélection d'une configuration finale de placement à partir de l'ensemble de Pareto n'est pas étudiée dans le cadre de cette contribution. Il s'agit d'une tâche subjective qui peut être basée sur d'autres facteurs et paramètres non pris en compte dans le processus d'optimisation [37].

Après l'étape d'*Analyse* vient l'étape *Plan* qui visent à construire des stratégies d'optimisation. Pour cela, nous avons proposé une approche basée sur la méta-heuristique évolutionnaire populaire *Non Dominated Sorting Genetic Algorithm version II* (NSGAII). L'approche développée CDP-NSGAII$_{IR}$ *(a Constraint Data Placement matheuristic based on NSGAII with Injection and Repair functions)* est une matheuristique de placement de données qui ajoute à NSGAII des fonctions d'injection et de réparation. La fonction d'injection vise à améliorer la qualité des solutions de NSGAII. Elle consiste à calculer des solutions à l'aide d'une méthode exacte puis à les injecter dans la population initiale de NSGAII. La fonction de réparation garantit que les solutions obéissent aux contraintes du problème et évite ainsi d'explorer de grands ensembles de solutions irréalisables.

1.4.3 Optimisation multi-objectifs de placement et sélection de répliques

La troisième contribution de ce travail de thèse concerne la réplication de données. Elle rentre aussi dans l'étape *Plan* du processus MAPE-K. En effet, fournir des services fiables avec une haute disponibilité des données et des performances adéquates sont des exigences clés qui doivent être satisfaites. Le concept de réplication est utilisé pour garantir de telles exigences.

Dans ce travail, nous avons proposé *StorNIR* (*a cost-efficient data object* ***Stor****ing scheme based on* ***N****SGAII upgraded with* ***I****njection and* ***R****eparation operators)*, une solution de placement de répliques d'objets de données pour un *Cloud* fédéré. D'abord, nous avons modélisé le problème de placement de répliques sous forme de problème d'optimisation multi-objectifs (*Multi-Objectif Optimization Problem* : MOOP) en tenant compte des différentes caractéristiques du système de stockage local, des services de stockage externe et des SLA des clients et leur charge de travail. En effet, le modèle précèdent (défini dans la contribution 2) a été mis à jour pour inclure la notion de réplication. Le calcul de quelques coûts a été modifié et des contraintes liées à la réplication (comme le nombre de répliques) ont été ajoutées.

Le même raisonnement pour la résolution du modèle proposé a été repris dans ce travail. En effet, nous avons utilisé la matheuristique combinant une méthode exacte avec la méta-heuristique NSGAII pour résoudre le problème de placement et de sélection des répliques. Il est à noter que nous avons mis à jour les opérateurs d'évolution de NSGAII pour les adapter au problème de réplication. De plus, lorsque les objets sont répliqués, la latence réseau des requêtes d'accès peut être optimisée. Par conséquent, dans l'approche matheuristique nous avons proposé un algorithme pour optimiser la latence des requêtes. Il consiste à affecter les requêtes à l'emplacement le plus proche qui satisfait les performances des E/S.

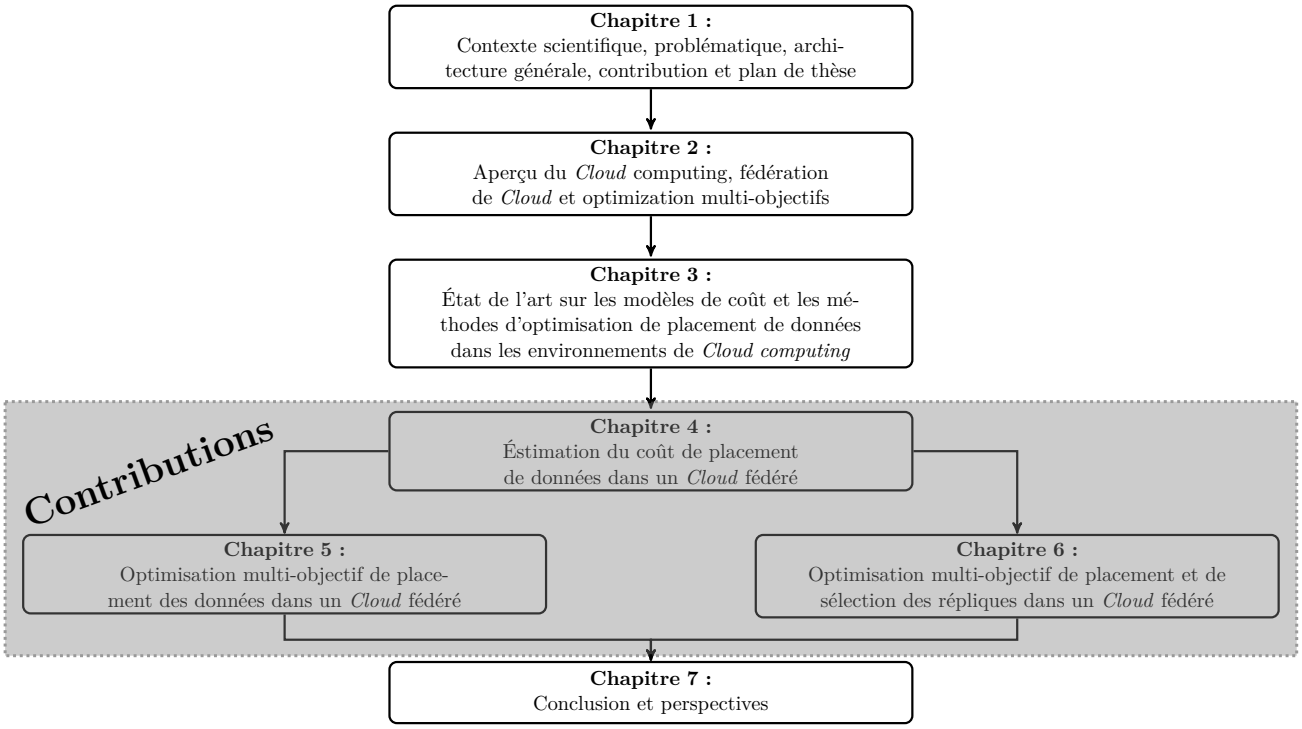


FIGURE 1.5 – Plan de mémoire

1.5 Plan du mémoire

Ce manuscrit de thèse est composé de sept chapitres comme indiqué dans la figure 1.5. Hormis l'introduction présentée dans ce chapitre, les autres six chapitres sont les suivants :

Le chapitre 2 définit le contexte et les concepts généraux sur le *Cloud computing*, la fédération de *Clouds* et l'optimisation multi-objectifs.

Le chapitre 3 présente un état de l'art des travaux d'évaluation du coût et d'optimisation de placement des données dans les *Clouds* centralisés et distribués.

Le chapitre 4 propose un modèle de coût pour le placement de données dans un *Cloud* fédéré. Il introduit les différents facteurs qui ont une influence sur le coût de placement et décrit les sous-coûts constituant le modèle de coût global.

Le chapitre 5 présente notre approche d'optimisation de placement de données dans un *Cloud* fédéré. Le problème de placement de données est d'abord formulé, ensuite l'approche d'optimisation matheuristique est détaillée en montrant les différentes étapes de sa conception.

Le chapitre 6 concerne l'optimisation du placement et de la sélection des répliques. Il commence par la formulation du problème. Pour la résolution, la même approche matheuristique utilisée dans le problème précédent est appliquée avec des mises à jour des algorithmes pour les adapter au problème de réplication.

Le chapitre 7 conclut la thèse par un résumé des principales contributions et résultats et

donne un aperçu des perspectives de travaux futurs .

1.6 Publications

Les publications associées aux travaux réalisés dans le cadre de cette thèse sont listées ci-dessous :

Contribution 1 : Estimation du coût de placement de données dans un *Cloud* fédéré
Amina Chikhaoui, Kamel Boukhalfa, et Jalil Boukhobza. *A cost model for hybrid storage systems in a Cloud federations.* In 2018 Federated Conference on Computer Science and Information Systems (FedCSIS), pages 1025–1034. IEEE, 2018.

Contribution 2 : Optimisation multi-objectif de placement des donnée dans un *Cloud* fédéré
Amina Chikhaoui, Laurent Lemarchand, Kamel Boukhalfa, et Jalil Boukhobza. *Multiobjective Optimization of Data Placement in a Storage-as-a-Service Federated Cloud.* In ACM Transactions on Storage (TOS), volume 17(3), pages 1-32, 2021.

Contribution 3 : Optimisation multi-objectif de placement et de sélection des répliques dans un *Cloud* fédéré
Amina Chikhaoui,Laurent Lemarchand, Kamel Boukhalfa, et Jalil Boukhobza. *StorNIR, a Multi-Objective Replica Placement Strategy for Cloud Federations.* In 36^{rd} Annual ACM Symposium on Applied Computing (SAC), pages 50-59, 2021.

Contexte et état de l'art

Chapitre 2

Contexte

Ce chapitre décrit le contexte scientifique des contributions présentées dans cette thèse de doctorat. Tout d'abord, nous donnons un aperçu des concepts de base du *Cloud computing*. Ensuite, nous étudions le concept de fédération de *Clouds*. Enfin, un aperçu de l'optimisation multi-objectifs conclut ce chapitre.

Sommaire

2.1 *Cloud computing*

Le *Cloud computing* [9, 12, 10] fournit des ressources informatiques (calcul, stockage, réseaux, plates-formes et applications) sous forme de services sur le réseau. Les utilisateurs du *Cloud*, qui n'ont généralement pas connaissance de l'endroit et de la manière dont les services sont hébergés, peuvent consommer ces services sur une base de paiement à l'utilisation. Le terme *Cloud computing* est apparu pour la première fois en 1996 [38].

Le concept de cette technologie correspond parfaitement à la prédiction de McCarthy faite il y a 60 ans qui a déclaré, lors de la célébration du centenaire du MIT en 1961, que *«Computing may someday be organized as a public utility just as the telephone system is a public utility. Each subscriber needs to pay only for the capacity he actually uses, but he has access to all programming languages characteristic of a very large system . . . Certain subscribers might offer service to other subscribers . . . The computer utility could become the basis of a new and important industry»*.

À travers cette section, nous examinons les principaux aspects du paradigme de *Cloud computing*.

2.1.1 Définition et caractéristiques

Bien que diverses définitions du *Cloud* ont été proposées par les communautés académique et industrielle [39, 40, 41], la définition la plus utilisée est celle fournie par le National Institute of Standards and Technology (NIST) aux États-Unis [11].

Le NIST définit le *Cloud Computing* comme étant un modèle permettant l'accès via un réseau à un ensemble de ressources informatiques mutualisées et configurables (réseaux, serveurs, stockage, applications et services) d'une manière simple et à la demande. Ces ressources informatiques peuvent être acquises et libérées rapidement avec le minimum d'effort de gestion ou d'interaction avec les fournisseurs de services.

Par ailleurs, le NIST précise que le *Cloud computing* est caractérisé par 5 propriétés essentielles qui le distinguent des autres services informatiques traditionnels. Ces caractéristiques sont détaillées comme suit [11] :

Ressources à la demande : Les utilisateurs sont en mesure de provisionner unilatéralement et automatiquement des ressources informatiques selon leurs besoins de façon simple et flexible sans nécessiter d'interaction humaine avec chaque fournisseur de services ;

Accès réseau : les ressources sont disponibles et accessibles à distance via des mécanismes standards qui favorisent leur utilisation à partir des plates-formes client hétérogènes telles que téléphones mobiles, tablettes, ordinateurs portables et stations de travail ;

Élasticité adaptable : les ressources sont rapidement et automatiquement allouées *(scale up)* et libérées *(scale down)* selon les besoins des utilisateurs sans perturbation des tâches en cours d'exécution. Cela donne aux consommateurs l'illusion de disposer de quantités infinies de res-

sources pouvant être allouées à tout moment.
Services mesurés : Toutes les ressources allouées sont surveillées et contrôlées afin de mesurer leur consommation avec un niveau d'abstraction approprié. Ces mesures sont rapportées de manière transparente, afin de permettre une facturation à l'usage.
Mutualisation des ressources : Les ressources informatiques du fournisseur de *Cloud* sont mutualisées pour servir plusieurs clients. Ces ressources sont dynamiquement allouées et libérées selon la demande des consommateurs qui n'ont aucune connaissance de l'emplacement physique exact des ressources fournies. Le partage des ressources est réalisé selon un modèle multi-locataires basé sur les technologies de virtualisation.

2.1.2 Technologies habilitantes du *Cloud computing*

Le *Cloud computing* fournit des services informatiques dynamiques fiables, personnalisés aux utilisateurs avec des garanties de QoS. Néanmoins, le *Cloud* n'est pas une idée complètement nouvelle, il a émergé comme une convergence de plusieurs technologies informatiques contemporaines comme le *Grid comuting*, la virtualisation, etc.

Avec la complexité des applications et la croissance des exigences, des technologies telles que la virtualisation et l'informatique autonomique sont apparues. La virtualisation désigne l'abstraction des ressources informatiques des applications et des utilisateurs. Quant à l'informatique autonomique, elle fait référence aux caractéristiques d'autogestion des ressources informatiques distribuées en s'adaptant aux changements imprévisibles. Ces technologies constituent des briques de base d'autres technologies comme le cluster, le *Grid computing* et actuellement le *Cloud computing*. Bien que le *Grid* et *Cloud* partage la même vision, celle de permettre un accès fiable à de grandes capacités de calcul et de stockage de manière virtualisée, ces deux technologies informatiques sont bien distinctes l'une de l'autre. Le *grid computing* a été principalement ciblé sur des initiatives scientifiques à grande échelle dans le contexte du calcul haute performance, il peut être assez complexe à utiliser. Le *Cloud computing* a émergé en offrant une puissance informatique à un public plus large qui n'a pas forcément une formation en informatique. La multi-location est un aspect important du *Cloud computing*. Elle fait référence à une architecture logicielle dans laquelle une seule instance de logiciel, s'exécutant sur un serveur, peut desservir plusieurs clients appelés locataires (*tenants* en anglais).

Cette section donne un aperçu de la base technologique du *Cloud computing*.

2.1.2.1 Virtualisation

Provenant du système d'exploitation IBM CP/CMS apparu dans les années 60, la virtualisation est l'une des pierres angulaires du *Cloud computing*. La philosophie de cette technologie consiste à déployer plusieurs serveurs virtuels sur une seule machine physique en abstrayant ses

ressources physiques [4]. Ainsi, la virtualisation permet d'isoler les applications et donc de faciliter leur portabilité, d'optimiser l'utilisation des ressources matérielles et d'offrir une flexibilité opérationnelle [42, 4].

Deux types de solutions de virtualisation existent selon le niveau d'abstraction [43, 44, 3, 45, 4] : (i) abstraction des ressources physique à travers l'utilisation d'un hyperviseur et (ii) abstraction du système d'exploitation à travers la conternalisation.

Systèmes basés sur un hyperviseur : ce type de virtualisation consiste à reproduire le comportement d'une machine physique (*Physical Machine* : PM) dans un environnement logiciel appelé machine virtuelle (*Virtual Machine* : VM) à l'aide d'un hyperviseur, appelé à l'origine un moniteur de machine virtuelle (*Virtual Machine Monitor* : VMM). Le matériel physique sur lequel l'hyperviseur s'exécute est généralement appelé hôte, tandis que les VM créées et prises en charge par l'hyperviseur sont appelées machines invitées. Un déploiement basé sur un hyperviseur est idéal lorsque les applications nécessitent différents systèmes d'exploitation ou différentes versions de système d'exploitation [46]. Les systèmes basés sur l'hyperviseur sont généralement classés en deux catégories : *hyperviseur bare metal* (Type-1) qui fonctionne directement sur le matériel hôte (par exemple XEN), ou *hyperviseur invité* (Type-2) qui fonctionne au-dessus du système d'exploitation de l'hôte (par exemple VMware workstation). La figure 2.1 et la figure 2.2 illustrent respectivement l'hyperviseur invité et l'hyperviseur bare metal.

Systèmes basés sur le conteneur : appelés aussi conteneurs légers (par opposition à la VM, parfois qualifiée de conteneur lourd) ce type de systemes de virtualisation n'utilise pas d'hyperviseur, mais virtualise les ressources au niveau du système d'exploitation comme c'est montré sur la figure 2.3. Cette approche vise à mutualiser le système d'exploitation et certaines librairies allégeant ainsi considérablement la granularité des briques déployées ce qui permet de gagner en rapidité de déploiement. Un conteneur léger peut être déployé aussi bien sur des PM que sur des VM. Docker [47, 48] est une solution *Open Source* de déploiement des applications dans des containers logiciels.

La virtualisation basée sur un conteneur entraîne des frais généraux inférieurs à ceux de la virtualisation basée sur un hyperviseur car un seul système d'exploitation est exécuté. Aussi, ce type de virtualisation peut atteindre une densité plus élevée d'instances virtualisées et les images de disque sont plus petites que les solutions basées sur un hyperviseur [3]. Cependant, le partage du noyau hôte présente également des inconvénients, tels qu'une isolation réduite entre les instances et l'incapacité à héberger différents systèmes d'exploitation invités [4, 3].

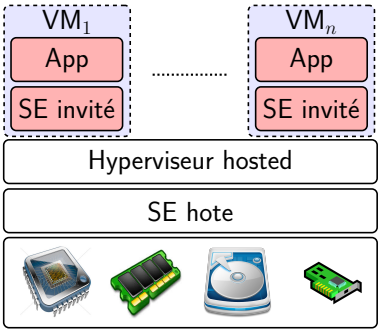


FIGURE 2.1 – Hyperviseur invité [2, 3]

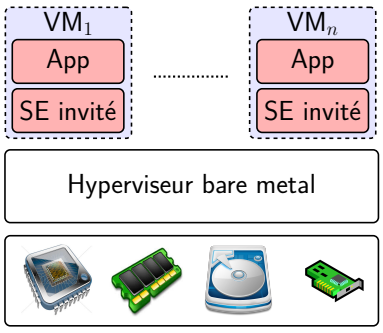


FIGURE 2.2 – Hyperviseur bare metal [2, 3]

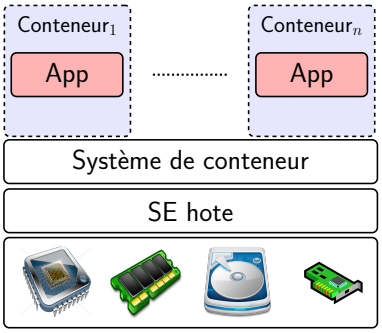


FIGURE 2.3 – Conteneur [4, 3]

2.1.2.2 Centres de données

Un centre de données est une infrastructure dans laquelle un certain nombre de serveurs, d'unités de stockage et d'équipements de réseau, ainsi que tout le matériel non informatique tels que les systèmes de refroidissement et les alimentations sans interruption, sont co-localisés en raison d'exigences environnementales communes, de besoins de sécurité physique ainsi que de facilité d'entretien [49].

Regrouper les ressources informatiques à proximité les unes des autres, plutôt que de les repartir géographiquement, permet un partage de puissance, une plus grande efficacité dans l'utilisation des ressources informatiques partagées et une meilleure accessibilité [50].

Les centres de données varient en taille, des salles de serveurs qui prennent en charge des petites et moyennes entreprises aux batteries de serveurs qui exécutent des services *Cloud* à très grande échelle [49].

2.1.2.3 *Grid computing*

Le *Grid computing* ou l'informatique en grille [51] est basée sur l'interconnexion de ressources de calcul et de stockage géographiquement dispersées pour former une infrastructure cohérente pour le calcul à grande échelle. L'exploitation des ressources du *Grid computing* est réalisée par un intergiciel *(middleware)* qui est une couche logicielle intermédiaire permettant le dialogue entre l'application de l'utilisateur et les ressources informatiques matérielles et logicielles sous-jacentes. Bien que le *Cloud Computing* soit reconnu comme un descendant du *Grid Computing*, il existe des différences significatives entre les deux [52]. Alors que le *Grid Computing* se concentre davantage sur des applications scientifiques nécessitant des capacités de calcul et/ou de stockage élevées, le *Cloud computing* est régi par des exigences spécifiques des petites et moyennes entreprises. D'ailleurs, selon les besoins et les objectifs visés par les applications, les grilles peuvent être classées en trois différentes catégories qui sont : grilles de calcul, grilles de données et grille de service [53]. De plus, le *Grid Computing* tente toujours d'atteindre une

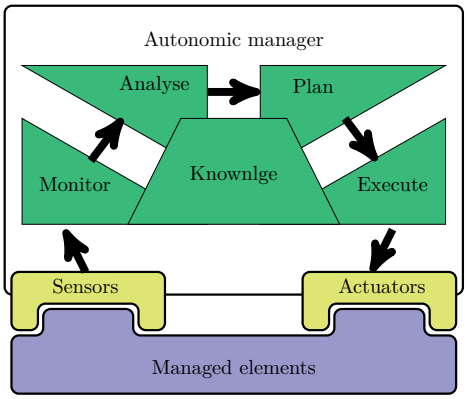


FIGURE 2.4 – Modèle de référence MAPE-K d'IBM pour les boucles de contrôle autonomes [5]

capacité de calcul maximale tandis que le *Cloud computing* se concentre sur les services à la demande où l'usage des ressources est à la hausse ou à la baisse selon les besoins des utilisateurs.

2.1.2.4 Informatique autonomique

Pour gérer efficacement l'infrastructure sous-jacente des systèmes informatiques modernes, qui sont souvent complexes et caractérisés par des charges de travail dynamiques, les entreprises doivent exercer un contrôle constant sur des centaines de composants et des milliers de paramètres de réglage [54]. L'effort humain nécessaire pour faire fonctionner et maintenir ces systèmes devient incontrôlable [55].

L'informatique autonomique constitue une solution pour éliminer la nécessité d'une intervention humaine. C'est une discipline issue de l'initiative d'informatique autonomique d'IBM [56, 5] visant à développer des systèmes informatiques capables de s'auto-gérer selon les objectifs fixés par les administrateurs. IBM a comparé des systèmes informatiques complexes au corps humain, qui est capable de réguler les fonctions corporelles inconscientes telles que la respiration, les fonctions digestives, les ajustements pupillaires au moyen du système nerveux autonome. De ce point de vue, les systèmes informatiques doivent également présenter certaines propriétés autonomiques et être capables de prendre en charge les tâches de maintenance et d'optimisation régulières, réduisant ainsi la charge de travail des administrateurs du système [55].

La vision d'IBM de l'informatique autonomique est structurée autour d'un modèle de référence pour les boucles de contrôle [55], connu sous le nom de boucle de rétroaction (*Monitor, Analyze, Plan, Execute, Knowledge* : MAPE-K) , illustrée dans la figure 2.4.

Un système autonomique est composé de deux modules principaux :

1. Les éléments à gérer (*Managed elements*) : peuvent être des composants matériels et/ou logiciels auxquels un comportement autonomique est applicable.
2. Le gestionnaire autonomique (*Autonomic manager*) : c'est un module logiciel régissant

le comportement des éléments à gérer associés. Il comporte quatre étapes et une base de connaissance qui sont expliquées brièvement ci-dessous.

Ces deux modules interagissent via des capteurs *(Sensors)* (également appelés sondes dans la littérature) et des actionneurs *(Actuators)*. En effet, le gestionnaire autonomique surveille régulièrement les éléments à gérer en utilisant des capteurs. Ces derniers collectent des métriques et les transmettent au gestionnaire autonomique et/ou lui permettent de les récupérer. Cela permet au gestionnaire autonomique d'être à jour par rapport à l'évolution des éléments gérés afin d'agir sur ces éléments en modifiant leur état ou leur comportement via des actionneurs pour atteindre les objectifs souhaités.

Les cinq étapes du modèle MAPE-K fonctionnent comme suit : tout d'abord, l'étape *Monitor* est chargée de remonter les informations brutes fournies par les capteurs sur l'état courant des éléments à gérer. Cette étape permet de sélectionner les données remontées et de les agréger afin d'en extraire des informations significatives. Dans, la deuxième étape *Analyze*, les résultats de l'étape précédente sont récupérés afin de les analyser et de déterminer l'état du système en utilisant des modèles prenant en compte les objectifs à atteindre. L'étape *Plan* permet de déterminer une nouvelle configuration des éléments à gerer en prenant en entrée les résultats de l'étape *Analyze*. Un plan d'actions est généré dans cette étape qui est appliqué sur les éléments à gérer à l'étape *Execute* à l'aide de l'ensemble des actionneurs.

La base de connaissance *Knowledge* est partagée entre les quatre précédentes étapes. Elle peut être mise à jour durant n'importe quelle étape et aussi par les administrateurs du système au fur et à mesure de l'exécution du modèle MAPE-K.

2.1.2.5 Multi-location

La multi-location (*multi-tenancy* en anglais) constitue un aspect important du *Cloud computing*. En effet, la multi-location fait référence au principe selon lequel une seule instance d'une application peut servir simultanément plusieurs locataires [57] tout en assurant une séparation de leurs données. Cette technologie permet une utilisation optimisée des ressources matérielles et facilite le déploiement et la maintenance de l'application car une seule instance d'application doit être déployée ce qui diminue les coûts globaux de l'application [58]. En outre, la multi-location améliore le passage à l'échelle de l'application.

2.1.3 Modèles de service du *Cloud computing*

Le terme service désigne une tâche automatisable, livrée aux clients de manière cohérente et reproductible pour résoudre des défis spécifiques [59]. Un modèle de service *Cloud* représente les services et les capacités qui seront proposés via le *Cloud*. Selon la définition du NIST [16], le *Cloud computing* offre trois modèles de service : (i) *Software as a Service* (SaaS), (ii) *Platform*

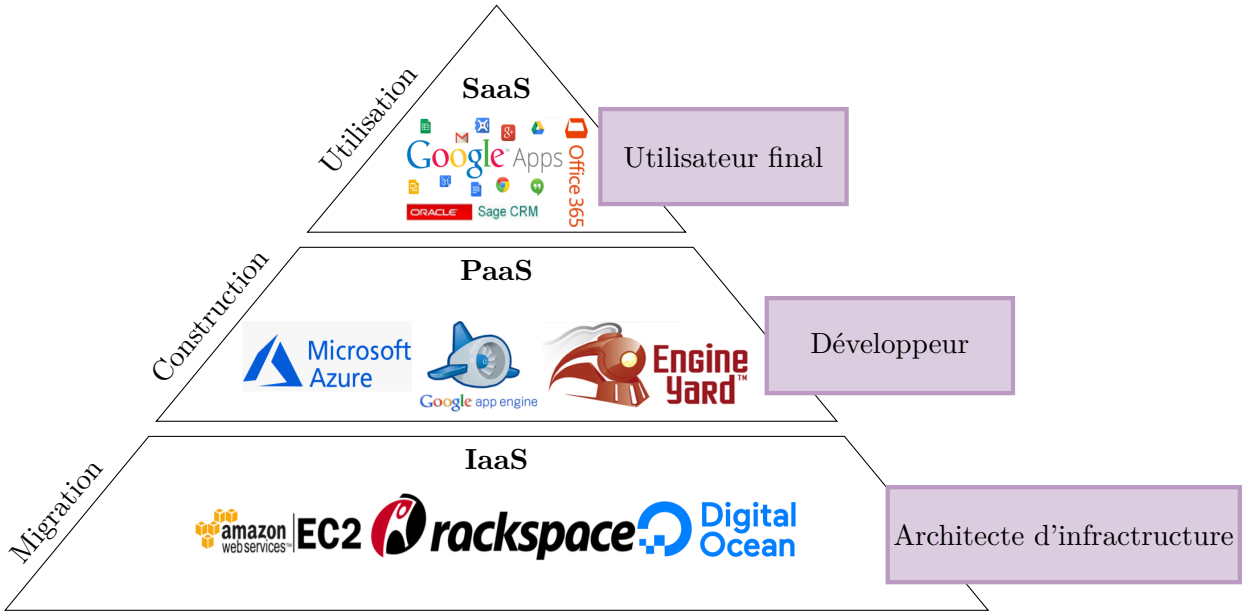


FIGURE 2.5 – Modèles de services du *Cloud computing*

as a Service (PaaS), et (iii) *Infrastructure as a Service* (IaaS). Ces modèles diffèrent dans les types de ressources mises à la disposition des utilisateurs. L'architecture du *Cloud computing* est souvent représentée comme une pile de ces trois couches abstraites, où chaque couche peut offrir ses ressources en tant que service aux couches supérieures comme c'est illustré dans la figure 2.5. Ces trois modèles sont présentés dans cette section.

2.1.3.1 SaaS :

Ce modèle désigne des applications prêtes à l'utilisation et offertes à distance par un fournisseur de services *Cloud*. Dans ce modèle, toutes les procédures d'installation, de mise à jour et de maintenance de l'application et de son environnement d'exécution sont effectuées par le fournisseur [60]. Parmi les applications SaaS les plus populaires, nous citons la suite logicielle de collaboration Google Apps, l'application de gestion de la relation client d'Oracle (Sage CRM), etc. Les solutions SaaS sont généralement soit disponibles pour une utilisation gratuite, soit facturées sur la base d'un abonnement.

2.1.3.2 PaaS :

Les services *Cloud* de type PaaS sont généralement conçus pour les développeurs de logiciels. Ils fournissent aux utilisateurs des plates-formes pour développer, tester et déployer leurs applications *Cloud* [61]. Les plates-formes PaaS sont des environnements intégrés de haut niveau (systèmes d'exploitation, langages de programmation, librairies, bases de données, etc.) prenant

en charge le cycle de vie complet du logiciel. PaaS permet le développement rapide d'applications à faible coût. Ce modèle de service est également utile pour développer des applications nécessitant des ressources informatiques puissantes telles que les applications d'analyse BigData [62]. Un exemple de solutions PaaS est Google App Engine qui fournit aux utilisateurs des outils de développement ainsi que des services sous-jacents tels que des bases de données et des mécanismes de cache.

2.1.3.3 IaaS :

IaaS fait référence à l'approvisionnement à la demande en ressources informatiques de base (calcul, mémoire, stockage et réseau). Les ressources de ce modèle sont utilisées comme des ressources virtualisées [60]. Dans ce modèle, les clients IaaS ont un meilleur contrôle sur leurs ressources par rapport aux modèles SaaS et PaaS. Ils sont responsables de la gestion du système d'exploitation, des applications et des données déployées, tandis que les fournisseurs IaaS gèrent toujours le matériel sous-jacent et les couches de virtualisation. De plus, les utilisateurs IaaS peuvent étendre dynamiquement leurs ressources louées en fonction de leurs besoins, ce qui leur permet de ne payer que ce qu'ils utilisent. *Amazon Elastic Compute Cloud* (EC2) est considéré comme l'acteur le plus important sur le marché IaaS. Ce service propose différentes instances de VM avec diverses configurations de calcul, mémoire, stockage et réseau. Les VM sont louées à des prix fixes et dynamiques pour s'adapter aux différents cas d'utilisation.

Ci-après nous donnons un aperçu du service de stockage dans le *Cloud* qui fait partie du modèle IaaS.

2.1.4 Aperçu sur le stockage dans le *Cloud*

L'une des principales utilisations du *Cloud computing* est le stockage de données. En effet, le stockage dans le *Cloud* (STaaS) est de plus en plus sophistiqué et flexible et est devenu très important dans les domaines industriel et académique [63]. Il permet aux organisations de réduire les dépenses en ressources internes. En effet, les organisations peuvent héberger leurs données dans le *Cloud* sans avoir besoin d'acquérir du matériel coûteux, d'installer et de maintenir un logiciel de base de données supplémentaire. De plus, le stockage dans le *Cloud* suit la philosophie de paiement selon l'usage. Par conséquent, le client n'a pas besoin d'anticiper la quantité d'espace de stockage dont il aura besoin dans le futur. Il peut ajuster ses ressources et payer en fonction de ses besoins actuels. Par rapport aux solutions de stockage conventionnelles, le stockage basé sur le *Cloud* peut augmenter l'intégrité, la disponibilité et la durabilité des données.

Pour un tel service, la performance des E/S de supports de stockage et la latence du réseau sont les deux métriques principales considérées par les clients. En effet, 90% du temps d'exécution de la transaction est passé dans le système d'E/S [22]. Concernant la latence de réseau, certaines

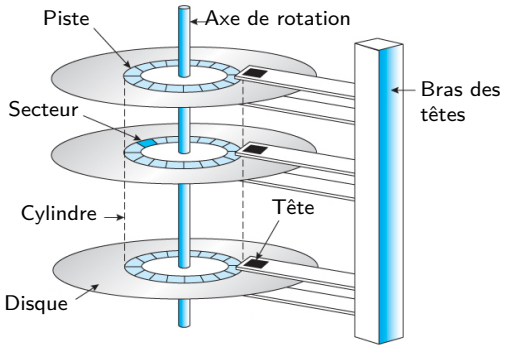


FIGURE 2.6 – Structure interne des disques durs [6]

sociétés de *Cloud computing* incluent déjà des garanties de latence dans leurs contrats de niveau de service (SLA).

Pour améliorer la latence réseau, les CSP essayent de placer les données près de leurs utilisateurs. Selon [64, 65, 66], Amazon a constaté que pour chaque 100 millisecondes, les ventes diminuaient de 1%, Google a constaté qu'un retard d'une demi-seconde entraîne une baisse de 20% du trafic ce qui nuit à la satisfaction des utilisateurs. De plus, un courtier *Cloud* (*broker* en anglais qui est une firme ou une personne intermédiaire qui organise des transactions entre le client et le fournisseur) pourrait perdre 4 millions de dollars de revenus par milliseconde si sa plateforme de commerce électronique a 5 millisecondes de retard par rapport à la concurrence [64, 65, 66].

Pour gérer les goulets d'étranglement des E/S, les infrastructures *Cloud* actuelles hébergent différents types de classes de stockage ayant des performances complémentaires allant des disques rotatifs (HDD) aux disques électroniques à base de mémoires flash (SSD) et des mémoires non volatiles (NVM) comme la mémoire Optane.

Dans le reste de cette section, nous allons présenter les deux principales technologies de stockage à savoir : les disques magnétiques (HDD) et les disques électroniques à base de mémoire Flash (SSD).

2.1.4.1 Disques durs magnétiques

Depuis plusieurs décennies, les disques durs magnétiques ont été les principaux périphériques de stockage non volatiles disponibles. Ils ont joué un rôle important dans le développement des bases de données grâce à leur propriété d'accès direct aux données. Les disques durs ont fait leur apparition la première fois en 1956 dans les laboratoires d'IBM (*IBM 350 disk*). La taille approximative de ce lecteur était de deux réfrigérateurs et stockait 5 millions de caractères de 6 bits (l'équivalent de 3,75 millions d'octets) sur une pile de 50 disques [67]. Grâce aux divers développements technologiques, le disque dur d'aujourd'hui est un système électro-magnéto-

mécanique très complexe avec plus de performances et des capacités de plusieurs Téra octet.

Physiquement, un disque dur est composé d'un ensemble de plateaux (en métal, en verre ou en céramique), concentriques et superposés. Ils tournent autour d'un axe (à plusieurs milliers de tours par minute actuellement).

Les surfaces des plateaux sont recouvertes d'oxyde magnétique ce qui permet de stocker et de récupérer les données sous forme de zéros et de uns en polarisant la zone concernée. Chaque plateau est divisé en plusieurs cercles concentriques appelés pistes qui sont, à leur tour, divisées en secteurs de taille généralement de 512 octets. L'ensemble des pistes qui se trouvent à la même position sur les plateaux constitue un cylindre.

Les données sont lues et écrites grâce aux têtes de lecture/écriture. Les tètes sont des électro-aimants qui modifient le champ magnétique local selon le courant électrique qui les traverse. De même, pour la lecture, le mouvement du champ magnétique local engendre aux bornes de la tête un potentiel électrique qui dépend de la valeur écrite. Dans les disques durs modernes, il y a une tête pour chaque surface de plateau, montée sur un bras commun qui déplace rapidement les têtes à travers les plateaux pendant qu'ils tournent. Ce bras permet à chaque tête d'accéder à presque toute la surface du plateau. La figure 2.6 illustre la structure interne des disques durs.

L'électronique associée contrôle le mouvement du bras des têtes ainsi que l'axe de rotation des plateaux afin de réaliser les opérations de lecture et d'écriture suivant les requêtes reçues.

Les HDD ont été le composant le plus lent de nombreux systèmes informatiques, en raison des longs temps d'accès aux données et aux limitations de leur bande passante. Ces supports de stockage sont à la fois énergivores et peu performants par rapport à leurs analogues de type SSD notamment lorsqu'il s'agit des opérations de lectures et écritures aléatoires. En effet, ces opérations aléatoires provoquent plus de déplacement du bras mécanique que les opérations séquentielles [68]. Cependant, les disques HDD ont un faible prix par gigaoctet et une longue durée de vie par rapport aux disques SSD ce qui maintient leur existence sur le marché du stockage.

Les disques SSD font l'objet de la section suivante.

2.1.4.2 Disques basés sur la mémoire Flash

La mémoire flash est une mémoire non volatile, inventée par Fujio Masuoka dans les années 1980 quand il travaillait à Toshiba. C'est une forme avancée de la mémoire EEPROM (Electricaly Erasable Programmable Read Only Memory) qui est une mémoire effaçable par courant électrique et capable de maintenir les données sauvegardées lors d'une mise hors tension.

Selon la porte logique utilisée dans les cellules mémoires, il existe deux types de mémoire flash : les mémoires flash NOR et les mémoires flash NAND. Les deux mémoires flash utilisent des transistors MOS à grille flottante comme c'est illustré dans la figure 2.7. La grille flottante existe entre la grille de contrôle et le substrat de silicium. Elle est complètement isolée par des

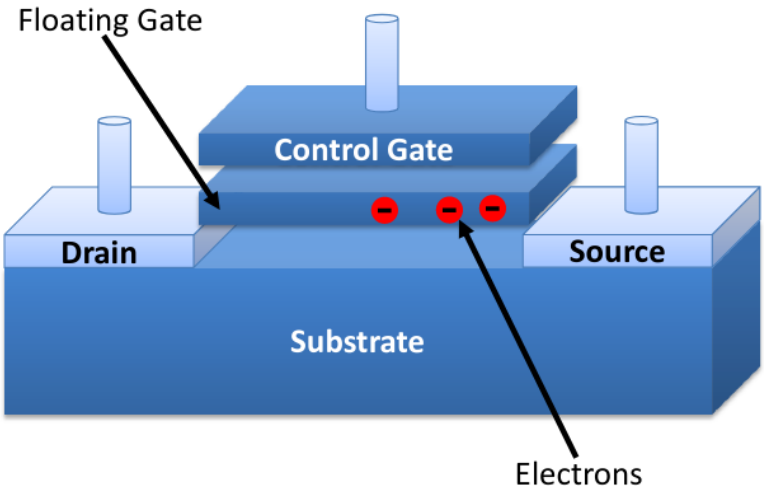


FIGURE 2.7 – Transistor à grille flottante [7]

matériaux diélectriques, donc, elle peut piéger les électrons et maintenir la charge afin de stocker l'information. Faire traverser l'oxyde aux électrons se fait par : (i) injection d'électrons chauds pour les mémoires de type NOR et (ii) injection Fowler-Nordheim pour les flash NAND.

Les cellules sont connectées en parallèle dans les mémoires NOR. Ce type de mémoire flash prend en charge les accès aléatoires par octet et est principalement utilisée pour stocker du code afin de remplacer la DRAM plus chère. Tandis que dans les mémoires flash NAND, les cellules sont connectées en série ce qui offre une meilleure densité de stockage et un accès rapide en écriture. Ce type de mémoire flash n'autorise l'accès qu'en unités de pages. Dans le reste de cette section, la mémoire flash fait référence à la mémoire flash NAND spécifiquement.

Les mémoires flash de type NAND peuvent être catégorisées en cinq classes selon le nombre de bits qui peuvent être stockées dans une cellule mémoire : 1) SLC (Single Level Cell), 2) MLC (Multi Level Cell), 3) TLC (Triple Level Cell), 4) QLC (Quad-Level Cell) et 5) PLC (Penta-Level Cell) [69]. Ces types peuvent respectivement stocker 1 (SLC), 2 (MLC), 3 (TLC), 4 (QLC) et 5 (PLC) bits par cellule. Ces types de cellules mémoire se distinguent par certaines caractéristiques. En effet, l'augmentation de nombre de bits par cellule, permet de baisser le prix de la mémoire flash mais, en revanche, elle diminue ses performance, sa fiabilité et sa durée de vie [69, 70].

Le disque SSD basé sur la mémoire Flash NAND possède une architecture hiérarchique. La figure 2.8 montre la façon dont la mémoire flash NAND est organisée au sein d'un SSD. La mémoire flash est répartie sur plusieurs puces (*chipset*). Chaque *chipset* contient un ou plusieurs dés eux même composés de plans. Les plans contiennent des blocs. Le bloc est considéré comme la plus petite unité effaçable. Il contient un nombre de pages multiple de 32. La page représente la plus petite unité adressable pour les opérations de lecture et d'écriture. La page est divisée en

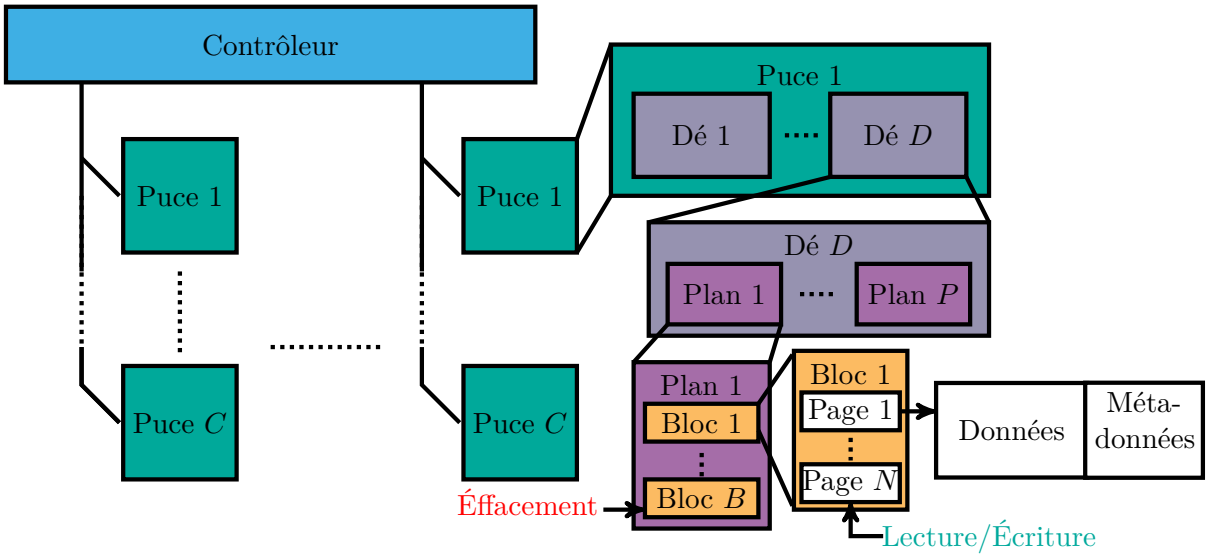


FIGURE 2.8 – Structure interne des disques SSD

deux parties : une partie contenant les données, et une autre partie *spare* utilisée pour stocker les méta-données.

Le contrôleur est responsable de nombreuses fonctionnalités de gestion du SSD, notamment la récupération de place, la gestion des blocs défectueux et la répartition de l'usure (*wear leveling*). Il utilise la couche *Flash Translation Layer* (FTL) pour maintenir l'adressage logique et physique des données.

Contrairement aux HDD, les disques SSD à base de mémoire flash sont plus performants et consomment peu d'énergie [71, 68]. De plus, ils résistent mieux aux chocs et aux températures élevées. Néanmoins, malgré la baisse de leur prix, ils restent plus chers que les disques HDD [72]. En effet, selon [72], les SSD sont deux à dix fois plus chers que les disques HDD pour des quantités de stockage similaires. De plus, leur courte durée de vie représenterait leur contrainte majeure.

2.1.4.3 Services de stockage à base des disques HDD et SSD

Il existe de nombreux services de stockage commerciaux proposés par différents CSP. Google Persistent Disk [73] peut provisionner des volumes sur des disques HDD ou SSD d'une taille allant jusqu'à 64 To. Le débit est directement lié à la taille choisie. Cela signifie que si de bonnes performances sont souhaitées, de grands volumes de stockage sont nécessaires. *Elastic Block Store* (EBS) [74] est un service de stockage fourni par Amazon. Ce service est proposé avec différentes options pour une variété de charges de travail et de cas d'utilisation. En effet, un client peut choisir entre des volumes soutenus par des disques SSD pour les charges de travail plus performantes nécessitant des IOPS élevées ou des disques HDD qui conviennent aux stockage

de gros volumes de données comme les entrepôts de données. La génération actuelle de volumes EBS d'AWS comprend : EBS Provisioned IOPS SSD (io1), EBS General Purpose SSD (gp2), Throughput Optimized HDD (st1) et Cold HDD (sc1). Enfin, nous citons Managed disk [75], un service de stockage de Microsoft Azure. Les types de Managed disk disponibles sont les disques ultra, les SSD premium, les SSD standards et les HDD standards [75].

2.2 Fédération de *Clouds*

La fédération de *Clouds* fait référence à un ensemble de CSP géographiquement répartis qui interagissent les uns avec les autres en partageant leurs ressources de manière coopérative via des marchés centralisés et décentralisés [20, 19, 18, 76]. Cette collaboration aide les CSP à augmenter leurs profits tout en fournissant un approvisionnement continu de services de haute qualité en exploitant la disponibilité temporelle et spatiale des ressources et la diversité des coûts opérationnels [10]. La fédération de *Clouds* permet de surmonter certaines limitations des *Clouds* centralisés traditionnels telles que la contention des ressources, l'interruption de service et la dégradation de la qualité de service [18, 77, 78]. Ces limitations sont apparues avec la maturité de la technologie du *Cloud computing* [20] et l'émergence de nombreux nouveaux paradigmes connexes tels que le *Cloud computing* mobile [79], l'IoT [80] et les applications *Big Data* [81]. Pour fonctionner correctement et maintenir l'intégrité de l'organisation, les CSP fédérés sont régis par un contrat appelé *Federation Level Agreements* (FLA) qui spécifie les règles d'interconnexion et décrit les responsabilités et les comportements autorisés de chaque participant, ainsi que les pénalités financières et administratives à appliquer en cas de violation de ses termes [82].

Aujourd'hui, plusieurs plate-formes académiques et commerciales permettent déjà la réalisation de fédérations de *Clouds* dans la pratique. OnApp Federation [83] est un réseau d'IaaS qui connecte plusieurs CSP. Les CSP interagissent via le marché OnApp, où ils peuvent vendre ou acheter des ressources de calcul à la demande. En outre, EGI *Federated Cloud* [84] est une grille de *Clouds* privés universitaires et de ressources virtualisées, qui est construite selon des normes ouvertes et se concentre sur les exigences de calcul de la communauté scientifique. Enfin, le projet Openlab du CERN [85] vise à construire une fédération transparente entre plusieurs plates-formes de *Clouds* privés et publics via OpenStack.

Dans cette section, nous abordons d'abord les motivations derrière l'adoption des *Clouds* fédérés. Ensuite, une classification des différentes formes de fédérations est présentée. Nous terminons cette section par les défis auxquels la fédération de *Clouds* est confrontée.

2.2.1 Motivations pour la fédération de *Clouds*

La fédération de *Clouds* apporte de nombreux avantages aux CSP et aux utilisateurs. Nous citons, entre autres [20, 19] :

Élasticité et disponibilité : L'élasticité est l'une des principales propriétés du *Cloud computing*. elle permet d'adapter dynamiquement le provisionnement des ressources en augmentant ou en diminuant leur quantité en fonction des charges de travail des clients [86]. La fédération de *Clouds* permet aux CSP d'ajuster efficacement leur capacité d'hébergement en coopérant avec d'autres au lieu de fournir des (nouvelles) capacités supplémentaires, ce qui augmente le gaspillage d'énergie et les coûts. De plus, la répartition géographique des infrastructures de *Clouds* fédérés permet de migrer des services depuis des zones susceptibles d'être affectées par des pannes. Elle permet également de répliquer géographiquement les données, ce qui maintient la disponibilité des services des clients.
Coût et efficacité : La fédération de *Clouds* permet aux CSP d'étendre leur couverture géographique sans avoir à créer de nouveaux centres de données [87]. Cela permet de réduire les coûts et en même temps d'améliorer la qualité de service offerte. En outre, la latence réseau peut être minimisée en satisfaisant les requêtes des clients d'un CSP plus proche et le temps de réponse peut être réduit en exécutant la requêtes sur un hôte plus puissant auprès d'un autre CSP. De plus, avec la fédération de *Clouds*, la charge peut être équilibrée entre les *Clouds* concernés en fonction de certaines mesures telles que la consommation d'énergie.
Gain économique : les infrastructures *Clouds* n'ont pas été conçues pour inter-opérer, il y a donc un manque de standardisation et de compatibilité entre les technologies sous-jacentes. Cela génère le phénomène de *verrouillage du fournisseur*, appelé en anglais *Vendor lock-in*. Ce phénomène fait référence à une pratique dans laquelle les clients peuvent être contraints de rester dépendants d'un CSP particulier pour ses services. Ce verrouillage peut entraîner des pertes économiques et fonctionnelles pour les clients, car la migration de leurs services vers d'autres *Clouds* est coûteuse et nécessite beaucoup d'efforts techniques. Dans une fédération de *Clouds*, le FLA peut être utilisé pour spécifier des interfaces et des protocoles de représentation de données qui doivent être utilisés par les CSP partenaires afin de faire face à ce phénomène [20].

2.2.2 Taxonomie de fédérations de *Clouds*

Il existe deux critères de classification principaux de la fédération de *Clouds*, comme illustré dans la Figure 2.9.
Expansion : Ce type de classification reflète la manière dont les ressources et les services disponibles sont utilisés dans la fédération [20]. Il existe des fédérations verticales, horizontales et hybrides [20, 88]. Dans une fédération verticale, l'expansion se produit entre des *Clouds* à différents niveaux de services, tandis que dans une fédération horizontale, l'expansion se produit entre des *Clouds* au même niveau de service de manière scalable. Enfin, les fédérations hybrides réalisent une expansion horizontale et verticale conformément à l'intérêt des clients et des fournisseurs [78].
Organisation : Ce critère de classification reflète la manière dont les CSP sont organisés. Il

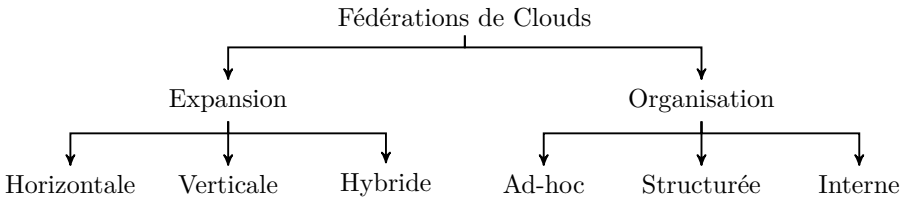


FIGURE 2.9 – Types de fédérations de *Clouds*

existe trois formes de fédérations [78, 18, 89, 78] :

(i) Fédération ad hoc : un CSP, un client ou un *broker* peut utiliser les ressources d'autres CSP publics tels qu'Amazon AWS et Google apps. Dans ce type de fédération, les *Clouds* publics ne sont pas nécessairement conscients qu'ils participent à une fédération. Par conséquent, il n'y a pas de contrat entre ces CSP publics et l'entité qui utilise leurs ressources doit établir un SLA spécifique avec chacun d'entre eux.
(ii) Fédération structurée : cette fédération est représentée par des CSP privés de taille petite et moyenne (entreprises qui gèrent leurs propres infrastructure *Cloud*) qui collaborent entre eux pour maintenir toutes les propriétés du paradigme *Cloud*. Les CSP sont régis par un FLA [20, 19, 18] qui décrit la relation entre les différents CSP à travers des propriétés fonctionnelles et non fonctionnelles. Cela peut être intéressant et utile dans de nombreux environnements critiques. En effet, le FLA détermine le comportement des *Clouds* hétérogènes et autonomes [20]. Par conséquent, cela atténue les complications entre les *Clouds* fédérés car le partage des ressources se fait correctement, conformément à ce contrat [20].
(iii) Fédération interne : ici la fédération de *Clouds* est composée d'un ensemble de centres de données répartis géographiquement appartenant à la même institution. Dans cette classe, il n'y a pas d'accords complexes à maintenir entre les *Clouds* car ils sont gérés par la même entité.

2.2.3 Défis pour la fédération de *Clouds*

la fédération de *Clouds* pose bien plus de défis que le *Cloud* traditionnel [88, 90, 91]. Dans cette section, les principaux défis de la fédération de *Clouds* sont examinés.

2.2.3.1 Interopérabilité et portabilité entre les *Clouds* fédérés

L'interopérabilité des services et leur portabilité entre les *Clouds* sont des exigences clé pour créer un écosystème *Cloud* et tirer pleinement parti de ses propriétés. L'interopérabilité des services est la capacité des utilisateurs du *Cloud* à utiliser leurs données et applications entre plusieurs CSP avec la même interface de gestion, tandis que la portabilité de service est la capacité de porter leurs applications d'un CSP à un autre quel que soit leur choix en termes

de système d'exploitation, format de stockage ou même API. Cela nécessite la définition de protocoles et d'API normalisés pour la gestion distribuée des ressources [92, 64]. Parmi les normes et API les plus adoptées, nous citons : l'*Open Cloud Computing Interface (OCCI)* fournissant une spécification d'une API de gestion RESTful pour le provisionnement et la surveillance des ressources IaaS [93] ; l'Open Virtualization Format (OVF) [94] pour le déploiement des VM sur des plates-formes hétérogènes ; l'API Libcloud [95] qui fait abstraction de l'hétérogénéité entre les *Clouds*. Néanmoins, il faudra du temps pour que ces normes soient prises en charge par les fournisseurs de *Cloud* public [96].

2.2.3.2 Gestion des ressources

La gestion des ressources consiste à trouver le placement optimal et l'affectation des services aux ressources physiques disponibles. C'est une tâche très difficile dans la fédération de *Clouds*. Le processus de prise de décision est particulièrement complexe en raison du nombre croissant des CSP et des paramètres de la fédération, ainsi que de la forte hétérogénéité et la dynamicité de cet environnement. L'optimisation dépend d'une panoplie de critères tels que les charges de travail des clients, les offres partagées des CSP, les contraintes applicatives, les coûts internes et externes [97, 98].

2.2.3.3 Sécurité et confidentialité

La sécurité et la confidentialité sont les principales préoccupations de tout système. Dans une fédération de *Clouds*, les risques d'attaques malveillantes sur le système sont élevés et la confidentialité des données peut être compromise. Les *Clouds* fédérés fonctionnent les uns avec les autres, de sorte que tout fournisseur de *Cloud* peut devenir une source d'attaques sur tous les systèmes. L'intégration des fonctionnalités de sécurité dans l'infrastructure existante est également un défi pour que l'efficacité ne soit pas entravée [99].

2.3 Optimisation multi-objectifs

Nous rappelons que le but de cette thèse est de proposer des approches intelligentes pour un meilleur placement des données dans un *Cloud* membre d'une fédération de *Clouds*. Afin d'optimiser le placement de données, différents coûts liés aux stockage, latence et migration de données doivent être pris en considération. Ces coûts sont corrélés et dans certains cas contradictoires. Par conséquent, une optimisation multi-objectifs est nécessaire.

L'optimisation multi-objectifs (MOO) est un domaine très important de la recherche opérationnelle à cause de la nature multi-objectifs de la plupart des problèmes réels. En effet, MOO consiste à optimiser simultanément de multiple objectifs par rapport à un ensemble de

contraintes. Les problèmes d'optimisation multi-objectifs (MOOP) se posent dans divers domaines [100] tels que les applications industrielles (par exemple, fabrication, conception, gestion) et les applications scientifiques (par exemple, domaines de la bio-informatique et de la chimie, conception de protéines, conception de médicaments, conception de systèmes de ressources en eau, gestion de ressources dans le *Cloud*). Les premiers travaux menés sur les problèmes multi-objectifs furent réalisés au 19$^{\text{ème}}$ siècle sur des études en économie par Edgeworth et généralisés par Pareto. Contrairement à l'optimisation mono-objectif, le but de MOO est de trouver un ensemble de bonnes solutions [101] représentant des compromis entre des objectifs contradictoires puisque l'existence d'une solution unique optimisant tous les objectifs simultanément est très rare pour ce genre de problèmes [102]. Une bonne solution est souvent définie par le concept de dominance. Ce concept est liée au fait qu'aucune solution n'est meilleure que les autres pour toutes les fonctions objectifs (voir la section 2.3.2). Un décideur choisit la solution finale parmi les bonnes [103] en prenant éventuellement en compte des facteurs non inclus dans le processus d'optimisation car ils sont difficiles à modéliser et parfois subjectifs.

Dans la suite de cette section, nous allons décrire les concepts de bases de MOO.

2.3.1 Problème d'optimisation multi-objectifs

Dans MOO, un problème d'optimisation sur n variables de décision peut être défini par l'optimisation (minimisation ou maximisation, minimisation dans la suite) d'un vecteur de k fonctions objectif (*fitness*) tout en satisfaisant un ensemble de m contraintes. Les fonctions objectif modélisent mathématiquement les buts à atteindre. Les contraintes définissent l'espace des solutions réalisables [101]. le problème général est formulé comme suit :

$$\begin{array}{lllll} \min\limits_{x \in X} & f(x) = [f_1(x), ...f_k(x)] & f_i(x) \in Y & i = 1, ..., k \\ \text{Sous contraintes} & h_j(x) \leq 0, & j = 1, ..., m & \end{array}$$

où x est le vecteur des variables de décision (solution), X l'espace de décision, f le vecteur des fonctions objectif, Y l'espace objectif et h_j les contraintes du problème.

2.3.2 Optimalité de Pareto

Les MOOP sont plus difficiles à traiter que les problèmes mono-objectif [37, 101]. La difficulté réside dans le fait qu'il n'y ait pas de relation d'ordre global entre les solutions. Sur certains objectifs, une solution peut être meilleure qu'une autre, tandis que sur d'autres objectifs, ce ne sera pas le cas. Par conséquent, il n'y a pas nécessairement de solution unique qui puisse être optimale pour tous les objectifs en même temps. Dans ce cas, la solution optimale n'est plus une solution unique, mais un ensemble de solutions qui font des compromis entre les différents objectifs à optimiser. La relation d'ordre partiel la plus connue et la plus couramment utilisée

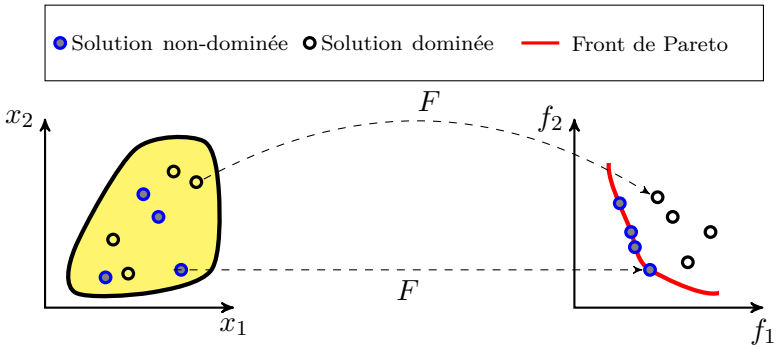


FIGURE 2.10 – Problème d'optimisation multi-objectifs (2 variables de décision et 2 fonctions objectifs)

pour déterminer ces meilleurs compromis est la relation de dominance au sens de Pareto. Une solution optimale au sens de Pareto est une solution qui n'est pas dominée par une autre solution, c'est-à-dire qu'il n'existe pas de solution meilleure pour tous les objectifs considérés. Plus formellement, la relation de dominance de Pareto est définie comme suit [104] :
Définition 1 : Un vecteur $v = (v_1, v_2, ..., v_k)$ domine un autre vecteur $w = (w_1, w_2, ..., w_k)$ si et seulement si $\forall i \in \{1, 2, ..., k\}, v_i \leq w_i$ et $\exists j \in \{1, 2, ..., k\}, v_j < w_j$. Ceci est désigné par $v \prec w$.
Définition 2 : l'ensemble optimal de Pareto est défini par : $P = \{x \in X, \nexists x' \in X, F(x') \prec F(x)\}$
Définition 3 : le front de Pareto est défini par : $PF = \{F(x), x \in P\}$
Définition 4 : une approximation de l'ensemble de Pareto est un ensemble de solutions où aucun élément de cet ensemble n'est dominé par un autre.

La figure 2.10 illustre un exemple d'un MOOP. La partie de gauche montre un espace de décision où les vecteurs de décision (ou solutions) sont composés de deux variables. Les contraintes du problème imposées aux solutions conduisent à un espace réalisable (zone en jaune). Chaque vecteur de décision dans l'espace de décision correspond à un vecteur objectif unique dans l'espace objectif, comme indiqué dans la partie droite. La figure 2.10 montre également l'ensemble de Pareto (les points bleu) et le front de Pareto associé (points reliés par une ligne rouge) qui représente le meilleur ensemble de compromis pour les objectifs considérés.

2.3.3 Techniques d'optimisation multi-objectifs

La résolution d'un MOOP doit passer par deux étapes. (i) La recherche de l'ensemble de Pareto : c'est la phase d'optimisation multi-objectifs, (ii) le choix de la solution à retenir dans l'ensemble de Pareto : c'est la tâche du décideur qui, parmi l'ensemble des solutions de compromis, doit sélectionner celle qu'il utilisera. Cette phase de décision multi-objectifs fait appel à la théorie de la décision. Dans le cadre de ce travail, nous nous intéressons à la première phase.

Le temps de résolution des MOOP augmente avec la croissance du nombre de variables et

d'objectifs. Plusieurs méthodes de résolution ont été proposées pour traiter MOOP en fonction de leur complexité. Les différentes techniques proposées peuvent être classifiées en deux catégories [105, 106, 107, 108] :

2.3.3.1 Approches non Pareto :

Les approches non Pareto ne traitent pas le problème comme un véritable problème multi-objectifs. Ces approches transforment le problème initial en un ou plusieurs problèmes mono-objectifs. Ces approches sont classées en :

(i) Approches agrégatives : Ces approches transforment le problème multi-objectifs en un problème mono-objectif. Parmi les méthodes qui utilisent cette approche il y a la *méthode d'agrégation par pondération* [109]. Cette méthodes est l'une des premières approches utilisées pour résoudre un MOOP. Elle consiste à transformer un problème multi-objectifs en un problème mono-objectif en pondérant et agrégeant les différentes fonctions objectifs en une seule fonction F comme illustré dans l'équation suivante :

$$F(x) = \sum_{i=1}^{k} \lambda_i * f_i(x) \qquad \lambda_i \in [0..1] \text{ avec} \sum_{i=1}^{k} \lambda_i = 1$$

La *méthode de ϵ-contraintes* [110] fait aussi partie des approches agrégatives. Elle consiste à optimiser une seule fonction objectif sujette à des contraintes sur les autres fonctions objectif. L'approche agrégative contient également la *méthode de Goal Programming* où le décideur doit définir des buts T_i qu'il désire atteindre pour chaque objectif f_i selon l'équation suivante :

$$\min \sum_{i=1}^{k} f_i(x) - T_i$$

Ces buts sont introduits dans la formulation du problème, le transformant en un problème mono-objectif. La nouvelle fonction objectif est modifiée de façon à minimiser la somme des écarts entre les résultats et les buts à atteindre. La technique min-max [111] est considéré comme une *méthode de Goal Programming*

(ii) Approches par traitement séparé : Dans cette catégorie, des opérateurs sont utilisés afin de traiter séparément les différentes fonctions objectifs. Cette approche est représentée par la méthode de sélection parallèle [112] (appelée VEGA : Vector Evaluated Genetic Algorithm) et la méthode de sélection lexicographique [113]. L'idée de la première méthode consiste à sélectionner les solutions selon chaque objectif de manière indépendante. Si le problème est défini par K objectifs et un ensemble de N solutions, une sélection de $\frac{N}{K}$ solutions est effectuée pour chaque objectif. De cette manière K sous-ensembles de solutions sont créées, chacune pour un objectif particulier. Ensuite, les K sous-ensembles sont fusionnées afin de construire un nouvel ensemble

de taille N.

La deuxième technique classe les objectifs suivant un ordre d'importance défini à priori par le décideur. La fonction objectif la plus importante est d'abord optimisée. Ensuite, la deuxième plus importante fonction est optimisée en intégrant la valeur de la première fonction comme contrainte pour sa résolution. Ce processus se répète sur les fonctions moins prioritaires en intégrant toujours les valeurs obtenues comme contraintes.

2.3.3.2 Approches Pareto :

Ces approches sont basées sur la dominance au sens de Pareto présentée précédemment dans ce chapitre (voir la section 2.3.2). Elles ne transforment pas les objectifs du problème, ceux-ci sont traités sans aucune distinction pendant la résolution. Les algorithmes de cette approche fournissent au décideur un ensemble de solutions en une seule résolution du problème. Ils ont prouvé leur efficacité dans plusieurs travaux [104, 114]. Une panoplie d'algorithmes évolutionnaires multi-objectifs, qui sont l'objet de la section suivante, font partie de cette catégorie.

2.3.4 Algorithmes évolutionnaires multi-objectifs

Les algorithmes évolutionnaires multi-objectifs (*Multi-Objectif Evolutionary Algorithm* : MOEA) [115, 116, 117, 118, 119] sont des algorithmes de population non déterministes proposés et utilisés pour résoudre les MOOP. Ils sont généralement basés sur la théorie de la sélection naturelle darwinienne dont le but est de faire évoluer une population vers des individus de bonne qualité. Ces algorithmes aident les décideurs à trouver un ensemble de Pareto dans un délai raisonnable au cas où les méthodes exactes ne seraient pas applicables. C'est généralement le cas pour les problèmes de grande échelle en raison du temps d'exécution prohibitif que leur résolution nécessite.

Tous les algorithmes évolutionnaires définissent un ensemble de solutions candidates, qui sont appelées "population" dans la littérature. Ces solutions sont généralement initialisées de manière aléatoire. La population subit itérativement trois étapes : i) évaluation, ii) sélection et iii) variation (voir la figure 2.11).

Dans l'étape d'évaluation, pour chaque individu, une valeur de *fitness* est calculée. Cette valeur mesure la qualité des solutions suivant une ou plusieurs métriques. En se basant sur ces valeurs, l'étape de sélection choisit les solutions les mieux évaluées pour se reproduire. Le choix se fait de manière déterministe ou stochastique. Les individus choisis (parents) passent par le processus de variation (reproduction) et génèrent des descendants. L'ensemble du processus d'évolution se répète sur la population des descendants jusqu'à ce que le critère d'arrêt soit satisfait.

Les algorithmes multi-objectifs évolutionnaires (MOEA pour *Multi-Objectif Evolutionary Algorithm*) standards ne fonctionnent pas bien parfois pour des problèmes difficiles avec des espaces

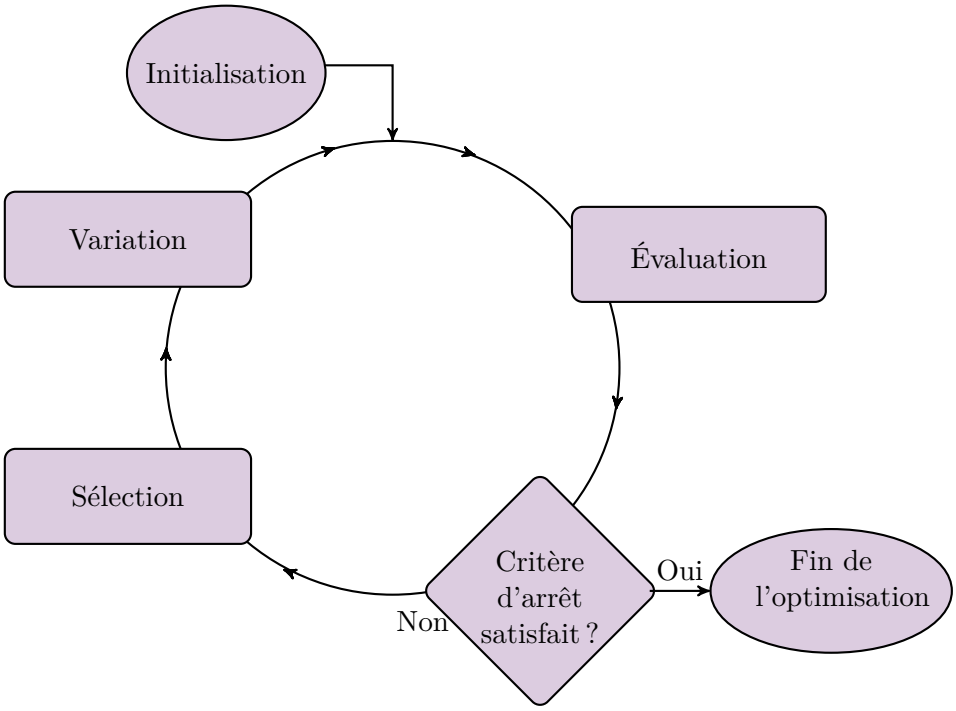


FIGURE 2.11 – Processus d'algorithme évolutionnaire

de recherche complexes ou un nombre élevé de contraintes [120]. Les heuristiques de réparation, les méthodes de pénalisation des solutions et les représentations et opérateurs spéciaux sont les techniques classiques de gestion des contraintes utilisées dans les algorithmes évolutifs. Concevoir des représentations et des opérateurs spéciaux peut être utile, mais prend également du temps et ne tire pas parti des travaux antérieurs. Les méthodes de pénalisation sont les plus anciennes approches utilisées pour gérer les contraintes [121]. La pénalisation consiste à ajouter/soustraire une certaine valeur à/de la fonction objectif des solutions infaisables selon le degré de violation des contraintes [121]. Néanmoins, ces techniques ne garantissent pas de trouver des solutions réalisables. Les opérateurs de réparation consistent à réduire l'espace de recherche à des solutions réalisables en effectuant des recherches locales afin de transformer des solutions irréalisables en solutions réalisables proches dans l'espace de recherche. Les opérateurs de réparation évitent les évaluations répétitives et coûteuses de solutions irréalisables maintenues par les méthodes de pénalisation et gèrent directement les contraintes du problème [120]. De plus, un opérateur de réparation permet d'utiliser tous les opérateurs standards bien connus et efficaces.

Les travaux réalisés dans le cadre de cette thèse sont basés sur l'algorithme génétique de tri non dominé version II appelé en anglais Non Dominated Sorting Genetic Algorithm version II (NSGAII). Dans ce qui suit, nous allons détailler le principe de fonctionnement de cet algorithme.

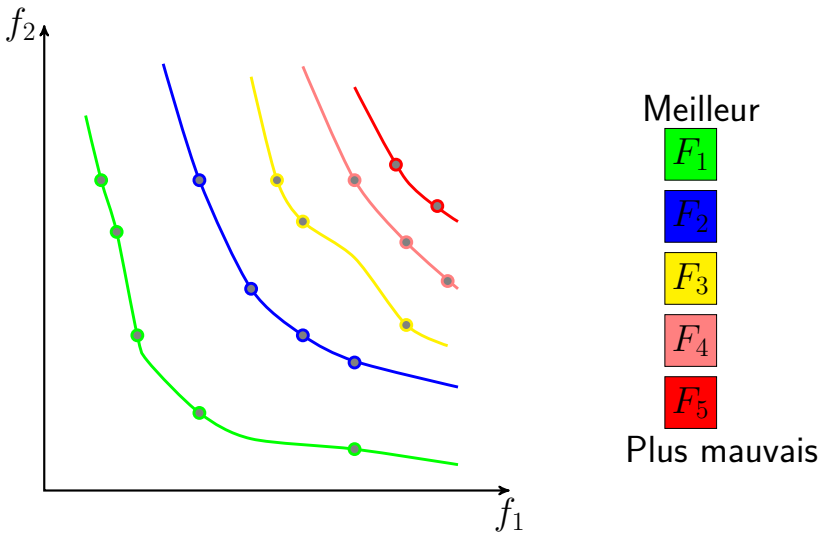


FIGURE 2.12 – Tri non dominé

2.3.5 Algorithme génétique de tri non dominé version II

NSGAII, proposé par Kalaynmoy Deb [115] en 2002, constitue l'un des algorithmes génétiques populaires résolvant des problèmes multi-objectifs grâce à ses performances avérées et son fonctionnement simple et efficace [122, 123, 124, 125, 126]. Cet algorithme est une version amélioré de l'algorithme NSGA qui manque d'élitisme et a une complexité de calcul de $O(MN^3)$ pour M objectifs et une taille de population N.

NSGAII se distingue par sa complexité polynomiale qui est égale à $\mathcal{O}(MN^2)$. Il permet de trouver une meilleure répartition des solutions et une bonne convergence près du vrai front de Pareto [127]. Il fonctionne selon les trois modules suivants : (i) le tri non dominé (*non-dominated sorting*), (ii) l'affectation des distances d'encombrement (*crowding distance assignment*), et (iii) l'opérateur de comparaison encombré (*crowded comparison operator*).

Dans ce qui suit, nous décrivons ces trois modules et donnons leur principe de fonctionnement.

2.3.5.1 Tri non dominé

Le but du tri non dominé est de trier les solutions de la population en différents fronts non dominés en fonction des valeurs de la fonction objectif. Le premier front non dominé contient toutes les solutions du premier niveau non dominé, ce qui signifie que les solutions de ce front sont les meilleures parmi la population car elles ne sont dominées par aucune autre solution en termes de valeurs de fonction objectif. Les solutions du deuxième front non dominé sont dominées par la solution du premier front non dominé, mais domine les solutions du troisième front non dominé et ainsi de suite. Une représentation graphique basée sur [115] est fournie dans la figure 2.12, où la population est triée en cinq fronts : F_i, $i = 1, 2, ...5$.

L'algorithme 1 décrit le fonctionnement de la procédure de tri non dominé proposée dans [115].

Algorithme 1 Algorithme de tri non dominé

```
1:  Procedure TRI-NON-DOMINÉ(Population P)
2:      Pour chaque p ∈ P Faire
3:          S_p ← ∅
4:          n_p = 0
5:          Pour chaque q ∈ P Faire
6:              Si p ≺ q Alors
7:                  S_p = S_p ∪ q
8:              Sinon Si q ≺ p Alors
9:                  n_p = n_p + 1
10:             Fin Si
11:             Si n_p = 0 Alors
12:                 p_rank = 1
13:                 F_1 = F_1 ∪ p
14:             Fin Si
15:         Fin Pour
16:     Fin Pour
17:     i = 1
18:     Tant que (F_i ≠ ∅ ) Faire
19:         Q ← ∅
20:         Pour chaque p ∈ F_i Faire
21:             Pour chaque q ∈ S_p Faire
22:                 n_q = n_q − 1
23:                 Si n_q = 0 Alors
24:                     q_rank = i + 1
25:                     Q = Q ∪ q
26:                 Fin Si
27:             Fin Pour
28:         Fin Pour
29:         i = i + 1
30:         F_i = Q
31:     Fin Tant que
32: Fin Procedure
```

Étape 1 : initialisation de tri non dominé, identification des solutions appartenant à F_1 et celles n'y appartenant pas

Étape 2 : identification des autres fronts non dominés et attributions des rangs à leurs solutions

La première phase de l'algorithme 1 (les lignes 2 à 16) initialise le tri de manière efficace. Pour se faire, deux entités sont introduites : (i) le nombre de domination n_p, et l'ensemble S_p. n_p est un indice attribué à chaque solution p dans la population P. Il compte le nombre total de solutions dans la population qui domine la solution p. L'ensemble S_p contient toutes les solutions de la population P que la solution p domine. Initialement, n_p est mis à zéro et S_p est un ensemble

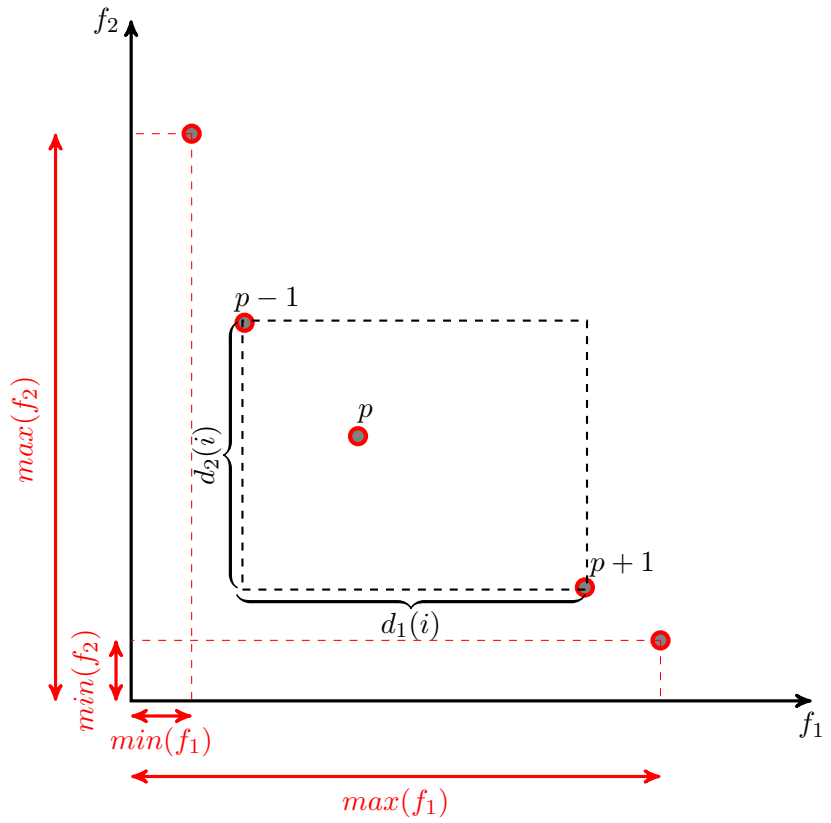


FIGURE 2.13 – Distance d'encombrement

vide. De plus, pour toutes les solutions du premier front non dominé F_1, n_p reste nul tout au long du processus de tri. A ce stade, toutes les solutions n'appartenant pas au premier front non dominé F_1 sont contenues dans l'ensemble S_p et leur n_p est incrémenté par 1. Ainsi, le premier front est à ce moment identifié. Un rang de $p_{rank} = 1$ est attribué aux solutions de ce premier front.

Les lignes (17 à 31) décrivent la deuxième phase de l'algorithme 1. Dans cette phase, les fronts non dominés sont identifiés. Pour chaque solution avec $n_p = 0$, chaque solution q, contenue dans l'ensemble correspondant S_p décrémente son nombre de domination n_q par 1. Après cela, toutes les solutions q de l'ensemble S_p en question ayant $n_q = 0$ sont mises dans une liste séparée Q. Les membres de la liste Q appartiennent au deuxième non dominé F_2, et sont attribués un rang égal à 2. Le processus se poursuit afin d'identifier le troisième front non dominé F_3 et ainsi de suite. Ceci est fait comme un processus itératif jusqu'à ce que tous les fronts non dominés F_i, $i = 1, 2, 3, ...$, dans la population soient identifiés et que toutes les solutions se voient attribuer un rang.

2.3.5.2 Affectation des distances d'encombrement

Dans les algorithmes génétiques, il est préférable de préserver la diversité des solutions au sein de l'espace de recherche. Pour déterminer la diversité entre les différents solutions dans la population, NSGAII calcule la distance d'encombrement pour chaque solution au sein de chaque front non dominé. La distance d'encombrement $CD(p)$ d'une solution p d'un front F_i est calculé comme suit :

$$CD(p) = \sum_{j=1}^{n_{obj}} \frac{d_j(p)}{\max(f_j) - \min(f_j)} \tag{2.1}$$

Où $d_j(p) =\mid f_j(p+1) - f_j(p-1) \mid$ est la distance séparant les deux plus proches voisins de la solution p. Les distances d'encombrement des solutions extrêmes (c'est-à-dire les solutions ayant les plus petites et les plus grandes valeurs des fonctions objectifs du front) se voient attribuer une valeur infinie afin de les préserver dans la génération suivante pour la diversification des solutions. Le calcul de la $CD(p)$ pour un individu p est illustré dans un cas bi-objectif dans la figure 2.13.

L'algorithme d'affectation des distances d'encombrement proposé dans [115] est reproduit dans l'algorithme 2. Le front F_i contient les solutions $F_i[p]$, $p = 1, 2, 3, ..., l$ chacune à laquelle est affectée une distance d'encombrement $DC(F_i[p])$. $\min(f_j)$ et $\max(f_j)$ désigne le maximum et le minimum de la fonction objectif f_j.

Algorithme 2 Algorithme de calcul de la distance d'encombrement

Procedure DISTANCE-ENCOMBREMENT(F_i)
  $l \leftarrow$ nombre de solutions dans le front F_i
  Pour chaque fonction objectif j **Faire**
    $F_i \leftarrow ordonner(F_i, j)$ //Tri par ordre croissant des solutions de F_i selon la fonction f_j
    $CD(F_i(1)) \leftarrow \infty$ *//attribution de l'infinie à la solution ayant la plus petite valeur de* f_j
    $CD(F_i(l)) \leftarrow \infty$ *//attribution de l'infinie à la solution ayant la plus grande valeur de* f_j
    Pour p=2 à (l-1) **Faire**
      Pour j=1 à n_{obj} **Faire**
        $CD(F_i(p)) \leftarrow CD(F_i(p)) + \frac{f_j(p+1) - f_j(p-1)}{\max(f_j) - \min(f_j)}$ *distance d'encombrement de la solution p*
      Fin Pour
    Fin Pour
  Fin Pour
Fin Procedure

2.3.5.3 Opérateur de comparaison d'encombrement

Après avoir introduit le tri non dominé et le calcul de la distance d'encombrement, l'opérateur de comparaison d'encombrement peut être défini. Le processus de sélection utilise l'opérateur de comparaison d'encombrement afin de garantir la diversité parmi les solutions. En bref, l'opéra-

teur de comparaison d'encombrement compare deux solutions pour leur degré de proximité avec d'autres solutions. Il prend également en compte le rang des solutions calculé par l'algorithme de tri non dominé (voir l'algorithme 1), car des rangs inférieurs sont préférables. Si deux solutions ont le même rang, la solution située dans une région moins encombrée, c'est-à-dire la solution avec la distance d'encombrement la plus élevée, est préférée. Prendre en compte les deux solutions p et p', avec leur rang correspondant p_{rank}, $p\prime_{rank}$ et distance d'encombrement $p_{crowded-distance}$ et $p\prime_{crowded-distance}$, le critère pour que la solution p domine la solution p', c'est-à-dire, $p \prec p'$ est donné dans l'algorithme 3.

Algorithme 3 Algorithme d'opérateur de comparaison d'encombrement

1: **Procedure** OPÉRATEUR-COMPARAISON-ENCOMBREMENT(Solutions : p, p')
2: **Si** ($p_{rank} < p'_{rank}$) ou ($p_{rank} = p'_{rank}$ and $p_{crowded-distance} > p'_{crowded-distance}$) **Alors**
3: $p \prec p'$
4: **Fin Si**
5: **Fin Procedure**

2.3.5.4 Fonctionnement de NSGAII

Le passage de la génération G à la génération $G+1$ se fait selon le processus illustré en figure 2.14. Ce processus est répété jusqu'à ce qu'un critère d'arrêt soit atteint. Chaque génération est composée d'une population de taille N_{pop} d'individus. Dans la figure 2.14, il est supposé que les individus de la génération G ont déjà été évalués.

Tout d'abord, les individus de la génération G sont sélectionnés selon un tirage au sort aléatoire sous forme d'un tournois 2 à 2 et sont comparés selon la dominance de Pareto. Les individus ayant remporté ces tournois sont ainsi sélectionnés et forment le groupe reproducteur. Il est à noter que selon le caractère stochastique de ce processus de sélection, un même individu peut être présent en plusieurs exemplaires dans le groupe reproducteur. Les opérateurs de croisement et de mutation sont ensuite appliqués sur les individus de ce groupe reproducteur.

Après la création des enfants et leur évaluation, ces derniers sont fusionnés avec les individus de la génération G. Ainsi, l'algorithme opère maintenant sur une population de taille égale à $2 * N_{pop}$ individus. Ces individus sont classés en fronts non dominés selon l'algorithme 1 et sont triés selon leurs distances d'encombrement au sein de chaque front en appliquants les algorithmes 2 et 3. C'est cette phase qui renferme le caractère élitiste de NSGAII puisque le processus de classement est opéré sur les parents et les enfants.

Enfin, la génération $G+1$ est constitué en choisissant N_{pop} individus des meilleurs premiers fronts. Sur la figure 2.14, les individus des fronts F_1 et F_2 sont ajoutés à la population de la génération $G+1$. Comme le front F_3 contient trop d'individus pour être complètement ajouté à la génération $G+1$, les places restantes disponibles dans cette génération sont alors attribuées aux

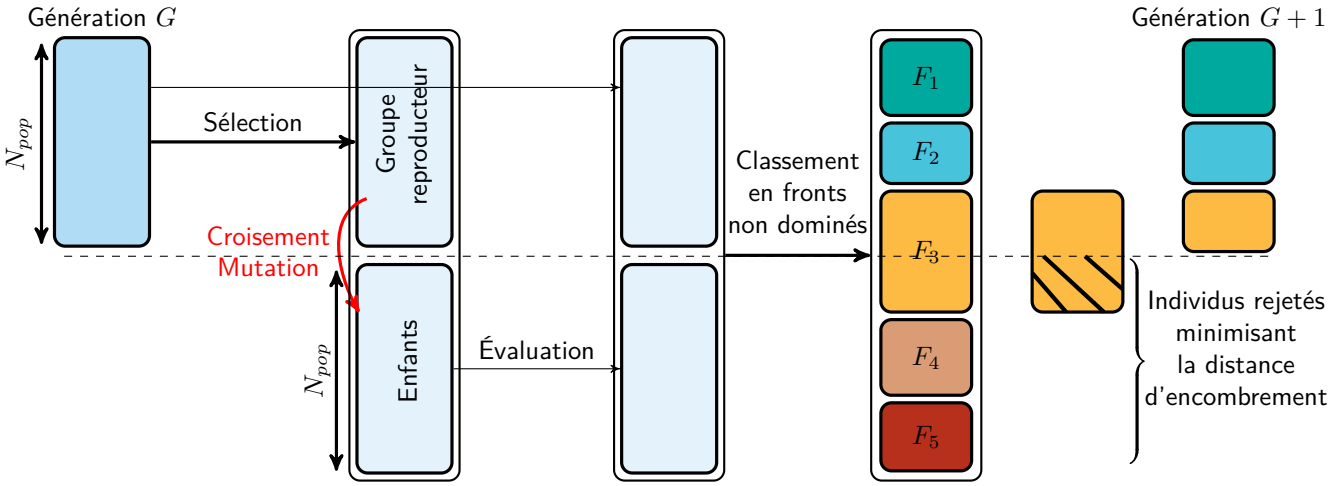


FIGURE 2.14 – Fonctionnement de NSGAII [8]

individus du front F_3 les plus isolés dans l'espace des objectifs, c'est-à-dire, ceux qui maximisent la distance d'encombrement calculée par l'algorithme 2.

NSGAII permet aussi de faire des optimisations en présence des contraintes. Dans ce cas, la comparaison des individus est un peu modifié selon ce qui suit :

— si les deux individus ne vérifient pas les contraintes, l'individu le plus satisfaisant est sélectionné.

— Si seulement un des individus ne satisfait pas les contraintes, c'est l'autre individu qui sélectionné.

— si les deux individus satisfassent les contraintes, ils sont comparés selon l'opérateur de comparaison d'encombrement.

2.3.6 Critères de qualité des algorithmes multi-objectifs

Trois critères doivent être pris en compte lors de la résolution d'un problème d'optimisation multi-objectifs. Ces critères sont [128, 129] (i) *la précision :* les solutions non dominées trouvées doivent être aussi proches que possible du front Pareto-optimal, (ii) *l'exhaustivité* : les solutions doivent être suffisamment nombreuses par rapport au spectre de valeurs du front de Pareto dans les différentes dimensions et (iii) *la diversité :* les solutions trouvées doivent être bien réparties dans ce spectre du front de Pareto.

Pour faire face à ces problématiques, les algorithmes évolutionnaires multi-objectifs s'appuient sur des mécanismes qui favorisent à la fois l'*exploration* (c'est-à-dire explorer des régions non encore visitées de l'espace de recherche) et l'*exploitation* (c'est-à-dire exploiter des solutions

prometteuses précédemment trouvées). La qualité d'un algorithme d'optimisation multi-objectifs résulte en grande partie dans l'équilibre entre ces deux aspects.

2.3.7 Indicateurs de performance

Dans tout problème d'optimisation, après la modélisation et le développement d'un algorithme de résolution, vient la phase d'évaluation de la qualité des solutions produites.

Dans l'optimisation mono-objectif, la comparaison de solutions est triviale. Evaluer une approximation d'un ensemble de Pareto est une tâche plus ardue [130]. En effet, l'existence d'un ensemble de solutions de compromis et l'absence d'ordre total entre ces solutions rendent la mesure de qualité d'un front difficile.

Pour quantifier et comparer la qualité des ensembles de solutions produits par différents MOEA, de nombreux indicateurs de performance ont été proposés dans la littérature. Ces indicateurs de performance consistent à attribuer des scores aux approximations du front de Pareto [130, 131]. Nous citons, entre autres, *la distance générationnelle*, *la distance générationnelle inversée*, *l'hypervolume*, *l'erreur frontale maximale de Pareto* et *l'espacement*. Ces indicateurs permettent de comparer les fronts de Pareto, en termes de précision, d'exhaustivité et de diversité ou toute combinaison de ces critères. Dans nos évaluations, nous avons utilisé l'indicateur *hypervolume*.

L'hypervolume (HV), également appelé *hyperarea*, est l'une des mesures les plus populaires pour l'évaluation des performances des algorithmes multi-objectifs [132]. Le HV englobe les trois critères de qualité, c'est à dire, l'exhaustivité, la précision et la diversité [131, 130]. Cet indicateur, proposé par [133], calcule la partie de l'espace objectif dominée par le front de Pareto délimité par un point de référence r. r désigne une borne supérieure sur tous les objectifs. Il doit être au moins dominé par toutes les solutions de l'approximation considérée. Comme ce point n'est généralement pas connu, il est principalement estimé comme la pire valeur possible dans l'espace objectif. Le HV d'un front de Pareto approximatif PF_{approx} est obtenu par $\lambda_k(\bigcup_{z \in PF_{approx}} [z; r])$ où λ_k est la k-mesure de Lebesgue dimensionnelle [130]. Plus la valeur HV est élevée, meilleure est l'approximation. Une illustration est donnée dans la figure 2.15 pour le cas bi-objectif ($k = 2$). Les points noirs représentent le front de Pareto et le point rouge est le point de référence. Le HV est représenté par la zone verte.

2.4 Conclusion

Le présent chapitre a introduit les connaissances essentielles pour le travail de cette thèse. Nous avons résumé les principes de base concernant le *Cloud computing* , la fédération de *Clouds* et l'optimisation multi-objectifs.

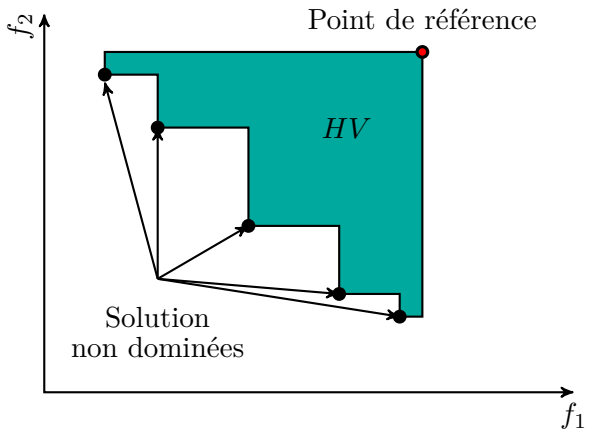


FIGURE 2.15 – Hypervolume

Dans ce travail, nous nous intéressons au problème de placement d'objets d'un CSP membre d'une fédération horizontale structurée qui est gérée par un FLA. Le CSP concerné peut utiliser ses propres ressources de stockage (HDD et SSD) ou celles des autres membres fédérés afin d'optimiser ses coûts tout en respectant le SLA de ses clients. Les performances de stockage d'E/S et la latence du réseau sont les métriques ciblées pour les clients.

Minimiser le coût de placement est contradictoire avec la satisfaction des exigences des clients. Par ailleurs, la satisfaction des performances d'E/S de stockage et de la latence du réseau peuvent être contradictoires dans certaines circonstances. Ainsi, des compromis sont inévitables pour ce type de problème de placement de données ce qui nous pousse à avoir recours à l'optimisation multi-objectifs.

Résoudre un MOOP doit passer par deux étapes (i) trouver l'ensemble de Pareto, (ii) sélectionner une solution parmi l'ensemble de Pareto à travers un processus de prise de décision. Dans ce travail, nous nous intéressons à la première étape.

Le chapitre suivant décrit les travaux de l'état de l'art connexes à nos contributions.

Chapitre 3

Placement de données dans le Cloud : État de l'art

La fédération de *Clouds* permet aux fournisseurs de coopérer et de partager leurs infrastructures pour répondre aux exigences des clients. Un défi majeur pour les fournisseurs fédérés est de définir des stratégies efficaces pour le placement de données afin de tirer pleinement parti de cette coopération. Mettre en adéquation les besoins des clients et les objectifs des fournisseurs pour un placement efficace des données dans une infrastructure de *Cloud* fédéré soulève plusieurs défis en raison de la complexité de cet environnement tels que l'hétérogénéité des ressources et de leurs prix ainsi que de la variabilité de la charge de travail des clients.

Parmi les principaux moteurs de migration vers le *Cloud* on retrouve le coût des services de stockage [134]. Dans cet esprit, les CSP essayent de minimiser le coût de placement de données de leurs clients tout en respectant leurs exigences. Afin d'optimiser ce coût, le choix d'une configuration de placement de données appropriée devient une tâche non triviale.

Dans ce chapitre, nous présentons dans un premier temps les travaux relatifs à l'évaluation du coût de stockage dans les environnements *Clouds*. Ensuite, nous étudions les travaux de placements de données dans les *Clouds* centralisés à base de systèmes de stockage hybrides et les *Clouds* inter-connectés avec un accent particulier sur les modèles de placement basés sur le coût. L'analyse de la littérature repose sur plusieurs aspects tels que, les objectifs et les méthodes d'optimisation utilisées.

Sommaire

Référence	Stockage			Migration	Pénalité
	Occupation	Énergie	Endurance		
[135]	✓				
[136, 137, 138, 139]	✓	✓			
[140, 141, 142]				✓	
[143]	✓	✓	✓		
[144]		✓	✓	✓	✓
[145]	✓	✓	✓	✓	✓

(a) Modèles de coût à base de systèmes de stockage hybrides

Références	Stockage	Trafic	Geo-migration	Hétérogénéité	Fédération
[146]	✓				✓
[147, 148, 149]	✓	✓			
[150]	✓	✓	✓	✓	
[151, 152, 153]			✓		
[18]				✓	✓
[19, 154, 153]	✓			✓	✓

(b) Modèles de coût basés sur les *Clouds* inter-connectés

Tableau 3.1 – Classification des travaux relatifs à l'évaluation du coût de placement de données

3.1 Coût de placement de données

Le coût opérationnel d'un *Cloud* a une grande importance économique [155]. Son optimisation repose sur des modèles de coût, qui doivent donc être le plus réaliste possible. En se penchant sur la littérature, plusieurs travaux ont abordé l'estimation des coûts des systèmes de stockage [145, 150, 136, 140, 147, 153]. Ces travaux peuvent être classés en deux catégories : (i) modèles de coût à base de systèmes de stockage hybrides, (ii) modèles de coûts basés sur les *Clouds* inter-connectés. Les tableaux 3.1a et 3.1b présentent une synthèse de ces travaux. Les lignes représentent les travaux connexes tandis que les colonnes représentent l'ensemble des coûts pris en compte par ces travaux. Une encoche ✓ dans une cellule confirme la prise en compte du coût correspondant, tandis qu'une cellule vide indique que le coût correspondant n'est pas pris en compte dans le modèle.

3.1.1 Modèles de coût à base de systèmes de stockage hybrides

De nombreux efforts ont été déployés pour estimer le coût de placement de données dans des systèmes de stockage hybrides combinant plusieurs classes de stockage comme HDD, SSD, etc.

Les travaux existants ont principalement porté sur des *Clouds* centralisés.

Selon l'état de l'art, le coût de placement de données dans un contexte de *Cloud* centralisé à base de système de stockage hybride inclut le coût de stockage, le coût de migration et le coût de pénalité. Ces trois coûts sont expliqués ci-dessous.

Coût de stockage : ce coût regroupe les coûts étroitement liés au placement de données sur les périphériques de stockage. Il a été estimé à l'aide du coût d'occupation, du coût énergétique ainsi que du coût d'usure. Dans [135], seulement le coût d'occupation a été considéré pour l'évaluation du coût de stockage. Les approches proposées dans [136, 137, 138, 139] ont calculé le coût de stockage en fonction de l'occupation et de l'énergie consommée. Enfin, les travaux présentés dans [143, 144, 145] ont considéré les trois coûts de stockage.

Coût de migration : ce coût est lié au mouvement des objets entre les périphériques des différentes classes de stockage. En effet, la mise en œuvre d'une configuration de placement de données peut nécessiter le changement d'emplacement de certains objets ce qui engendre l'exécution d'une charge de travail supplémentaire consommant de l'énergie, et contribuant à l'usure des supports de stockage. Comme le montre le tableau 3.1a , le coût de migration a été pris en compte dans plusieurs travaux dans la littérature [140, 141, 142, 144, 145]. Il est à noter que dans [140, 141, 142], le coût de migration a été évalué seulement en fonction de l'énergie consommée par la migration des objets entre les classes de stockage tandis que dans [144, 145], les deux coûts (énergie et endurance) ont été pris en considération.

Coût de pénalité : dans le contexte du *Cloud computing*, le SLA est très important car il spécifie un ensemble de termes pour garantir la QoS demandée par les clients. Si la QoS offerte par le fournisseur de *Cloud* est inférieure à celle demandée par le client dans le SLA, alors une pénalité est appliquée au fournisseur. Ceci peut être engendré par un mauvais placement de données. Généralement, la pénalité est exprimée par une remise sur la facture globale du client. Cette remise dépend de plusieurs paramètres notamment la durée et le degré de la violation. Il est à noter que peu de travaux de l'état de l'art ont considéré le coût de pénalité dans le coût global de stockage. Parmi ces travaux, nous citons [144, 145].

3.1.2 Modèles de coût basés sur les *Clouds* inter-connectés

De nombreux travaux existants ont été portés sur le coût de placement des objets de données dans les *Clouds* inter-connectés. Différents aspects ont été abordés dans ces travaux qui sont (i) le coût de stockage, (ii) l'hétérogénéité des services de stockage, (iii) le coût du trafic réseau, (iv) la géo-migration des objets entre les différents centres de données et enfin (v) l'environnement fédéré (voir le tableau 3.1b). La plupart des travaux ont utilisé un coût de stockage fixe principalement lié à l'espace de stockage occupé [146, 147, 148, 149, 19, 154, 153]. La charge de travail d'E/S a été rarement prise en compte. Dans [150], la charge de travail a été calculée en fonction des opérations de lecture et écriture de haut niveau. Le coût du trafic réseau a été abordé dans

des travaux liés à l'exécution des applications Big Data comme les réseaux sociaux [150] et les workflows [147, 148, 149]. Dans [150], le coût de la géo-migration des objets entre les centres de données a été pris en considération. L'hétérogénéité des ressources a été prise en compte dans [150, 18, 19, 154, 153]. Enfin, divers modèles de coûts pour les *Clouds* fédérés ont été proposés. Toosi et al [18] ont conçu un modèle de coût qui fixe dynamiquement le coût des VM fédérées en fonction de la quantité de ressources inutilisées des CSP. Dans [146], un modèle de coût, comprenant les coûts d'internalisation et d'externalisation des ressources virtuelles, a été proposé. Dans [19, 154], les modèles de coût comprennent les coûts locaux, les coûts d'internalisation, les coûts d'externalisation et ceux du trafic réseau.

3.1.3 Discussion

Après avoir décrit les différents modèles de coût de l'état de l'art, nous avons constaté que leurs principales limites peuvent être récapitulées comme suit :

— Les modèles de coût de stockage hybride existants destinés aux *Clouds* centralisés ne sont pas applicables lorsqu'il s'agit de *Clouds* inter-connectés car ces derniers sont plus complexes. En effet, les modèles proposés n'intègrent pas les exigences et les coûts de communication requis par les *Clouds* inter-connectés. Cela peut conduire à une mauvaise estimation du coût de placement de données dans les *Cloud* inter-connectés.

— En ce qui concerne les *Clouds* inter-connectés, le coût du système d'E/S n'a pas été examiné de manière précise. En effet, le coût de stockage a été généralement évalué en fonction de l'occupation.

— La violation des termes de SLA peut engendrer des coûts supplémentaires. Ces coûts de pénalité sont devenus de plus en plus contraignant pour les fournisseurs de *Cloud* [156]. Les métriques SLA liées au stockage ont été rarement prises en compte, en particulier lorsqu'il s'agit d'un *Cloud* inter-connectés.

Il est donc nécessaire de proposer un modèle de coût prenant en compte les différents paramètres liés aux environnements des *Clouds* fédérés.

Une fois ce modèle établi, il pourra être utilisé dans toute approche de placement de données. La prochaine section parcourt différentes approches de placement de données présentées dans la littérature traitant des problèmes de placement de données dans les *Clouds*.

3.2 Placement de données dans le *Cloud*

Le placement efficace des données est l'un des problèmes clés dans les environnements *Cloud*. Ce problème, généralement classé NP-Hard [157, 158], présente un intérêt primordial pour les fournisseurs et les utilisateurs du *Cloud*. Il consiste à trouver la meilleure affectation des objets

des clients ayant différentes tailles et charges de travail sur les ressources de stockage locales et éventuellement distantes disponibles. Ces ressources possèdent des capacités, des performances et des modèles de tarification hétérogènes. Cette affectation présente plusieurs défis et est motivée à la fois par les exigences des clients (par exemple, SLA, localisation, latence) et les objectifs commerciaux du fournisseur (par exemple, optimisation de ses coûts, amélioration de sa réputation).

Dans le reste de cette section, nous allons étudier le problème de placement de données dans les *Clouds* centralisés et dans les *Clouds* inter-connectés. Nous analysons les travaux de l'état de l'art selon les quatre dimensions suivantes :

— Objectifs : Les stratégies d'optimisation sont exécutées en prenant en considération un ou plusieurs objectifs. Différents objectifs ont été considérés par les travaux de placements des objets dans les environnements *Clouds*. Ces objectifs diffèrent selon la plate-forme considérée. Nous catégorisons les travaux existants selon les trois objectifs suivants :

1. Minimisation du coût de stockage : divers sous-coûts ont été pris en compte. Pour le stockage de données, les sous-coût sont le coût d'occupation, le coût de l'exécution de la charge de travail et le coût lié aux violations des termes du SLA.
2. Minimisation du coût de migration : les clients sont de plus en plus exigeants en termes de QoS. Leurs charges de travail varient dans le temps. De plus, les ressources de stockage sont généralement hétérogènes en termes de performances, prix, etc. Afin de répondre aux exigences des clients et aussi minimiser les coûts opérationnels, les stratégies d'optimisation procèdent souvent à la migration des objets entre les ressources de stockage disponibles. Cette migration engendre des coûts supplémentaires qui doivent être pris en considération par l'opération d'optimisation [150, 28, 144, 151, 140].
3. Minimisation de la latence réseau : la latence réseau est une métrique importante. Sa dégradation nuit à la satisfaction des clients. En effet, stocker les données près de leurs utilisateurs a été une motivation de plusieurs travaux existants sur le placement de données comme [150, 134, 35]. Cette métrique peut être optimisée en déployant les données sur plusieurs services de stockage plutôt qu'au sein d'un seul service afin de tirer parti des emplacements distribués de ces services de stockage.

— Contraintes : Généralement, un problème d'optimisation est défini en prenant en considération un ensemble de contraintes. Ces dernières définissent l'espace des solutions valides. Le problème de placement d'objets dans les environnements *Cloud* est caractérisé par des exigences fonctionnelles qui définissent ce qu'un système est censé faire. Pour le système de stockage, la limitation des capacités et performances de ressources de stockage et aussi les QoS exigées par les clients et définies dans les SLA sont des contraintes fonctionnelles qui doivent être prises en considération lors du placement de données.

— Réplication : La réplication consiste à créer plusieurs copies d'un objet de données et à les stocker dans des emplacements différents. Cette stratégie est utilisée dans les systèmes de stockage distribués afin d'améliorer leurs performance et fiabilité. Cependant, la réplication des données pose plusieurs défis. En effet, la réplication entraîne divers coûts pour le fournisseurs tels que le stockage et la consommation d'énergie pour la conservation des répliques. Aussi, la nature hétérogène d'un *Cloud* fédéré, la dynamicité de la charge de travail des clients et leur dispersion complique le maintien de la cohérence des répliques. De plus, un accès efficace aux données à moindre coût est une exigence critique dans le contexte du *Cloud*. Ainsi, le placement et la sélection des répliques doivent être effectués de manière judicieuse.

— Approches d'optimisation : Différentes approches d'optimisation ont été proposées ou réutilisées pour le placement des objets. Nous les catégorisons en deux classes : 1) les approches mono-objectif, et 2) les approches multi-objectifs. Chaque catégorie peut être, à son tour, classifiée en (i) exacte, (ii) heuristique, et (iii) méta-heuristique.

Le tableau 3.2 présente une synthèse des travaux de la littérature. Les symboles E, H et MH, utilisés dans la colonne **Approche d'optimisation**, correspondent respectivement à méthode exacte, heuristique et méta-heuristique.

3.2.1 Placement de données dans les *Clouds* centralisés

Dans cette partie nous présentons les travaux de l'état de l'art traitant le problème de placement de données dans un système de stockage hybride dans le cadre des *Clouds* centralisés. De grands efforts ont été faits sur le problème de placement des objets dans les environnements *Cloud* basés sur un système de stockage hybride. Dans ce qui suit, nous allons étudier brièvement ces travaux selon les objectifs considérés, les contraintes affrontées, ainsi que l'approche d'optimisation appliquée.

3.2.1.1 Objectifs du placement

En tant que propriétaire de l'infrastructure physique, le fournisseur de *Cloud* est responsable du placement des objets des clients. Réduire les coûts Opex de placement tout en maintenant des niveaux élevés de satisfaction des utilisateurs sont des facteurs importants pour que le fournisseur augmente ses revenus et gagne en réputation. Atteindre cet objectif nécessite des stratégies efficaces pour le placement des objets des clients sur l'infrastructure du fournisseur.

Pour le placement des données dans les *Clouds* centralisés, des approches de l'état de l'art ont considéré divers types de dispositifs (HDD, SSD) et de structures (horizontales, verticales) de stockage afin de trouver un compromis entre le coût et les performances au sein d'une infrastructure de stockage. Par structure horizontale, nous désignons la structure où les disques SSD

Catégories	Références	Objectifs				Contraintes		Réplication	Approche d'optimisation	
		Stockage	Migration	Latence	Autres	Performance E/S	SLA		SOO	MOO
***Clouds* centralisés**	[136]	✓				✓			E	
	[139]	✓							H	
	[159]		✓			✓				H, MH
	[137, 160]	✓				✓	✓		H	
	[161]		✓		✓			✓		MH
	[28]	✓	✓			✓	✓		H, MH	
	[162]				✓	✓		✓		H
	[163]				✓	✓		✓	E	
	[164]				✓				H	
	[165]	✓	✓			✓				MH
***Clouds* inter-connectés**	[166]		✓	✓			✓		H	
	[150]	✓	✓							H
	[29, 167]	✓			✓				H, MH	
	[77]	✓		✓				✓		
	[168, 169]	✓					✓	✓	H, MH	
	[170]	✓		✓	✓			✓	E	
	[171]	✓			✓			✓		MH
	[172]		✓		✓			✓		H
										MH

Tableau 3.2 – Classification des travaux relatifs au placement de données dans les environnement *Cloud* centralisés et inter-connectés.

sont mis au même niveau que les disques durs HDD tandis que dans une structure verticale les disques SSD sont utilisés comme cache pour les disques HDD.

Étant donné que la solution doit être orientée coût dans un contexte *Cloud*, plusieurs travaux ont essayé de minimiser le coût de placement de données. Certains travaux [139, 136, 137] ont visé la minimisation du coût de stockage. Ils l'ont calculé en fonction de l'occupation et de l'exécution de la charge de travail des clients. Nous constatons que dans [139, 136], ce sont les coûts d'achat (Capex) qui ont été optimisés tandis que dans [137], l'optimisation porte sur les coûts Opex de stockage. Dans [159], les auteurs ont proposé une solution pour le placement et la migration des données dans le but de minimiser la latence au niveau des dispositifs de stockage et le mouvement des données au moment de l'exécution. Bien que le coût de migration a été estimé par le nombre d'octets transférés entre les classes de stockage, le coût de stockage n'a pas été pris en compte. Les auteurs dans [28] ont présenté deux stratégies de placement d'objets basées sur les coûts pour minimiser le coût de placement de données dans un système de stockage hybride. Le coût de placement a été estimé en fonction de l'occupation, de l'exécution de la charge de travail et aussi en fonction de pénalité en cas de violation des termes du SLA.

Toutefois, un éventail de travaux sont orientés performance où le but était d'optimiser les performances sans prendre en considération le coût de stockage comme [162, 163, 164, 173, 174]. Ces approches ne peuvent pas être utilisées par les applications de *Cloud computing* comme ces dernières sont généralement orientées vers la réduction des coûts financiers. Néanmoins, leur utilisation peut se restreindre aux *Clouds* privés lorsque les performances emportent sur le coût.

3.2.1.2 Contraintes

Une solution d'un problème d'optimisation peut être contrainte par certaines propriétés qui doivent êtres respectées. Ces contraintes permettent en général de limiter l'espace de recherche (solutions réalisables). Comme tout problème d'optimisation, le problème de placement de données dans les environnements de *Cloud* doit répondre à certaines exigences. La capacité et les performances des ressources de stockage, qui sont limitées, doivent impérativement être prises en considération. Autrement dit, la somme des tailles de l'ensemble des objets stockés dans un périphérique de stockage ne doit pas dépasser la capacité de ce périphérique. De même, les performances des ressources de stockages en terme d'IOPS sont limitées, par conséquent, la charge de travail de l'ensemble d'objets stockés dans un périphérique ne doit pas dépasser ses performances. Ces contraintes ont été prises en considération dans la majorité des travaux comme dans [136, 28, 139, 137, 160]. Cependant, dans certains travaux [139, 161, 164] seule la contrainte de capacité de stockage limitée a été prise en charge sans considération de la limite des performances des périphériques de stockage.

En plus de ces contraintes, dans les environnements *Cloud* un CSP doit répondre aux QoS exigées par les clients et décrites dans le SLA. Pour le service de stockage, les contraintes liées

aux SLA de stockage ont été rarement considérées. Dans [28, 160], le SLA de stockage a été défini par deux paramètres : (i) *soft* SLA qui représente la QoS convenu du client et (ii) *hard* SLA définissant la limite de dégradation de service tolérée par le client. Dans ces travaux, le *hard* SLA doit être satisfaite. En revanche, si les IOPS effectivement délivrées au client sont inférieures au *soft* SLA, le fournisseur de service subit une pénalité.

3.2.1.3 Réplication

La réplication des données est utilisée pour assurer la disponibilité des données et la tolérance aux pannes. Le système de fichiers distribué Octopus [162] prend en charge le stockage des répliques de fichiers sur les supports de stockage qui sont attachés localement sur les nœuds du cluster, y compris la mémoire et les disques SSD. Long et al [175] ont proposé MORM une approche multi-objectif basée sur l'algorithme immunitaire [176] pour la gestion de la réplication. Le placement de répliques dans ce travail est basé sur un modèle mathématique avec plusieurs objectifs comprenant l'indisponibilité, le temps de service, la variation de la charge de travail, la consommation d'énergie. Les auteurs de [161] ont proposé un algorithme prenant en compte cinq facteurs qui pourraient influencer la décision de réplication. Il s'agit de l'indisponibilité des données, le trafic réseau, la performance du disque, l'équilibrage de charge, et la consommation de l'énergie. En se basant sur les accès aux fichiers, le système est capable d'ajuster le placement des répliques du fichier de manière dynamique.

3.2.1.4 Approches d'optimisation

Le problème de placement des données a été largement étudié dans l'environnement *Cloud* à base de système de stockage hybride. Des techniques d'optimisation mono et multi-objectifs ont été proposées pour atteindre le(s) objectif(s) souhaité(s).

Approches mono-objectif : Kim et al [136] ont développé un modèle afin de trouver la configuration HDD/SSD la plus économique pour des charges de travail données à l'aide de la programmation linéaire en nombres entiers mixtes. Le solveur CPLEX [177] a été utilisé pour résoudre leur système linéaire. Plusieurs travaux à base d'heuristique ont été proposés afin d'optimiser le placement de données dans un système de stockage hybride [139, 137, 160]. Dans [28], les auteurs ont proposé une heuristique pour le placement d'objets dans un système de stockage hybride afin de minimiser le coût de placement des données tout en satisfaisant les SLA des clients. De plus, dans ce travail, une méta-heuristique à base d'algorithme génétique a été utilisée pour résoudre le problème de placement. Dans [165], La méta-heuristique PSO (essaim de particules) a été utilisée pour fournir une fiabilité d'accès aux données dans le *Cloud*.

Approches multi-objectif : Dans la littérature, plusieurs travaux de recherche se sont intéressés au placement d'objets dans des environnements *Cloud* selon des critères multiples au moyen de techniques d'optimisation. Dans [162] des politiques de gestion de données automatisées basées sur une formulation de problème d'optimisation multi-objectifs ont été proposées. Un algorithme glouton a été proposé pour la résolution du problème de placement. Dans [175] une stratégie multi-objectif à base d'algorithme immunitaire artificiel a été utilisée pour optimiser le placement de répliques. Wu et al [159] ont proposé un schéma d'optimisation multi-objectifs basé sur un algorithme génétique pour l'optimisation du placement des données dans les systèmes de stockage hybrides.

3.2.2 Placement de données dans les *Clouds* inter-connectés

Dans les scénarios de *Clouds* inter-connectés, le placement d'objets est généralement axé sur la sélection des meilleures configurations de services pour distribuer et déployer les données et la charge de travail des clients. Différentes applications ont été envisagées telles que l'analyse de *big data*, l'optimisation des workflow, etc. Dans cette section, nous passons en revue les travaux de recherche sur les modèles de placement d'objets, avec un accent particulier sur ceux qui utilisent l'externalisation des données pour l'optimisation du placement.

3.2.2.1 Objectifs

Le problème de placement des données a été largement étudié dans l'environnement de *Clouds* inter-connectés. Les études de l'état de l'art ont envisagé différents objectifs. Plusieurs travaux ont optimisé le coût de stockage [150, 29, 167, 168, 170, 171, 169]. Dans ces travaux, le coût de stockage était lié à la capacité (Prix/Go) et [150] a considéré la charge de travail à un niveau élevé.

Certains travaux ont traité l'optimisation de la migration de données [166, 150, 172]. Généralement, les travaux traitant la migration ont tenu compte de la diversité des prix des services de stockage et de trafic du réseau offerts par les *Clouds* inter-connectés.

L'interconnexion des services de plusieurs *Clouds* est une solution importante pour les applications sensibles à la latence du réseau telles que le e-commerce, les jeux en ligne, le travail collaboratif assisté par ordinateur, etc. De plus, la latence du réseau a un rôle central pour la satisfaction des utilisateurs même s'il ne s'agit pas d'applications sensibles à la latence. En effet, une mauvaise expérience de latence réseau peut déprécier le service auprès de l'utilisateur l'incitant ainsi à chercher un fournisseur avec une bonne latence. Cette métrique a été optimisée dans plusieurs travaux comme [166, 77, 170].

3.2.2.2 Contraintes

La contrainte de capacité de stockage était considérée systématiquement par les travaux de l'état de l'art. Ce n'était pas le cas pour la contrainte de performance de stockage. Aucun travail parmi ceux étudiés (portant sur les *Clouds* inter-connectés) n'a considéré cette seconde contrainte. Nous pensons qu'il est très important de la considérer notamment car les services de stockage *Cloud* commercialisés sont caractérisés par des performances d'E/S limitées. Pour les SLA de stockage, quelques travaux les ont pris en compte mais ce n'est pas au niveau IOPS. Dans [150, 169] le SLA était exprimé en fonction de la latence réseau qui ne doit pas être dépassée pour les requêtes de lecture et écriture accédant aux objets stockés. Dans [168], le SLA était aussi exprimé en fonction de la latence. Celle-ci se décompose en latence réseau et latence d'accès aux supports de stockage.

3.2.2.3 Réplication

Les environnements informatiques distribués hétérogènes émergent pour développer des applications de haute disponibilité avec un accès rapide aux données. La réplication des données est une technique essentielle appliquée pour atteindre ces objectifs en stockant judicieusement plusieurs répliques à travers différentes régions. Dans les *Clouds* inter-connectés, un certain nombre de stratégies de réplication de données ont été proposées pour améliorer la QoS.

Li et al [172] ont construit une approche de placement de répliques basée sur la disponibilité des fichiers, la charge des nœuds et le coût de transmission. Spanstore [169] a optimisé le coût de placement de réplique en exploitant les différences de tarification entre les CSP tout en garantissant la latence requise pour l'application. Charm [178] a combiné les mécanismes de réplication et de *erasure coding* (une technique consistant à diviser un objet en plusieurs morceaux de taille égale contenant les données originales ainsi qu'en morceaux supplémentaires qui contiennent un codage de données) pour tirer parti de la disponibilité dans les environnements *multi-Clouds* tout en optimisant le coût de placement de données. Chandra et Weissman [168] ont introduit une approche pro-active pour éviter une réévaluation coûteuse du placement optimal des répliques.

3.2.2.4 Approches d'optimisation

Plusieurs contributions ont porté sur l'optimisation du placement de données dans les *Clouds* inter-connectées. Tout comme les *Clouds* centralisés, des approches mono et multi objectifs ont été utilisées pour l'optimisation du problème de placement.

Approches mono-objectif : Mansouri et al [150] ont proposé deux heuristiques mono-objectifs pour optimiser le coût de placement. Les algorithmes proposés font un compromis entre les coûts résidentiels et de migration et sélectionnent dynamiquement les classes de sto-

ckage. Dans [168], les auteurs ont proposé *Trips*, un système léger qui prend en compte à la fois les emplacements des centres de données et les classes de stockage pour déterminer le placement des données pour les systèmes de stockage géo-distribués. *Trips* modélise le problème de placement de données sous la forme d'un programme linéaire en nombres entiers mixtes (MILP) mono-objectif et utilise le solveur CPLEX pour résoudre le problème d'optimisation. SPANStore [169] est un autre système de stockage *multi-Cloud* qui cherche à minimiser le coût du stockage tout en minimisant la latence réseau. Dans ce travail, le problème de placement a également été modélisé sous forme d'un programme linéaire et le solveur CPLEX a été utilisé pour sa résolution.

Approches multi-objectifs : Dans [77], une optimisation multi-objectifs basée sur MOPSO (Multi-Objective Particle Swarm Optimization) a été proposée pour équilibrer la fiabilité des données, le coût de sauvegarde et le temps de récupération de données. Salem et al [170] ont appliqué une méta-heuristique de type colonie d'abeilles artificielles multi-objectifs pour trouver le meilleur chemin pendant le processus de sélection des répliques. Dans [29], une approche méta-heuristiques multi-objectifs basée sur l'algorithme NSGAII a été proposée pour le placement de données dans un environnement *multi-Cloud* en utilisant la technique *erasure coding*. L'objectif est d'éviter le phénomène de verrouillage envers les fournisseurs (*vedor lock-in*) tout en réalisant un compromis entre la disponibilité et le coût de placement.

3.2.3 Discussion

Les techniques de placement dans les *Clouds* centralisés sont généralement basées sur les caractéristiques et l'architecture du système de stockage, les modèles d'E/S de la charge de travail et les exigences SLA de stockage. Cependant, ces stratégies ne peuvent pas être utilisées directement dans les *Clouds* inter-connectés en raison de l'absence du facteur réseau et de la complexité des environnements fédérés. Compte tenu des techniques de placement dans les *Clouds* inter-connectés, le coût de stockage n'était lié qu'à la capacité (Prix/Go) et seuls [150, 168] ont considéré la charge de travail à un niveau élevé. De plus, dans ce type de *Clouds*, le SLA de stockage était rarement considéré. Dans [169, 150], le SLA des requêtes de lecture/écriture était évalué en terme de la latence réseau. Seul [168] a considéré la latence des classes de stockage. À notre connaissance, les SLA en terme d'IOPS des services de stockage n'ont pas été pris en compte dans les *Clouds* inter-connectés.

De plus, afin d'optimiser le placement des données, les coûts de stockage, de migration et de latence doivent être pris en compte. À notre connaissance, ces coûts n'ont pas été considérés comme objectifs distincts dans les approches multi-objectifs traitant le problème de placement des données. Nous estimons qu'il est important de les considérer comme des fonctions objectifs car ces coûts sont corrélés et dans certains cas contradictoires, des compromis sont donc

inévitables pour le CSP.

3.3 Conclusion

Ce chapitre a décrit les principaux efforts de recherche dans le domaine du placement de données dans un environnement *Cloud*. Nous nous concentrons principalement sur le coût de placement, l'objectif d'optimisation et l'architecture *Cloud* sous-jacente. Cela nous aide à mieux comprendre les problèmes de recherche que nous tentons de résoudre dans les chapitres restants.

L'orientation principale de cette thèse est la conception de modèles et d'algorithmes de placement d'objets efficaces pour la distribution des données et la charge de travail des clients à travers une fédération, tout en minimisant les coûts de placement des fournisseurs et augmentant la satisfaction des clients. Les chapitres suivants décrivent en détail nos contributions dans ce domaine de recherche.

CONTRIBUTIONS

Chapitre 4

Estimation du coût de placement de données dans un *Cloud* fédéré

La fédération de *Clouds* a émergé comme une solution potentielle pour surmonter les défis des *Clouds* traditionnels basés sur une architecture centralisée. Elle étend les offres *Cloud* à des services plus complexes impliquant des ressources distribuées sur plusieurs infrastructures. Ainsi, la fédération permet d'atténuer les problèmes de limitation des ressources et les fluctuations de la charge de travail en répondant continuellement aux besoins des clients.

Des algorithmes efficaces sont nécessaires pour aider les fournisseurs à définir des stratégies effectives de placement de données afin d'améliorer leur profit et satisfaire leurs clients. La modélisation des coûts est une étape très intéressante pour les CSP qui visent à offrir de meilleurs services à des prix compétitifs. En effet, les modèles de coûts sont fréquemment utilisés à des fins d'optimisation dans les environnements *Cloud* ce qui fait de leur conception un problème critique [179, 144, 145].

De nombreux travaux ont porté sur l'évaluation des coûts de placement des données dans les *Clouds* centralisés et inter-connectés [136, 145, 144, 18, 146, 19]. L'évaluation des coûts du système d'E/S de stockage a été profondément étudiée dans le cadre des *Clouds* centralisés et les modèles de coûts proposés ne sont pas applicables lorsqu'il s'agit de *Clouds* inter-connectés car le placement des données dans ce cas implique d'autres facteurs liés à l'environnement distribué comme le facteur réseau.

Dans ce chapitre, nous proposons notre première contribution, un modèle pour évaluer le coût du placement d'objets des clients d'un *Cloud* fédéré.

Nous avons suivi la méthodologie composée des trois étapes suivantes et illustrée dans la figure 4.1 pour la définition du modèle de coût :

1. **Étape1 :** Premièrement, énumérer tous les coûts fixes et variables qui peuvent être engendrés par une configuration d'un placement donné. Ces coûts peuvent être induits par les caractéristiques des clients, les détails du système de stockage et les propriétés de la fédération de *Clouds*.
2. **Étape2 :** Ensuite, élaborer les modèles à partir d'équations détaillant les différentes composants du modèle de coûts. Les différents paramètres liés à l'hétérogénéité des ressources

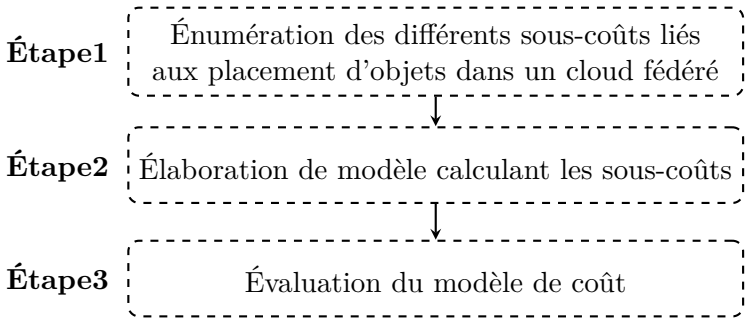


FIGURE 4.1 – Aperçu de la méthodologie décrivant les étapes de la modélisation du modèle de coût

locales de stockage et des services de stockage proposés par les *Clouds* partenaires, l'environnement variable de la fédération, les charges de travail variables des clients et à leur différent SLA doivent être pris en compte.

3. **Étape3 :** Enfin, pour chaque composant dans le modèle de coût, évaluer sa pertinence. Également, comparer le modèle de coût proposé avec des travaux de l'état de l'art et montrer ses différentes utilisations.

Le problème de placement de données dans un *Cloud* faisant partie d'une fédération de *Clouds* est très complexe. En effet, dans une telle infrastructure, il peut être plus rentable de stocker les données d'un client sur un HDD distant que sur un SSD local en cas de mobilité du client par exemple. Par conséquent, pour une estimation précise, le CSP doit prendre en compte toutes les caractéristiques liées à cet environnement hétérogène et variable comme la taille des objets, la charge de travail des clients, les classes de stockage internes et externes, le modèle de facturation des ressources et le degré de pénalité.

Le modèle de coût proposé est basé sur les types des clients. En effet, dans une fédération de *Clouds*, chaque CSP possède deux types de clients : (i) **clients internes :** qui sont ses propres clients, et (ii) **clients externes :** qui sont les clients des CSP partenaires dont il internalise les objets. Par conséquent, le coût global de placement de données se compose des éléments suivants :

— Coût de placement des objets des clients internes : ce coût est composé du coût de placement local, du coût d'externalisation, du coût de rapatriement des objets qui ont été déjà externalisés ainsi que du coût de pénalité.

— Coût de placement des clients externes : ce coût est constitué du coût d'internalisation et du coût de renvoi des objets qui ont déjà été internalisés.

Notre contribution présentant ce modèle de coût a été publiée dans [180].

Le chapitre en cours est organisé en 5 sections. La section 4.1 rappelle la problématique. La section 4.2 présente les hypothèses et quelques définitions. La section 4.3 détaille les différents

composants du modèle de coût. Nous présentons l'évaluation du modèle de coût proposé dans la section 4.4 avant de conclure le chapitre avec la section 4.5

Sommaire

4.1 Problématique

Rappelons que l'objectif principal du fournisseur de *Cloud* est de trouver une meilleure configuration de placement des objets qui minimise ses coûts tout en garantissant les SLA des clients. En effet, la minimisation des coûts permet aux fournisseurs d'offrir des services attractifs à des prix compétitifs.

La fédération de *Clouds* permet à différents CSP de travailler en collaboration en partageant une partie de leurs ressources afin d'améliorer leur productivité et offrir de meilleurs services aux clients sans investir dans la construction de nouveaux centres de données.

Toutefois, une fédération de *Clouds* est un environnement dynamique et hétérogène. En effet, un environnement fédéré est très complexe dû d'une part à l'hétérogénéité des ressources locales et des services de stockage externes proposés par les *Clouds* partenaires et d'autre part à l'environnement dynamique de la fédération et la charge de travail variable des clients et leur différent SLA. Cela implique que le coût de placement peut provenir de plusieurs parties. Il est donc important d'identifier l'ensemble de ces coûts. L'exactitude de cette estimation permet un meilleur placement des objets des clients. Dans cette perspective, nous avons adressé une première problématique consistant à savoir comment évaluer le coût de placement de données dans un *Cloud* membre dans une fédération.

4.2 Hypothèses et définitions

Dans cette section, nous introduisons quelques définitions et les hypothèses nécessaires dans la définition du modèle de coût.

4.2.1 Définitions

Pour plus de précision, nous introduisons les définitions suivantes des concepts utilisés dans notre modélisation :

Externalisation [18, 88, 19, 20] : la capacité d'un CSP à envoyer des objets de ses clients internes à d'autres CSP membres de la fédération.

Internalisation [19, 154, 88, 20] : le processus inverse de l'externalisation. Les CSP mettent à disposition une partie de leurs ressources inutilisées pour répondre aux demandes des autres membres de la fédération.

Géo-migration [150, 147] : le processus de déplacement d'objets vers d'autres *Clouds*.

Migration interne [145, 144] : le mouvement de certains objets entre différentes classes de stockage au sein d'une même infrastructure. Un CSP déplace périodiquement certains objets entre les différentes classes de stockage locales.

Rapatriement *:* Cette opération consiste à ramener des objets précédemment externalisés vers l'infrastructure locale.

4.2.2 Hypothèses

Notre modèle de de coût se base sur les hypothèses suivantes :

— Nous considérons une fédération F= $\{CSP^d, d \in [1..D]\}$ composée de plusieurs CSP, coopérant de manière pair à pair. Cette architecture a été utilisée dans plusieurs travaux comme [19, 88].

— Chaque CSP^d possède un seul centre de données comportant différents types de classes de stockage $SC = \{sc_j, j \in [1..J]\}$. Chaque classe de stockage sc_j est caractérisée par : sa capacité c_{sc_j}, ses performances en terme d'IOPS $iops_{op,sc_j}$, son coût d'achat p_{sc_j}, son endurance wo_{sc_j}. Nous supposons que les CSP consacrent une partie de leurs ressources à leurs partenaires fédérés.

— Nous supposons qu'un CSP achète une bande passante bw auprès d'un fournisseur Internet avec un coût d'achat p_{bw}.

— Les CSP peuvent externaliser les objets de leurs clients auprès d'autres membres de la fédération, par exemple, lorsqu'ils ne peuvent pas satisfaire les exigences de qualité de service des clients. Par conséquent, chaque CSP^d possède deux types de clients :

1. **Clients internes** : clients ayant contractualisé avec le $CSP^d, U_{int} = \{u_k, k \in [1, K]\}$.
2. **Clients externes** : clients des CSP partenaires ($\{CSP^{d'} \in F, d \neq d'\}$). Les objets de ces clients sont externalisés dans $CSP^d, U_{ext} = \{u^{d'}_{k'}, k' \in [1, K']\}, CSP^{d'} \in F - \{CSP^d\}$. On dit que les objets des clients de $CSP^{d'}$ sont internalisés par CSP^d.

— Nous supposons que chaque client :

1. Possède un ensemble d'objets $\{o_{i,k}, i \in [1..N]\}$. Chaque objet a une taille notée $s_{o_{i,k}}$.
2. Génère une charge de travail wl_k représentant les opérations d'E/S générées par l'ensemble des requêtes émises par le client. Selon le type d'accès (séquentiel, aléatoire) et le type d'opération (lecture, écriture), on distingue quatre motifs d'E/S comme dans [145] : lecture séquentielle (*sequential read* : SR), écriture séquentielle (*sequential write* : SW) , lecture aléatoire (*random read* : RR) et enfin, écriture aléatoire (*random write* : RW).
3. Est caractérisé par une QoS demandée (SLA) en termes d'IOPS ($iops_{sla,k}$.) et de latence $ltc_{sla,k}$.
4. Possède une fonction de pénalité pn_k composée de deux parties, une liée aux performances d'E/S de stockage $pn_{iops,k}$ et l'autre à la latence du réseau $pn_{ltc,k}$.

— Périodiquement, l'administrateur du *Cloud* prend des décisions sur le placement des objets. On note par T la période de temps pendant laquelle la surveillance est exécutée pour extraire les motifs d'E/S des objets afin d'évaluer le coût global de placement comme dans [145, 144].

— Dans notre étude, nous supposons qu'un CSP donné facture à ses *Clouds* partenaires des opérations d'internalisation. Des prix réduits sont utilisés afin de favoriser la coopération au sein de la fédération. Tout comme dans [18, 19], les CSP mettent à jour dynamiquement le prix de ses ressources de stockage en fonction de la quantité de ressources inutilisées. Soit $Cap_{max_{rsc}}$ et $Cap_{idl_{rsc}}$ les capacités totales et inactives pour une ressource donnée rsc du fournisseur CSP^d. rsc peut être l'occupation du stockage (occ) ou les performances en IOPS ($iops$) ou une combinaison des deux. Si p_{rsc} est le prix payé par les clients internes pour la ressource rsc, son prix d'internalisation F^d_{res} est obtenu à partir de l'expression donnée dans l'équation (4.1) utilisée dans [18]. Pour chaque période T, nous supposons que les CSP utilisent l'équation (4.1) pour ajuster leurs prix d'internalisation.

$$F^d_{rsc} = \frac{Cap_{max_{rsc}} - Cap_{idl_{rsc}}}{Cap_{max_{rsc}}} * (p_{rsc}) \tag{4.1}$$

— Sur la base des exigences du SLA des clients et de la quantité de ressources de stockage disponibles, les objets sont servis localement (placés et migrés entre les différentes classes de stockage internes), rapatriés, géo-migrés, géo-répliqués vers des *Clouds* partenaires ou renvoyés vers leurs *Clouds* originaux.

4.3 Modèle de coût de stockage dans une fédération de *Clouds*

Un CSP pense principalement aux gains que son centre de données pourrait lui apporter. Il lui faut donc des moyens efficaces pour réduire les coûts tout en satisfaisant ses clients ceci signifie qu'avant toute chose, un *Cloud* doit estimer le coût de placement de données de ses clients afin de pouvoir l'optimiser. Le reste de cette section décrit le modèle de coût proposé pour l'évaluation du placement de données dans un *Cloud* fédéré.

4.3.1 Aperçu

Nous définissons ci-après un modèle permettant d'estimer le coût d'une configuration de placement donnée. Nous supposons que cette dernière est obtenue en déplaçant un ensemble d'objets à partir d'une configuration initiale. La figure 4.2 montre les principaux composants de notre modèle ainsi que les numéros des équations détaillant le coût correspondant.

Nous avons catégorisé les coûts liés au placement d'objets en deux grandes parties. En effet, le coût total de placement des objets $Cost_{plc,T}$ est la somme du coût de placement des objets des

clients internes ($Cost_{plc_{int},T}$) et des clients externes ($Cost_{plc_{ext},T}$) comme indiqué dans l'équation (4.2).

$$Cost_{plc,T} = Cost_{plc_{int},T} + Cost_{plc_{ext},T} \tag{4.2}$$

Le coût de placement des objets des clients internes est composé du coût de placement local, du coût d'externalisation, du coût de rapatriement des objets qui ont été déjà externalisés ainsi que du coût de pénalité. Le coût de placement des objets des clients externes est constitué du coût d'internalisation et du coût de renvoi des objets qui ont été déjà internalisés. Il faut noter que chaque fois qu'un coût est discuté, il est exprimé par un coût monétaire.

Notez que les coûts non récurrents qui ne dépendent pas du placement des objets, tels que les coûts de maintenance, les coûts en ressources humaines et les coûts de refroidissement n'ont pas été pris en compte dans notre modèle. Nous n'avons pas pris également en compte les coûts liés à la sécurité des données.

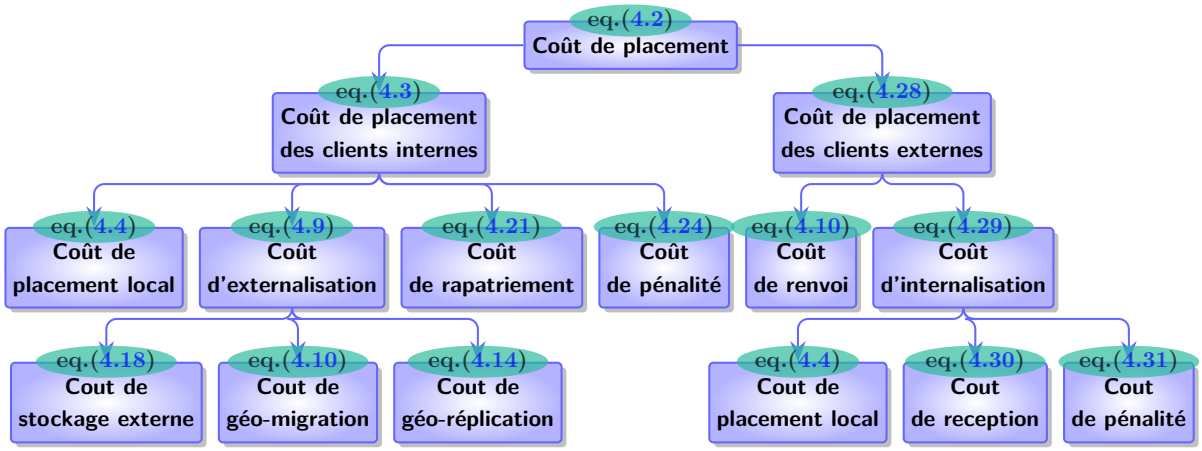


FIGURE 4.2 – Coût de placement global

4.3.2 Coût de placement des objets des clients internes

Les objets des clients internes peuvent être placés localement ou chez les autres *Clouds*. Ainsi, le coût de placement des objets des clients internes comprend le coût de placement local $Cost_{lcl_{int},T}$, le coût d'externalisation $Cost_{out_{src},T}$, le coût du rapatriement $Cost_{bck_{mgr},T}$ et le coût de la pénalité $Cost_{pnt_{int},T}$ comme indiqué dans l'équation (4.3).

Le coût de placement local est le coût de stockage des objets des clients internes dans l'infrastructure locale. Le coût d'externalisation est lié au placement des objets des clients internes dans les CSP partenaires. Le coût du rapatriement représente le coût du retour des objets précédemment externalisés dans les *Clouds* partenaires. Enfin, le coût de pénalité représente le coût

monétaire supplémentaire découlant de la violation des SLA des clients.

$$Cost_{plc_{int},T} = Cost_{lcl_{int},T} + Cost_{out_{src},T} + Cost_{bck_{mgr},T} + Cost_{pnt_{int},T} \tag{4.3}$$

4.3.2.1 Coût de placement local

Ce coût est obtenu à partir du coût de stockage des objets des clients internes $Cost_{stg_{int},T}$ dans les périphériques de stockage internes et du coût de migration interne (migration entre ces périphériques de stockage) $Cost_{mgr_{int},T}$. L'équation (4.4) montre ce coût.

$$Cost_{lcl_{int},T} = Cost_{stg_{int},T} + Cost_{mgr_{int},T} \tag{4.4}$$

Coût de stockage : Le coût de stockage $Cost_{stg_{int},T}$ inclut l'ensemble des dépenses liées à l'utilisation des périphériques de stockage. Ce coût est composé du coût d'occupation $Cost_{occ,T}$, du coût énergétique $Cost_{erg,T}$ et du coût d'endurance $Cost_{edr,T}$ dû à l'usure causée par l'exécution de la charge de travail d'E/S des clients comme c'est indiqué dans l'équation (4.5).

$$Cost_{stg_{int},T} = max[Cost_{occ,T}, Cost_{edr,T}] + Cost_{erg,T} \tag{4.5}$$

Le coût d'occupation est le coût amorti des périphériques de stockage sur la période T. Il est calculé en fonction de la répartition du coût d'achat du dispositif de stockage sur une période d'amortissement, représentée généralement par la période de garantie comme c'est indiqué dans l'équation (4.6).

$$Cost_{occ,T} = \sum_{sc_j \in SC} \frac{\sum_{u_k \in U_{int}, o_{i,k} \in sc_j} S_{o_{i,k}}}{c}_{sc_j} * Cost_{amz,T}(sc_j) \tag{4.6}$$

Où $Cost_{amz,T}(sc_j)$ désigne le coût d'amortissement du périphérique sc_j pour la période T. Nous le calculons en répartissant le coût d'achat du périphérique de stockage sc_j sur la période d'amortissement.

Le coût énergétique est lié à l'énergie consommée par le système de stockage pour exécuter la charge de travail d'E/S multipliée par le prix unitaire de l'énergie E_{up} que nous considérons constant comme le montre l'équation (4.7).

$$Cost_{erg,T} = \sum_{sc_j \in SC} (P_{atv,sc_j} * t_{atv,sc_j} + P_{idl,sc_j} * t_{idl,sc_j}) * E_{up} \tag{4.7}$$

Où P_{atv,sc_j}, P_{idl,sc_j} sont les puissances électriques de la consommation d'énergie en fonction des états d'alimentation du périphérique de stockage (actif et inactif) tandis que t_{atv,sc_j} et t_{idl,sc_j}

représentent respectivement le temps nécessaire pour gérer la charge de travail d'E/S et le temps d'inactivité des périphériques de stockage.

Enfin, le coût d'endurance est causé par l'exécution de la charge d'E/S des clients. En effet, la charge d'E/S impacte la durée de vie des périphériques de stockage [181]. Pour le SSD, ce coût est lié à la quantité de données écrites tandis que pour le HDD, il dépend de la quantité de données lues et écrites. L'équation suivante calcule le coût d'endurance.

$$Cost_{edr,T} = \sum_{sc_j \in SC} Cost_{edr,T}(sc_j) \tag{4.8}$$

Pour plus de détail sur les coûts $Cost_{occ,T}$, $Cost_{erg,T}$ et $Cost_{edr,T}$, voir [145, 144]

Coût de migration interne : pour atteindre une configuration d'un placement donnée, des objets doivent être déplacés à partir d'un placement initial vers le placement souhaité. La migration interne consiste à lire un ensemble d'objets du périphérique de stockage interne (périphérique source) et à les écrire sur un autre périphérique de stockage interne (périphérique destination). Cette opération engendre l'exécution d'une charge de travail d'E/S supplémentaire. Nous avons calculé le coût de la migration interne en fonction des coûts d'énergie (équation (4.7)) et d'endurance liés (équation (4.8)) à l'exécution de cette charge comme dans [145, 144].

4.3.2.2 Coût d'externalisation

Un CSP peut externaliser une partie des objets de ses clients internes et les charges de travail correspondantes à d'autres membres de la fédération, en empruntant des ressources auprès de ces Clouds externes pour obtenir des ressources de stockage supplémentaires. L'externalisation peut être réalisée en migrant les objets sans garder une copie dans l'infrastructure locale ou en répliquant les objets dans plusieurs emplacements. Le coût de cette opération $Cost_{out_{src},T}$ est composé du coût de géo-migration $Cost_{geo_{mgr},T}$, du coût de réplication $Cost_{geo_{rpl},T}$, ainsi que du coût de stockage $Cost_{ext_{plc},T}$ dans les CSP partenaires générés par ces deux géo-opérations. L'équation (4.9) de ce coût est établie comme suit :

$$Cost_{out_{src},T} = Cost_{geo_{mgr},T} + Cost_{geo_{rpl},T} + Cost_{ext_{plc},T} \tag{4.9}$$

Coût de géo-migration : la géo-migration des objets entraîne des coûts de lectures locales $Cost_{rd,T}$ à partir du périphérique source ainsi que du coût d'utilisation de la bande passante Internet $Cost_{bw,T}$. Ce coût est calculé par l'équation (4.10) comme ci-dessous :

$$Cost_{geo_{mgr},T} = Cost_{rd,T} + Cost_{bw,T} \tag{4.10}$$

Le coût de l'opération de lecture locale engendre des coûts d'énergie $Cost_{rd_{erg},T}$ consommée par le périphérique source et des coûts d'usure $Cost_{rd_{edr},T}$ subie par ce périphérique comme le montre l'équation (4.11).

$$Cost_{rd,T} = Cost_{rd_{erg},T} + Cost_{rd_{edr},T} \tag{4.11}$$

Le coût de la bande passante Internet est lié à la bande passante consommée par la géo-migration des objets vers des *Clouds* externes.

Nous calculons ce coût en multipliant la taille de tous les objets à migrer par la bande passante amortie sur une unité de temps $Cost_{bw_{amz},1}(bw_{int})$ (voir l'équation (4.12)).

$$Cost_{bw,T} = \frac{\sum_{o_i \in O_{g_{mgr}}} (S_{o_i}) * Cost_{bw_{amz},1}(bw_{int})}{bw_{int}} \tag{4.12}$$

Nous calculons le coût de bande passante amorti sur une unité de temps en répartissant le coût d'achat de bw_{int} sur la période d'abonnement T_{sp}.

$$Cost_{bw_{amz},1}(bw_{int}) = \frac{p_{bw} * \frac{bw_{int}}{bw}}{T_{sp}} \tag{4.13}$$

Coût de géo-réplication La réplication de données consiste à créer plusieurs copies d'une donnée. Cette technique est très utilisée et efficace pour augmenter la disponibilité de données, réduire les coûts d'accès aux données et assurer une meilleure tolérance aux pannes. La géo-réplication consiste à répliquer les données sur plusieurs sites (CSP) répartis géographiquement. Cette opération entraîne des coûts de stockage et de réseau supplémentaires. La géo-réplication consiste à ajouter, supprimer et synchroniser des répliques. Ainsi, son coût global est la somme des coûts d'exploitation susmentionnés comme indiqué dans l'équation (4.14). Le coût de suppression de réplique est supposée être nul.

$$Cost_{geo_{rpl},T} = Cost_{add_{rpl},T} + Cost_{syc_{rpl},T} \tag{4.14}$$

Le coût d'ajout de répliques est le même que le coût de géo-migration sans compter le coût de suppression de la copie d'origine et en considérant l'ensemble des objets à répliquer $\{O^{d'}_{g_{rpl}}, d' \neq d\}$.

$$Cost_{add_{rep},T} = \sum_{CSP^{d'} \in F - \{CSP^d\}} Cost_{geo_{mgr},T}(O^{d'}_{g_{rpl}}) \tag{4.15}$$

Concernant la synchronisation des répliques, le CSP^d est le responsable du déclenchement de cette opération, c'est-à-dire, celui qui transmet la mise à jour aux autres copies, ou il reçoit la demande de mise à jour de la copie qu'il possède. $O^{d'}_{s_{rpl}}$ est la liste des objets répliqués dans

$CSP^{d'}$.

Lorsque CSP^d transmet la synchronisation, son coût est la somme des opérations de lecture induites par la synchronisation et des coûts de bande passante Internet consommée, voir l'équation (4.16).

$$Cost_{syn_{rep},T} = \sum_{CSP^{d'} \in F - \{CSP^d\}} Cost_{geo_{mgr},T}(O^{d'}_{s_{rpl}}) \tag{4.16}$$

Tandis que lorsque CSP^d reçoit la synchronisation, ce coût comprend les opérations d'écriture, l'utilisation de la bande passante Internet locale et les coûts du réseau sortant (c'est-à-dire le coût du rapatriement, voir l'équation. (4.21)). L'équation (4.17) montre le coût de synchronisation.

$$Cost_{syn_{rep},T} = \sum_{CSP^{d'} \in F - \{CSP^d\}} Cost_{bck_{mgr},T}(O^{d'}_{s_{rpl}}) \tag{4.17}$$

Coût de stockage externe : Une fois les objets migrés vers d'autres *Clouds*, le CSP concerné par l'externalisation doit payer les *Clouds* partenaires pour l'hébergement des objets de ses clients internes. Par conséquent, le coût de stockage externe comprend le coût d'occupation externe $Cost_{ext_{occ},T}$, le coût d'exécution de la charge de travail externe $Cost_{ext_{wld},T}$ et le coût de pénalité externe $Cost_{ext_{pnl},T}$. Le coût de pénalité ici est le montant payé par les *Clouds* partenaires en cas de violation des SLA pour les objets externalisés. Ce coût est indiqué dans l'équation (4.18).

$$Cost_{ext_{plc},T} = Cost_{ext_{occ},T} + Cost_{ext_{wld},T} - Cost_{ext_{pnl},T} \tag{4.18}$$

Le coût d'occupation externe représente le coût de stockage des objets des clients internes migrés ou répliqués dans les *Clouds* partenaires. Nous calculons ce coût en utilisant le coût d'occupation fédéré $F^{d'}_{occ}$ de $CSP^{d'}$ comme indiqué dans l'équation (4.19).

$$Cost_{ext_{occ},T} = \sum_{d'|CSP^{d'} \in F - \{CSP^d\}} \Big(\sum_{o_i \in CSP^d} S(o_i) \Big) * F^{d'}_{occ} \tag{4.19}$$

Le coût d'exécution de la charge de travail représente la quantité d'IOPS qui devrait être consommée par les clients internes. Elle est calculée comme indiqué dans l'équation 4.20 où $F^{d'}_{iops}$ est le coût d'une IOPS dans le *Cloud* $CSP^{d'}$.

$$Cost_{ext_{wld},T} = \sum_{d'|CSP^{d'} \in F - \{CSP^d\}} \Big(\sum_{o_i \in CSP^d} IOPS(o_i) \Big) * F^{d'}_{iops} \tag{4.20}$$

$F^{d'}_{occ}$ et $F^{d'}_{iops}$ sont calculés par l'équation (4.1).

Le coût de la pénalité est un pourcentage du total des frais payés par CSP^d au *Cloud* partenaire, comme dans Amazon [145, 182].

4.3.2.3 Coût de rapatriement

Le coût de rapatriement $Cost_{bck_{mgr},T}$ correspond au coût du rapatriement des objets précédemment externalisés ($O_{b_{mgr}}$) dans l'infrastructure locale. Il est composé du coût du transfert de données (coût du réseau sortant) depuis les *Clouds* partenaires $Cost_{ntw_{out},T}$, le coût de la bande passante Internet locale $Cost_{bw,T}$ correspondant à la bande passante consommée localement pour recevoir les objets précédemment externalisés et le coût d'écriture $Cost_{wr,T}$ des objets reçus sur le système de stockage local. Le coût de rapatriement est donné comme suit :

$$Cost_{bck_{mgr},T} = Cost_{wr,T} + Cost_{ntw_{out},T} + Cost_{bw,T} \tag{4.21}$$

Le coût d'écriture local est le coût de placement des objets précédemment externalisés dans le système de stockage local. Ce coût engendre un coût $Cost_{wr_{erg},T}$ de l'énergie consommée par les dispositifs de stockage et un coût de l'usure de ces dispositifs $Cost_{wr_{edr},T}$ occasionnée par les opérations d'écriture. Certains de ces objets sont placés sur des disques HDD et d'autres sur des disques SSD. Ce coût est établi comme suit :

$$Cost_{wr,T} = Cost_{wr_{erg},T} + Cost_{wr_{edr},T} \tag{4.22}$$

Le coût du réseau sortant est facturé par les *Clouds* partenaires lorsque CSP^d ramène les objets de ses clients internes vers son infrastructure locale. Nous calculons ce coût en multipliant la taille des objets à ramener par le coût du réseau sortant $Cost_{out}(CSP^{d'})$ du CSP partenaire comme indiqué dans l'équation (4.23).

$$Cost_{ntw_{out},T} = \sum_{CSP^{d'} \in F - \{CSP^d\}} (Cost_{out}(CSP^{d'}) * (\sum_{\substack{o_i \in O_{b_{mgr}} \wedge \\ o_i \in CSP^{d'}}} S_{o_i})) \tag{4.23}$$

Le coût de la bande passante Internet locale est calculé avec l'équation (4.12).

4.3.2.4 Coût de la pénalité

Le placement des objets doit être effectué sans affecter la qualité de service demandée par les clients. En cas de violation des termes du SLA par le CSP, des sanctions peuvent être appliquées sous forme d'une pénalité financière subie par le CSP. Cette pénalité est un taux qui réduit la facture du client. Elle dépend du prix unitaire du service. Trois types de pénalités existent [183] : (i) pénalité fixe où un rabais fixe est appliqué en cas de violation des clauses du SLA, (ii) pénalité en fonction du retard où le montant de la pénalité est lié au temps de violation et enfin (iii)

pénalité proportionnelle au retard et au degré de violation du SLA.

Le coût global de pénalité est la somme de toutes les pénalités des clients internes comme précisé dans l'équation suivante :

$$Cost_{pnt_{int},T} = \sum_{u_k \in U_{int}} (Cost_{pnt,T}(u_k)) \tag{4.24}$$

Le coût de la pénalité des clients internes est composé d'une pénalité liée aux performance de stockage en terme d'IOPS $Cost_{pnt_{iops},T}(u_k)$ et d'une autre liée à la latence réseau $Cost_{pnt_{ltc},T}(u_k)$ comme le montre l'équation (4.25).

$$Cost_{pnt,T}(u_k) = Cost_{pnt_{iops},T}(u_k) + Cost_{pnt_{ltc},T}(u_k) \tag{4.25}$$

Le coût de pénalité IOPS est calculé comme dans [145], il est proportionnel au rapport entre l'IOPS offert $iops_{offered}(u_k)$ et celui demandé ($iops_{sla,k}$). L'IOPS demandé est défini par les clients tandis que nous calculons l'IOPS offert à partir du temps nécessaire pour gérer la charge d'E/S du client u_k et le nombre total de demandes d'E/S émises vers leurs objets. Ce coût est calculé dans l'équation suivante (équation (4.26)) comme suit :

$$Cost_{pnt_{iops},T}(u_k) = pnt_{iops,k}(\frac{iops_{offered}(u_k)}{iops_{sla,k}}) \tag{4.26}$$

Dans notre modèle, nous supposons que les clients paient des frais supplémentaires pour avoir une latence réseau réduite. Ce coût de pénalité est proportionnel au degré de violation et au nombre de requêtes nb_{rqt}, comme indiqué dans le tableau 4.1. Une pénalité est appliquée au CSP selon la latence offerte $ltc_{offered}(u_k)$ au client u_k et celle définie dans le SLA du client $ltc_{sla,k}$. Nous calculons le coût de la pénalité de latence comme suit :

$$Cost_{ltc_{iops},T}(u_k) = \sum_{i=1} nb_{rqt}(pn_{ltc,k}(ltc_{offered}(u_k), ltc_{sla,k})) \tag{4.27}$$

$ltc_{offered}$	Penalité (%)
$[0..B_0]$	0
$[B_i..B_{i+1}]$	x_i
$> B_n$	x_n

Tableau 4.1 – Pénalité de latence par requête

Dans le tableau 4.1, les paramètres B_0, ... , B_i, ... , B_n représentent les bornes des intervalles de la latence du réseau offerte, et 0, ... , x_i , ... , x_n sont les pénalités fixes correspondantes.

4.3.3 Coût de placement des objets des clients externes

Le coût de placement des objets des clients externes $Cost_{plc_{ext},T}$ est la somme du coût d'internalisation $Cost_{in_{src},T}$ des objets des clients externes et le coût de renvoi $Cost_{re_{mgr},T}$ qui est le coût généré lorsque certains objets des clients externes sont géo-migrés vers d'autres *Clouds* ou repris par leurs *Clouds* d'origine. Ce coût est exprimé dans l'équation (4.28) comme suit :

$$Cost_{plc_{ext},T} = Cost_{in_{src},T} + Cost_{re_{mgr},T} \tag{4.28}$$

4.3.3.1 Coût d'internalisation

Le coût d'internalisation est composé du coût de réception des objets des clients externes $Cost_{rcv_{ext},T}$, du coût de placement local $Cost_{lcl_{ext},T}$ et du coût de pénalité $Cost_{pnt_{ext},T}$ comme illustré dans l'équation (4.29).

$$Cost_{in_{src},T} = Cost_{rcv_{ext},T} + Cost_{lcl_{ext},T} + Cost_{pnt_{ext},T} \tag{4.29}$$

Le coût de réception des objets des clients externes est la somme du coût d'écriture de ces objets, calculé à partir de l'équation (4.22) et du coût de la bande passante Internet consommée localement, calculé avec l'équation (4.12).

$$Cost_{rcv_{ext},T} = Cost_{wr,T} + Cost_{bw,T} \tag{4.30}$$

Le coût de placement local est le coût de stockage et de migration des objets des clients externes dans le système de stockage interne. Ce coût est calculé à partir de l'équation (4.4).

Le coût de pénalité est la somme des coûts de pénalité des clients externes.

$$Cost_{pnt_{ext},T} = \sum_{u_{k'}^{d'} \in U_{ext}} (Cost_{pnt,T}(u_{k'}^{d'})) \tag{4.31}$$

Dans la fédération, les prix des ressources sont définis de manière dynamique (voir l'équation (4.1)). Par conséquent, le coût de la pénalité est défini en conséquence et est lié aux pourcentage des ressources de stockage inutilisées β qui est calculé comme suit :

$$\beta = \frac{Cap_{max_{rsc}} - Cap_{idl_{rsc}}}{Cap_{max_{rsc}}} \tag{4.32}$$

Avec la ressource de stockage rsc étant une combinaison de l'espace de stockage et des performances demandées. Le coût de pénalité d'un client externe $u_{k'}^{d'}$ est donc donné dans l'équation (4.33) :

$$Cost_{pnt,T}(u_{k'}^{d'}) = \beta * Cost_{pnt_{iops},T}(u_{k'}^{d'}) \tag{4.33}$$

4.3.3.2 Coût de renvoi

Ce coût est déterminé par la décision des *Clouds* originaux de géo-migrer les objets de leurs clients externes vers leur infrastructure ou vers d'autres *Clouds*. Cela entraîne des opérations de lecture locales et des coûts de bande passante Internet, comme décrit dans l'équation (4.10).

Après avoir défini notre modèle de coût, il est indispensable de l'évaluer pour juger sa pertinence. Nous décrivons dans le reste du chapitre les résultats de notre évaluation

4.4 Évaluation

Nous présentons dans cette section les expérimentations que nous avons mené afin d'évaluer le modèle de coût proposé. Notre objectif comporte deux volets : (i) valider la pertinence des sous-coûts utilisés dans notre modèle de coûts et les comparer aux modèles de l'état de l'art, (ii) montrer la flexibilité du modèle de coûts à travers l'investigation de l'impact de différents paramètres sur le coût de placement.

Avant d'évaluer le modèle de coût proposé, nous allons présenter la méthodologie d'évaluation et les paramètres expérimentaux utilisés.

4.4.1 Méthodologie d'évaluation

Afin d'évaluer le modèle de coût proposé, nous l'appliquons pour estimer le coût de placement de plusieurs bases de données construites en utilisant les benchemarks TPC-H et TPC-C. Nous faisons varier la taille, la charge de travail des bases de données et la latence du réseau des requêtes pour simuler un environnement réel.

Dans un premier temps, nous validons la pertinence des sous-coûts en variant aléatoirement la latence du réseau ([200ms, 700ms]) entre chaque pair de *Clouds* . Nous évaluons les différents sous-coûts pour chaque CSP, puis nous calculons le coût moyen de placement des (1) bases de données internalisées et des (2) bases de données externalisées et comparons les coûts résultants avec ceux de certaines études de l'état de l'art [147, 148, 150, 18, 19].

Une fois la pertinence validée, nous montrons comment le modèle de coût proposé permet d'évaluer facilement le modèle de tarification des ressources non utilisées en fonction de la taille des bases de données et la charge de travail exécutée. Pour ce faire, nous comparons, d'une part, les coûts des objets internalisés et le prix facturé aux *Clouds* partenaires, et d'autre part, les coûts des objets externalisés lors de l'achat de ressources auprès de *Clouds* partenaires par rapport à leur achat depuis un *Cloud* externe à la fédération.

Nous montrons l'impact du degré de violation de la latence du réseau sur le coût moyen de placement et quand est ce que c'est intéressant l'externalisation des objets des clients. Pour cela, nous comparons le coût moyen de placement des bases de données des clients mobiles en faisant varier la latence du réseau entre les *Clouds* selon trois scénarios : (1) la géo-migration à l'aide

d'une fédération, (2) la géo-migration à l'aide d'une plate-forme de *Cloud computing* distribué et (3) *Cloud* centralisé sans géo-migration.

4.4.2 Paramètres expérimentaux

Nous avons utilisé le simulateur CloudSim [184]. Le scénario simulé est composé d'une fédération de 9 CSP répartis géographiquement. Chaque CSP est composé d'un centre de données exécutant 1000 VM comme dans [18]. Certaines de ces VM (fixées à 20% dans nos expérimentations) sont dédiées à la fédération. Chaque CSP dispose d'une bande passante Internet de 1 Gbps achetée auprès d'un fournisseur d'accès Internet (le prix a été fixé à 1500$ par mois). Les spécifications du système de stockage sont fournies dans le tableau 4.2.

Characteristiques	HDD	SSD
Prix ($)	230	200
Taille	1 TB	128 GB
Performance	Seek time :8.5 ms	rr :10000 IOPS, rw :40000 IOPS
	lecture/écriture : 9.5 ms	sr :540 MB/s, sw : 520 MB/s

Tableau 4.2 – Spécifications des périphériques de stockage

Chaque CSP gère un ensemble de bases de données (DB) générées à partir des benchmarks TPC-H et TPC-C avec différentes tailles et charges de travail. Nous avons utilisé la même configuration définie dans [145]. Pour plus de détails sur les différentes bases de données (voir le tableau 4.3). Le montant de la pénalité de stockage est fixé à 30 % du total des frais payés par le client comme dans [145]. Le prix du stockage est fixé selon le modèle de prix Amazon gp2 (0,1$/Go/mois) tandis que le coût de l'énergie est fixé à 0,1$ par kWh comme dans [136].

Certains clients sont mobiles et demandent une bonne latence en payant des frais supplémentaires. Leur CSP d'origine migre généralement leurs objets vers le CSP le plus proche qui respecte la contrainte de latence, faute de quoi le CSP d'origine risque de subir une pénalité. Les clients mobiles sont supposés avoir une mobilité d'une semaine. Le prix de la latence est donné dans le tableau 4.4 comme dans [23] et la pénalité associée dans le tableau 4.5. L'évaluation est

Base de donnée	Bench.	Reqs. nbr (op/h)	rr(op/h)	rw(op/h)	sr(op/h)	sw(op/h)
DB1(32GB/HDD)	TPC-C	432000	136800	46800	108000	32400
DB2(60GB/SSD)		28800000	601200	104400	5436000	147600
DB3(147GB/HDD)	TPC-H	331200	10800	216000	3600	18000
DB4(34GB/HDD)		355200	32400	216000	7200	10800
DB5(381GB/HDD)		15360	1080	8280	360	1080

Tableau 4.3 – Spécifications des bases de données

Latence (ms)	Prix par requete ($)
< 200	0.0000001
< 400	0.00000005
< 600	0.00000002
> 600	0

Tableau 4.4 – Prix de latence/req.

Latence (ms)	Penalité ($)
< 200	0
< 400	0.00000005
< 600	0.00000008
> 600	0.0000001

Tableau 4.5 – Pénalité de latence/req.

menée sur une période d'un mois avec une période d'une heure pour la mise à jour dynamique du prix des ressources. Les prix de l'internalisation sont fixés dynamiquement par les CSP à l'aide de l'équation (4.1) et modifiés chaque heure (période T). Nous supposons que le coût du réseau sortant est défini sur 50% de celui d'Amazon *Cloud*, soit (0,09$/Go).

4.4.3 Résultats de l'évaluation

Cette partie présente une analyse des résultats obtenus à partir des expérimentations réalisées avec CloudSim avec les paramètres définis dans la section précédente.

4.4.3.1 Pertinence des sous-coûts

Nous rappelons que le coût de placement global est constitué du coût de placement des objets des clients internes et du coût de placement des objets des clients externes.

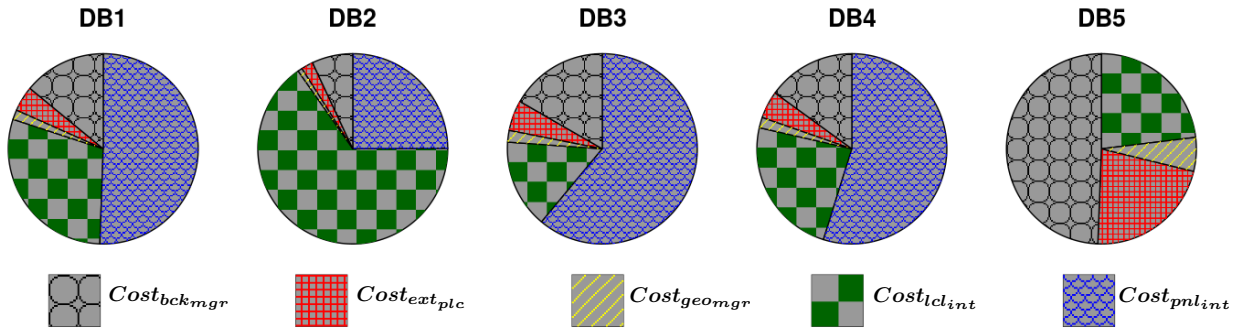
Le coût de placement des objets des clients internes est composé du coût de placement local ($Cost_{lcl_{int}}$), du coût d'externalisation ($Cost_{out_{src}}$), du coût de rapatriement ($Cost_{bck_{mgr}}$) et du coût de pénalité ($Cost_{pnt_{int}}$). Le coût d'externalisation est composé de coût de géo-migration ($Cost_{geo_{mgr}}$), du coût de geo-réplication ($Cost_{geo_{rpl}}$), et du coût de placement externe ($Cost_{ext_{plc}}$). Nous avons supposé que les objets ne sont pas répliqués. De ce fait, le coût de réplication n'est pas considéré dans les évaluations.

Le coût de placement des objets des clients externes est composé du coût d'internalisation ($Cost_{in_{src}}$) et du coût de renvoi ($Cost_{re_{mgr}}$). Le coût d'internalisation est constitué du coût de placement local ($Cost_{lcl_{ext}}$), du coût de pénalité ($Cost_{pnt_{ext}}$) et du coût de réception des objets des clients externes $Cost_{rcv_{ext}}$

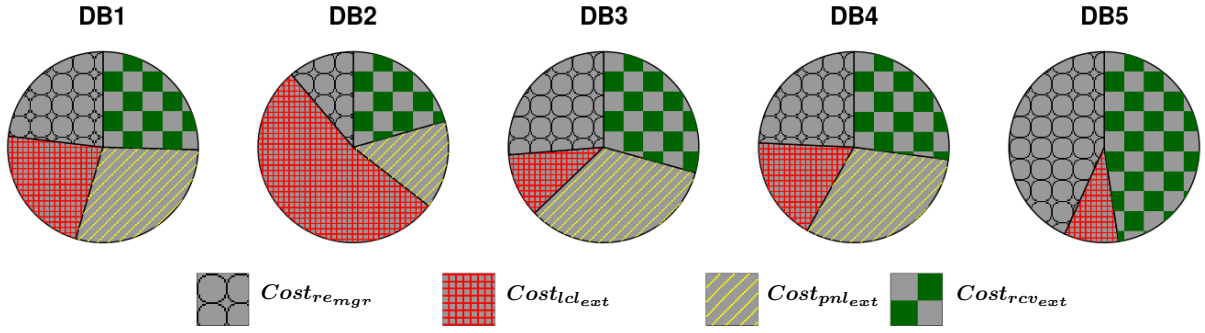
Les figures 4.3a et 4.3b illustrent les différents sous-coûts de notre modèle de coût (les feuilles de la figure 4.2, voir la page 85). Nous observons que l'ensemble des coûts modélisés sont pertinents car chaque coût est suffisamment élevé pour au moins une base de données testée et peut donc avoir un impact significatif sur le coût total. Nous observons que les coûts locaux de placement et de pénalité des clients internes sont les plus élevés. Cela est dû au fait que seul un sous-ensemble des clients est mobile (20%) et pour de petites périodes (1 semaine). En outre, la taille de la base de données affecte directement le coût de placement externe, car

le stockage dans ce cas est acheté auprès d'autres fournisseurs. La taille de la base de données affecte également tous les coûts, y compris la géo-migration et le rapatriement. Par exemple, le coût de rapatriement représente plus ou moins la moitié du coût total pour DB5 (l'objet ayant la plus grande taille). La même chose pour le coût de géo-migration, le pourcentage de ce coût pour DB5 et le plus grand par rapport aux autres bases de données. En outre, les coûts restants sont affectés par le type de périphérique de stockage et les modèles de charge de travail. Par exemple, le coût de placement local représente plus de la moitié du coût total pour DB2 car cette dernière est caractérisée par une charge de travail lourde et est placée sur un SSD.

Les figures 4.4a et 4.4b montrent le coût des bases de données internalisées et externalisées pour un CSP donné pour notre modèle de coût par rapport aux modèles de l'état de l'art. Un premier constat que l'on peut tirer est que les modèles de [147, 148, 150] ne prennent pas en compte le coût d'internalisation. De plus, le coût des services de stockage des objets externalisés est calculé selon l'occupation/unité de temps sans prendre en compte la charge de travail. Dans [18, 19], seul le coût d'occupation est considéré sans le coût de géo-migration ce

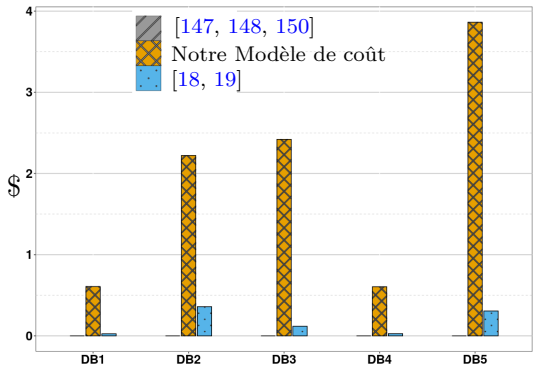


(a) Sous-coûts de placement des objets des clients internes

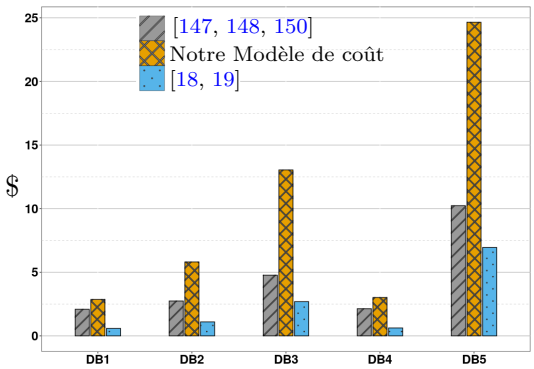


(b) Sous-coûts de placement des objets des clients externes

FIGURE 4.3 – Répartition des sous-coûts du coût de placement simulé



(a) Coût de la base de données internalisée



(b) Coût de la base de données externalisée

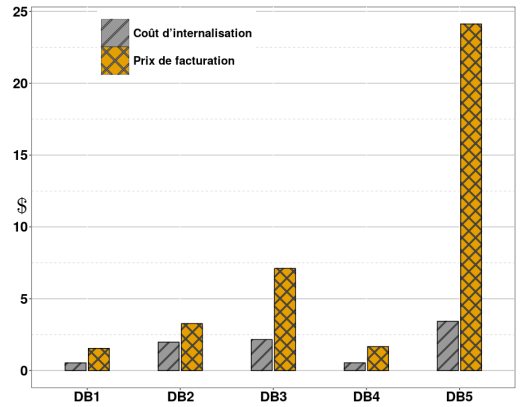
FIGURE 4.4 – Le coût de placement moyen par semaine des bases de données internalisées et externalisées pour un CSP donné.

qui peut conduire, dans le pire des cas, à une différence avec notre modèle de coût de 95% pour le coût de placement des objets des clients externes et de 80% pour le coût d'externalisation. De plus, le coût de pénalité est ignoré dans ces études alors que dans nos travaux il peut représenter jusqu'à 61% du coût de placement des objets des clients internes et 32% de celui des objets des clients externes.

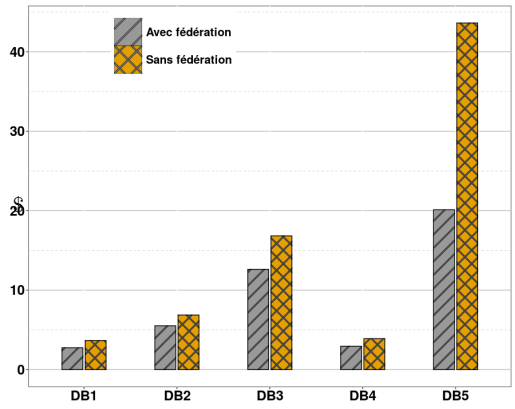
4.4.3.2 Évaluation du modèle de tarification des ressources non utilisées

La figure 4.5a montre, d'une part, le coût d'exploitation des bases de données internalisées, et d'autre part, le prix facturé aux *Clouds* partenaires. Comme on peut le constater, le gain lié à l'opération d'internalisation des objets dépend fortement de la nature de la charge de travail exécutée et le volume de données. Bien que le gain d'internalisation de la base de données DB5 est (x4) le coût d'internalisation, celui de la base de données DB2 est de 34.%. L'internalisation de gros volumes de données générant un faible nombre de requêtes (DB5) est bien entendu plus intéressant.

La figure 4.5b illustre le coût des objets externalisés lors de l'achat de ressources auprès de *Clouds* partenaires par rapport à leur achat depuis un *Cloud* externe à la fédération. Cette figure montre la capacité du modèle à mettre en évidence le gain obtenu pour un CSP donné connaissant ses objets externalisés et le modèle de tarification de fédération utilisé. Cela peut aider un CSP à optimiser la décision d'externalisation. Selon nos évaluations, le gain généré par l'achat des ressources auprès une fédération peut aller de 19.65% (DB2) à 53.87% (DB5).



(a) Coûts d'internalisation et prix facturation



(b) Coût d'externalisation avec et sans fédération

FIGURE 4.5 – Impact du modèle de tarification sur les coûts d'internalisation et d'externalisation

4.4.3.3 Impact de la latence : Fédération vs *Cloud* distribué homogène vs *Cloud* centralisé

Nous remarquons, à partir de la figure 4.6, que généralement, le coût sans externalisation des objets des clients mobiles augmente avec l'augmentation de la latence du réseau. Pour les bases de données à forte charge de travail (par exemple DB2), il est toujours intéressant d'externaliser les objets. En effet, l'externalisation est intéressante tant que la quantité de données externalisées est petite et la charge de travail est conséquente. Ce n'est pas le cas pour les données avec une charge de travail réduite et un volume important. La raison en est que la géo-migration et le rapatriement de ces données impliquent un coût élevé du trafic réseau (par exemple, DB5). Pour certaines bases de données de grande taille et de charges de travail moyennes, il n'est pas rentable de les externaliser tant que la violation de la latence n'est pas élevée. Cependant, lorsque la dégradation de la latence du réseau atteint un certain niveau, l'externalisation devient intéressante. Par exemple, pour DB3 le coût sans migration (avec pénalité) est inférieur au coût avec migration lorsque la latence est inférieure à $400ms$ mais c'est l'inverse lorsque la latence devient supérieure à $600ms$.

Notre modèle de coût peut être utilisé pour trouver un compromis entre tous ces paramètres pour optimiser le coût global d'exploitation d'un *Cloud* dans une fédération.

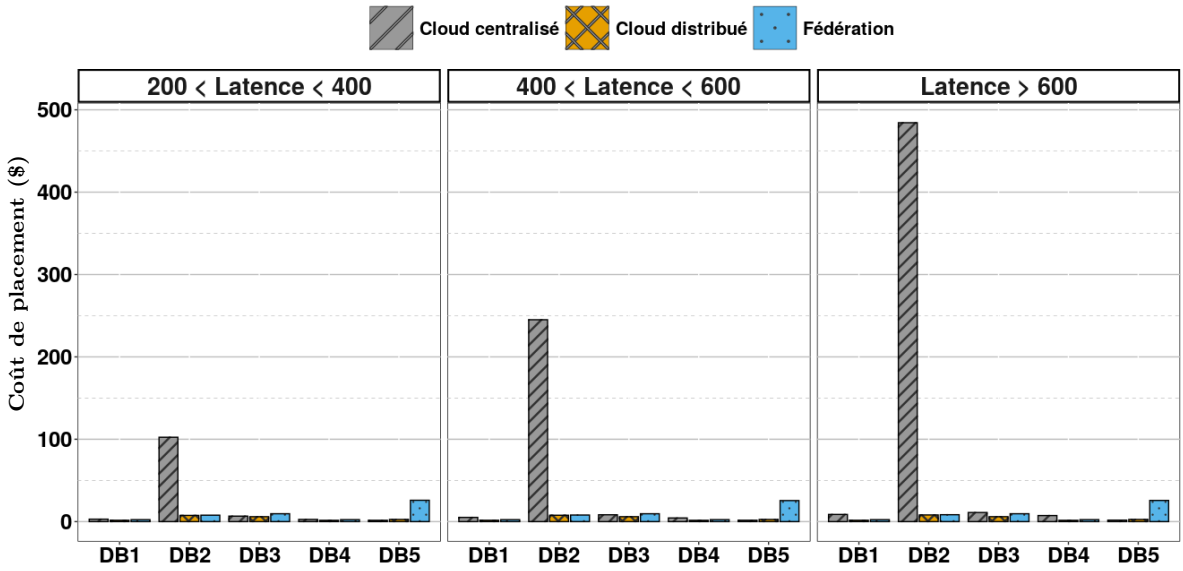


FIGURE 4.6 – Coût moyen avec migration dans un *Cloud* fédéré VS un *Cloud* distribué VS coût moyen sans migration

4.5 Conclusion

Un CSP peut satisfaire la QoS de ses clients et réduire le coût de placement des objets de ses clients en utilisant des ressources locales et/ou celles de ses *Clouds* partenaires de la fédération.

Dans ce travail, nous avons modélisé le coût de placement des objets des clients d'un CSP appartenant à une fédération pour une période de temps donnée T. Le coût total de placement des objets représente la somme du coût de placement des objets des clients internes et du coût de placement des clients externes.

Le coût de placement des objets des clients internes est composé du coût de placement local, du coût d'externalisation, du coût de rapatriement des objets qui ont été déjà externalisés ainsi que du coût de pénalité. Le coût de placement des objets des clients externes est constitué du coût d'internalisation et du coût de renvoi des objets qui ont été déjà internalisés. Noter que les coûts non récurrents qui ne dépendent pas du placement des objets, tels que les coûts de maintenance, les coûts en ressources humaines et les coûts de refroidissement n'ont pas été pris en compte dans notre modèle. Nous n'avons pas non plus pris en compte les coûts liés à la sécurité des données.

Notre modèle étend les travaux de l'état de l'art en tenant compte des coûts de géo-migration, de pénalité, de rapatriement et de géo-réplication. Ce modèle de coût peut être utilisé par un

Cloud hybride ou un *Cloud* distribué composé de plusieurs centres de données. Il peut être aussi utilisé par un *Cloud* n'ayant pas d'infrastructure physique. De plus, plusieurs modèles de facturation de ressources fédérées existent. Le modèle de coût proposé peut permettre d'évaluer ces différents modèles de facturation.

Les évaluations réalisées ont prouvé la pertinence des coûts considérés. Il est également à souligner que l'externalisation et l'internalisation sont des tâches complexes qui nécessitent la prise en compte d'un grand nombre de paramètres. Même si notre modèle est orienté stockage, il peut être utilisé et intégré dans un modèle de coût plus large en tenant compte d'autres ressources (par exemple, le processeur et la mémoire).

Après avoir présenté le modèle de coût, nous décrivons, dans le chapitre suivant, notre approche de placement de données où nous intégrons notre modèle de coût dans un processus d'optimisation.

Chapitre 5

Optimisation multi-objectif pour le placement des données dans un *Cloud* fédéré

La fédération de Clouds permet aux fournisseurs de services de collaborer pour fournir de meilleurs services aux clients. Pour les services de stockage, l'optimisation du placement des objets des clients pour un CSP membre d'une fédération est un véritable défi.

Dans ce travail, nous modélisons le problème de placement d'objets sous forme d'un problème d'optimisation multi-objectifs. Notre but est d'obtenir un ensemble de solutions non dominées qui minimisent simultanément les coûts de stockage, de latence du réseau et de migration. Ces coûts sont liés et contradictoires dans certains cas. Le modèle de coût proposé [180] a été exploité pour définir les fonctions objectifs. Le modèle de placement est soumis à différentes contraintes liées à l'hétérogénéité des classes et services de stockage locaux et fédérés, la charges de travail des clients et leur SLA.

Pour résoudre ce problème de placement, nous proposons CDP-NSGAII$_{IR}$ *(a Constraint Data Placement matheuristic based on NSGAII with Injection and Repair functions)*, une matheuristique pour le placement de données basée sur *NSGAII* avec des fonctions d'injection et de réparation. La fonction d'injection vise à améliorer la qualité des solutions. Elle consiste à calculer des solutions à l'aide d'une méthode exacte puis à les injecter dans la population initiale de NSGAII (voir la section 2.3.5). La fonction de réparation garantit que les solutions sont réalisables, i.e. obéissent aux contraintes du problème et empêche ainsi d'explorer de grands ensembles de solutions irréalisables. Ainsi, cette fonction de réparation réduit considérablement le temps d'exécution de NSGAII.

En effet, CDP-NSGAII$_{IR}$ est composé des deux étapes suivantes :

— **Étape de pré-traitement :** dans cette étape, le calcul d'un ensemble de solutions exactes à l'aide d'un solveur est effectué. En effet, le problème a été transformé en problème d'optimisation mono-objectif en normalisant et en pondérant les différentes fonctions objectifs afin de les agréger en une seule. CPLEX [177] a été utilisé pour résoudre le problème mono-objectif.

— **Étape évolutionnaire :** tout d'abord, l'ensemble de solutions précédemment obtenues est injecté dans la population initiale de la méta-heuristique NSGAII. Le processus évolutif de NSGAII est exécuté avec une fonction de réparation qui consiste à corriger les individus non valides à chaque itération.

Les résultats expérimentaux montrent que la fonction d'injection améliore la qualité des ensembles de solutions (mesurée par l'hypervolume) de NSGAII et la méthode exacte jusqu'à 94% et 60% respectivement tandis que la fonction de réparation réduit le temps d'exécution d'environ 68%. Il est à noter que la méthode exacte calcule un sous ensemble des solutions supportées.

Notre approche de placement a été publiée dans la revue ACM Transactions on Storage (TOS), Volume 17, 2021, [185].

Dans les parties qui suivent, nous rappelons la problématique traitée dans ce chapitre dans la section 5.1. Puis, nous donnons un aperçu du système considéré dans la section 5.2. La formulation du problème de placement est présentée dans la section 5.3. Par la suite, nous détaillons l'approche matheuristique proposée pour résoudre le problème de placement dans la section 5.4. Enfin, nous présentons les expérimentations et analysons les résultats dans la section 5.5 et terminons ce chapitre avec la section 5.6.

Sommaire

5.1 Problématique

La fédération de *Clouds* permet à différents fournisseurs de services *Cloud* de travailler en collaboration afin d'améliorer leur productivité et d'offrir de meilleurs services aux clients. Pour le service de stockage de données dans le *Cloud*, les performances des E/S des supports de stockage et la latence du réseau sont parmi les métriques principales considérées par les clients. De ce point de vue, placer efficacement les objets des clients tout en minimisant le coût de placement pour un *Cloud* membre d'une fédération constitue un véritable défi.

Afin d'optimiser le placement de données, les coûts de stockage, de migration et de latence doivent être pris en considération. Ces coûts sont corrélés et dans certains cas contradictoires.

Minimiser les coûts de placement est contradictoire avec la satisfaction des exigences des clients. Par exemple, le placement des objets dans un emplacement peut réduire les coûts mais ne vérifie pas les performances d'E/S de stockage exigées par les clients et/ou génère une latence de réseau élevée. Par ailleurs, la satisfaction des performances d'E/S de stockage et de la latence du réseau peuvent être contradictoires dans certaines circonstances. Par exemple, placer un objet d'un client dans un emplacement peut respecter les performances d'E/S de stockage demandées mais peut générer une latence élevée et inversement, un placement proche du client, générant une faible latence, peut se faire sur une ressource dont les performances d'E/S ne sont pas satisfaisantes. De plus, la configuration du placement doit s'adapter à l'environnement dynamique et hétérogène. En effet, les ressources locales et externes utilisées ont des caractéristiques différentes en termes de capacité, de performance et de prix. En outre, les CSP de la fédération peuvent mettre à jour leurs prix de ressources de stockage en fonction de leur disponibilité. De plus, les clients ont des charges de travail différentes qui peuvent changer avec le temps. Aussi, les clients peuvent être mobiles ou avoir des utilisateurs dispersés géographiquement, ce qui a un impact sur la latence du réseau pour accéder à leurs objets.

Faire face à un tel système **dynamique** et **hétérogène** rend les stratégies de placement de données très difficiles à définir et une formulation détaillée du problème de placement doit prendre en compte tous les facteurs liés au système afin de prendre des décisions adéquates pour placer les objets des clients.

5.2 Aperçu du système

Nous considérons une fédération F composée de plusieurs *Clouds* répartis sur différentes zones géographiques comme illustré dans la figure 5.1. Il peut y avoir un ou plusieurs CSP dans chaque zone.

Comme mentionné auparavant, on s'intéresse au placement des objets du point de vue d'un seul CSP.

Le CSP concerné héberge une infrastructure hybride composée de plusieurs classes de sto-

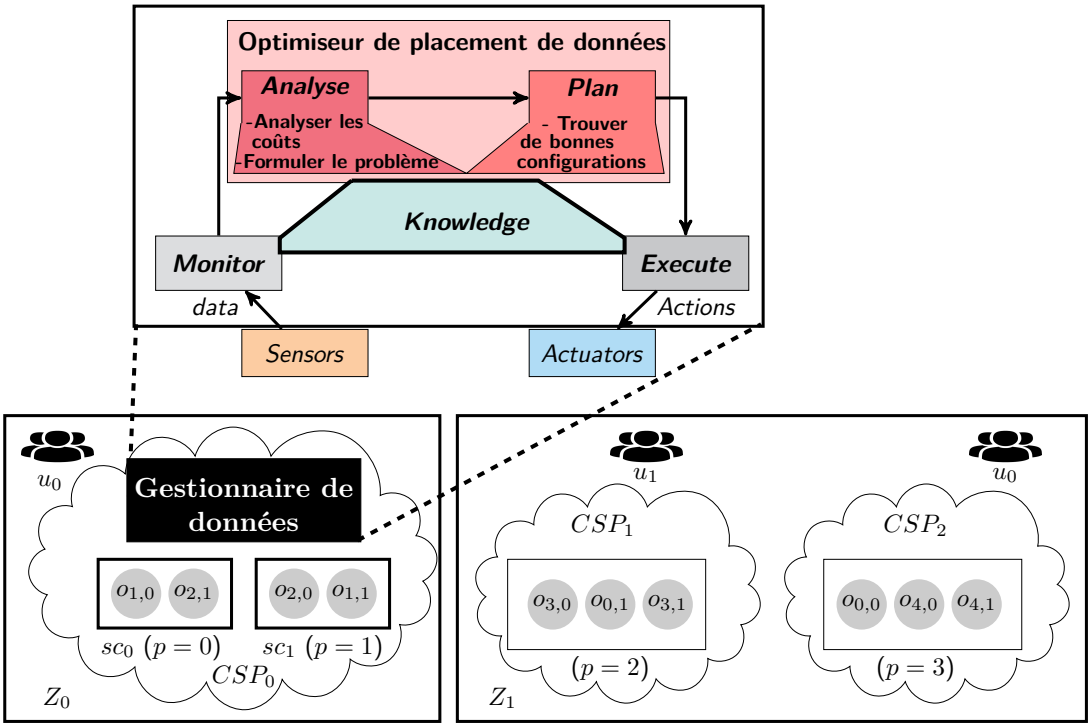


FIGURE 5.1 – Instance de modèle de système

ckage pour stocker les objets de ses clients. Chaque classe de stockage est caractérisée par sa capacité, ses performances en terme d'IOPS, son prix d'achat et sa durée de vie. De plus, il peut exploiter les services de stockage proposés par ses CSP partenaires en externalisant les objets de ses clients auprès de ces derniers, par exemple, lorsqu'il ne peut pas répondre aux exigences de QoS de ses clients ou lorsque le placement d'un objet dans un emplacement coûte moins cher que dans un autre. Les services de stockage externes proposés par les *Clouds* partenaires possèdent différentes caractéristiques en termes de capacité de stockage, performances d'E/S et coûts (occupation, IOPS et transfert réseau).

Dans ce travail, le processus d'optimisation du placement des données est effectué par le composant **Gestionnaire de données** comme cela est indiqué dans la figure 5.1. En effet, le **Gestionnaire de données** intègre les différentes étapes du modèle de gestion autonomique MAPE-K. Ces différentes étapes sont décrites ci-dessous.

Dans un premier temps, l'étape *Monitor* consiste à effectuer un suivi continu de l'environnement fédéré en collectant et en agrégeant les paramètres surveillés tels que les ressources de stockage disponibles internes et externes (des autres CSP), leurs prix, le placement actuel des objets, les emplacements des clients et la latence de leurs requêtes, etc. Des outils de surveillance, tels qu'Amazon CloudWatch [186], OpenStack Watcher [187] ou EZIOTracer [188], peuvent être utilisés dans cette étape. De plus, chaque CSP peut partager une partie de ses propres données

surveillées avec les autres membres de la fédération.

Ensuite, les données surveillées sont passées au module **Optimiseur de placement de données**. Ce module est composé des étapes *Analyse* et *Plan* du processus MAPE-K. L'étape *Analyse*, à son tour, se compose de deux phases. Dans la première phase, les métriques surveillées sont introduites dans un modèle de coût afin de calculer les coûts liés au placement des données. La seconde phase utilise les métriques surveillées en plus des résultats de la première phase afin de construire et formaliser un problème d'optimisation. L'étape *Plan* incorpore certaines méthodes qui visent à trouver les configurations de placement de données qui permettent un compromis entre certains critères donnés. Enfin, sur la base des résultats obtenus dans l'étape *Plan* et des aspects non techniques tels que la réputation des CSP, dans l'étape *Execute*, un schéma de placement est sélectionné et le placement des objets clients est reconfiguré de manière autonome.

La base de connaissances (*Knowledge*) possède toutes les informations sur les modèles et l'historique des étapes précédentes du modèle MAPE-K, et donc chaque étape peut exploiter ces données. Par exemple, pour l'étape *Monitor*, la base de connaissances maintient l'historique des données collectées, cette étape peut utiliser la base de connaissances afin de prédire les données futures sans nécessiter de surveillance du système.

On s'intéresse particulièrement au composant **Optimiseur de placement de données**, c'est-à-dire, aux étapes *Analyse* et *Plan* du modèle MAPE-K en formulant et en résolvant le problème de placement parce que le *monitoring* a été traité dans d'autres travaux.

5.3 Formulation du problème

Notre objectif est de trouver une bonne configuration de placement pour les objets des clients d'un CSP en utilisant de manière opportuniste à la fois ses classes de stockage internes et les services de stockage des CSP partenaires. La configuration de placement recherchée doit respecter les exigences de QoS des clients tout en minimisant les coûts pour le CSP.

Lorsque nous analysons le coût de placement (voir la section 4), nous constatons qu'il est composé de trois composantes principales qui sont : **(1) le coût de stockage** lié aux coût de l'occupation et de la charge d'E/S, **(2) le coût de migration** qui est défini par les différentes migrations dans le système, et **(3) le coût de latence du réseau** qui est la pénalité générée par la détérioration de la QoS causée par le latence du réseau. Ces coûts sont dans certains cas contradictoires. Par exemple, placer des données localement peut être peu coûteux en termes de coûts de stockage et de migration, mais peut générer une mauvaise latence du réseau pour les clients situés dans d'autres régions. Aussi, un emplacement qui minimise à la fois les coûts de stockage et de latence peut ne pas être intéressant car la migration vers cet emplacement génère un coût prohibitif. Il est donc parfois inévitable de faire un compromis lors du placement des objets.

Afin de donner plus de flexibilité à l'administrateur du *Cloud* pour mettre à jour la configuration de placement des objets des clients, nous formulons le problème de placement sous la forme d'un problème d'optimisation multi-objectifs prenant en compte les trois fonctions à savoir (1) le coût de stockage, (2) le coût de migration, et (3) le coût de latence. L'optimisation est soumise à des contraintes liées aux ressources locales et externes limitées et aux SLA des clients.

Dans cette section, nous allons, dans un premier temps, définir les fonctions de coût du problème de placement et montrer comment les évaluer. Ensuite, les contraintes à respecter sont décrites.

5.3.1 Fonctions objectifs

Le problème abordé dans ce travail consiste à déterminer un placement des objets des clients sur des systèmes de stockage internes et/ou externes de manière à fournir de meilleures performances d'accès aux données tout en minimisant le coût de placement. Les modèles d'accès aux données évoluent dans le temps et les clients sont mobiles ou leurs utilisateurs peuvent être répartis géographiquement. Les exigences de QoS des clients sont exprimées en termes de performances d'E/S de stockage et de latence du réseau. Placer un objet d'un client dans un endroit peut satisfaire une exigence de QoS mais pas l'autre exigence. Par ailleurs, compte tenu des caractéristiques du système, pour atteindre l'objectif principal (minimiser le coût de placement et satisfaire les SLA des clients), la configuration de placement des objets des clients peut évoluer dans le temps. Cela signifie que certains objets doivent être migrés. Le coût de la migration des objets est variable selon les propriétés des sources et destinations, ce qui rend le coût de migration une autre dimension à optimiser. Par conséquent, les coûts de stockage, de latence du réseau et de migration sont considérés comme les critères à optimiser pour le problème de placement, comme indiqué dans l'équation (5.1).

$$F = \begin{bmatrix} Store \\ Migrate \\ Latency \end{bmatrix} \tag{5.1}$$

Où $Store$, $Migrate$ et $Latency$ représentent respectivement les coûts de stockage, de migration et de latence.

La figure 5.2 montre un aperçu des fonctions objectif considérées. Elles englobent le coût global de placement défini dans le chapitre 4 et seront décrits dans les sections suivantes.

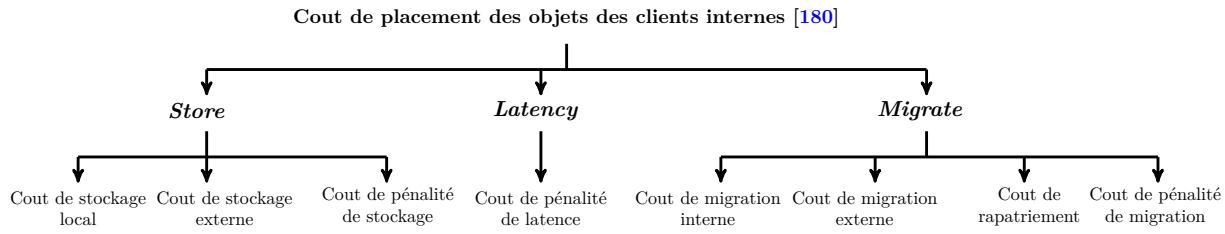


FIGURE 5.2 – Fonctions objectif basées sur le modèle de coût

5.3.1.1 Coût du stockage

Store est le coût de stockage des objets des clients. Il est composé des coûts de stockage local et externe $Cost_{stg_{int},T}$ et $Cost_{ext_{plc},T}$ de l'équation (4.5) et de l'équation (4.18). Il inclut également le coût de pénalité de la partie performances de stockage d'E/S $Cost_{pnt_{iops},T}$ (voir l'équation (4.26)). Nous rappelons que cette pénalité représente le coût monétaire supplémentaire causé par la violation du SLA d'E/S. *Store* est défini comme suit :

$$Store = Cost_{stg_{int},T} + Cost_{ext_{plc},T} + Cost_{pnt_{iops},T} \tag{5.2}$$

5.3.1.2 Coût de la migration

Migrate englobe le coût de migration interne $Cost_{mgr_{int},T}$ de l'équation (4.4) (voir la page 86) lorsque les objets sont déplacés entre les classes de stockage internes, le coût migration externe $Cost_{geo_{mgr},T}$ de l'équation (4.9) (voir la page 87) lorsque les objets sont envoyés vers un autre CSP et le coût de rapatriement $Cost_{bck_{mgr},T}$ à partir de l'équation (4.3) (voir la page 86) lorsque les objets sont renvoyés à l'infrastructure locale. Nous avons également considéré le coût de pénalité lié à l'opération de migration $Cost_{pnt_{mgr},T}$. Ce coût est induit par la dégradation de la QoS provoquée par la migration. Nous l'avons calculé en termes de nombre total de clients K, le temps de migration maximum $mrg_{time,T}$ et certains coûts unitaires α comme indiqué dans l'équation (5.3).

Nous avons considéré α comme un coût unitaire de performance de stockage et de latence en supposant que pendant le temps de migration, la QoS n'est pas respectée. En faisant ainsi, ce coût de pénalité de migration représente une limite supérieure car la dégradation de la QoS n'affectent pas nécessairement tous les clients. Dans notre évaluation, nous avons défini α comme la somme du coût correspondant aux IOPS moyens nécessaires par les clients et du coût de latence correspondant aux requêtes moyennes envoyées par les clients par seconde.

$$Cost_{pnt_{mgr},T} = \alpha * K * mrg_{time,T} \tag{5.3}$$

$$Migrate = Cost_{mgr_{int},T} + Cost_{geo_{mgr},T} + Cost_{bck_{mgr},T} + Cost_{pnt_{mgr},T} \tag{5.4}$$

5.3.1.3 Coût de la latence

La latence du réseau a été considérée comme un coût dans de nombreuses études comme [189, 23, 134]. Dans notre travail, ce coût (*Latency*) reflète le coût de pénalité de la latence réseau. Comme nous l'avons déjà mentionné dans le Chapitre 4, la latence du réseau est facturée aux clients, et la détérioration de la latence délivrée entraîne des pertes pour le CSP en fonction du niveau de sa dégradation. Concrètement, il s'agit de la partie du coût de pénalité liée à la latence du réseau ($Cost_{pnt_{ltc},T}$ extrait de l'équation (4.25)). Ce coût est la somme de toutes les pénalités de latence des clients, comme indiqué dans l'équation. 5.5 :

$$Latency = Cost_{pnt_{ltc},T} = \sum_{u_k \in U} Cost_{pnt_{ltc},T}(u_k) \tag{5.5}$$

$Cost_{pnt_{ltc},T}(u_k)$ représente la pénalité de latence du réseau du client u_k. Nous l'avons calculé en multipliant le nombre de requêtes émises par le client u_k ou ses utilisateurs finaux depuis différentes zones par la pénalité de latence du réseau. Cette pénalité est liée à la latence entre la localisation de u_k ou de ses utilisateurs finaux et celle des objets demandés. Il est défini selon le tableau 4.1. Ainsi, $Cost_{pnt_{ltc},T}(u_k)$ est calculé comme dans l'équation (5.6) :

$$Cost_{pnt_{ltc},T}(u_k) = \sum_{Z_m} \sum_{o_{i,k} \in O_k} tot_k^{m,o_{i,k}} * penalty(l(m, x_{o_{i,k}})) \tag{5.6}$$

5.3.2 Contraintes

Dans ce travail, nous avons des contraintes liées à la limitation de capacité et du débit des systèmes de stockage locaux et externes. De plus, il existe également des contraintes liées aux SLA des clients.

Pour chaque classe de stockage locale sc_j, la somme des tailles des objets affectés à cette classe ($S_{o_{i,k}}$) ne doit pas dépasser sa capacité de stockage csc_j. Ceci est exprimé dans l'équation (5.7). De plus, la capacité de service de stockage allouée css_d de chaque CSP_d de la fédération est limitée. Par conséquent, la somme des tailles d'objets affectés à chaque service de stockage

partenaire ss_d ne doit pas dépasser sa capacité, comme indiqué dans l'équation. (5.8)

$$\sum_{u_k} \sum_{o_{i,k} \in sc_j} S_{o_{i,k}} \leq csc_j \qquad \forall j < J \tag{5.7}$$

$$\sum_{u_k} \sum_{o_{i,k} \in ss_d} S_{o_{i,k}} \leq css_d \qquad \forall d \geq J \tag{5.8}$$

Les débits des systèmes de stockage internes et externes sont limités. Pour les classes de stockage local, l'équation (5.9) exprime que la somme de la charge d'E/S composée des différentes opérations d'E/S des objets ($io_{o_{o_{i,k}}}(op)$) affectés à chaque classe de stockage sc_j doit être inférieure au débit d'E/S fourni par cette classe de stockage $io_j(op)$. D'autre part, l'équation (5.10) indique que la charge globale d'E/S émise par tous les objets affectés à un service de stockage donné ss_d d'un partenaire CSP_d ne doit pas dépasser les performances d'E/S de ce service de stockage io_d.

$$\sum_{op \in OP} \frac{\sum_{u_k} \sum_{o_{i,k} \in sc_j} io_{o_{o_{i,k}}}(op)}{io_j(op)} \leq 1, \qquad \forall j < J \tag{5.9}$$

$$\sum_{op \in OP} \frac{\sum_{u_k} \sum_{o_{i,k} \in ss_d} io_{o_{o_{i,k}}}(op)}{io_d} \leq 1, \qquad \forall d \geq J \tag{5.10}$$

Concernant le SLA d'E/S des clients, lorsque $ioOffered_k$ d'un client donné u_k est compris entre $ioSoft_k$ et $ioHard_k$, le CSP subit une pénalité d'E/S de stockage. Cependant, $ioHard_k$ doit être satisfait. Cela implique que les performances de stockage $ioOffered_k$ de chaque client doivent dépasser ses $ioHard_k$ comme indiqué dans l'équation (5.11).

$$ioOffered_k >= ioHard_k \qquad \forall u_k \in U \tag{5.11}$$

Enfin, nous supposons que les objets ne sont pas répliqués. Cette condition est garantie par les variables de décision utilisées lors de la résolution du problème.

5.3.3 Récapitulatif

Compte tenu des entrées du système, de la fonction objectif et des contraintes, nous définissons le problème d'optimisation multi-objectifs de placement comme suit :

$$\min \begin{bmatrix} Store \\ Migrate \\ Latency \end{bmatrix} \tag{5.12a}$$

$$S.T. \quad \sum_{u_k} \sum_{o_{i,k} \in sc_j} S_{o_{i,k}} \leq csc_j \qquad \forall j < J \tag{5.12b}$$

$$\sum_{u_k} \sum_{o_{i,k} \in ss_d} S_{o_{i,k}} \leq css_d \qquad \forall d \geq J \tag{5.12c}$$

$$\sum_{op \in OP} \frac{\sum_{u_k} \sum_{o_{i,k} \in sc_j} io_{o_{o_{i,k}}}(op)}{io_j(op)} \leq 1, \qquad \forall j < J \tag{5.12d}$$

$$\sum_{op \in OP} \frac{\sum_{u_k} \sum_{o_{i,k} \in ss_d} io_{o_{o_{i,k}}}(op)}{io_d} \leq 1, \qquad \forall d \geq J \tag{5.12e}$$

$$io\text{-}offered_k >= ioHard_k \qquad \forall u_k \in U \tag{5.12f}$$

L'équation (5.12a) représente la fonction objectif, où $Store$ est le coût de stockage, $Migrate$ le coût de migration et $Latency$ le coût de latence du réseau.

Les deux premières contraintes (équation (5.12b)) et (équation (5.12c)) expriment la limitation de capacité des ressources de stockage internes et externes. Les contraintes (équation (5.12d)) et (équation (5.12e)) indiquent que ces ressources de stockage ont une performance d'E/S finie. Enfin, la contrainte (équation (5.12f)) garantit que le SLA (performances de stockage) doit être limité.

Pour résoudre le problème ci-dessus, nous devons utiliser un algorithme d'optimisation multi-objectifs adapté à notre problème. Nous avons choisi NSGAII pour résoudre le problème de placement. Cependant, comme les méta-heuristiques évolutionnaires ne fonctionnent pas toujours bien dans les problèmes difficiles avec des espaces combinatoires complexes liés à un nombre élevé de contraintes [120], nous exploitons une formulation mathématique de notre problème de placement tout en la combinant avec NSGAII. La matheuristique résultante est l'objet de la section suivante.

5.4 CDP-NSGAII$_{IR}$: solution matheuristique pour le placement de données proposées

Une fois le problème défini, nous adaptons un algorithme d'optimisation multi-objectifs à notre problème pour le résoudre.

Le problème de placement est connu pour être NP-difficile [157]. En effet, énumérer toutes les options de placement possibles et finalement choisir la meilleure configuration conduit à une explosion combinatoire.

Les méta-heuristiques constituent une bonne alternative pour alléger la complexité du temps d'exécution. Compte tenu de la popularité et de l'efficacité de l'algorithme de NSGAII [115], nous l'utiliserons pour résoudre notre problème de placement. Cependant, étant un algorithme évolutionnaire, NSGAII souffre de problèmes liés à l'infaisabilité des solutions lorsque la taille du problème augmente comme l'encodage des solutions ne garantie pas leur faisabilité. L'espace de recherche exploré devient alors très grand par rapport à l'espace des solutions faisables. Cela a un impact négatif sur la qualité du front de Pareto qui en résulte et sur le temps d'exécution. Ainsi, notre mise en œuvre de NSGAII a besoin de quelques modifications pour résoudre ces problèmes.

Pour cela, nous avons mis à niveau NSGAII vers une matheuristique [190] en la fusionnant avec une méthode exacte par programmation linéaire avec une fonction objective agrégeant nos 3 fonctions *Store*, *Migrate* et *Latency*. Cette dernière calcule des solutions qui sont ensuite injectées dans la population initiale de NSGAII. De plus, nous avons proposé une fonction de réparation pour améliorer le temps d'exécution de NSGAII. Lorsque les opérateurs de variation de l'algorithme produisent des solutions infaisables, cette fonction modifie la configuration de placement infaisable en déplaçant certains objets vers d'autres emplacements afin de respecter les contraintes de placement. Nous appelons la matheuristique proposée CDP-NSGAII$_{IR}$ pour ***C**onstrained **D**ata **P**lacement **NSGAII** avec des fonctions d'**I**njection et de **R**éparation.*

Dans cette section, nous allons, dans un premier temps, spécifier comment est encodée une solution représentant une configuration de placement. Ensuite, nous décrirons la problématique des opérateurs évolutionnaires qui nous pousse à mettre à jour NSGAII afin d'améliorer ses résultats pour notre problème. Enfin, nous décrirons la matheuristique proposée.

5.4.1 Encodage des solutions

Dans ce travail, une solution pour le placement des objets des clients est représentée au moyen d'un schéma classique d'encodage d'entiers. Avec un tel encodage, chaque individu, nommé aussi chromosome, est défini comme un tableau d'entiers qui représente une configuration de placement de tous les objets de tous les clients dans des emplacements P où $P = D + J - 1$. En effet, du point de vue du CSP, le nombre total d'emplacements P est égal au nombre de classes de

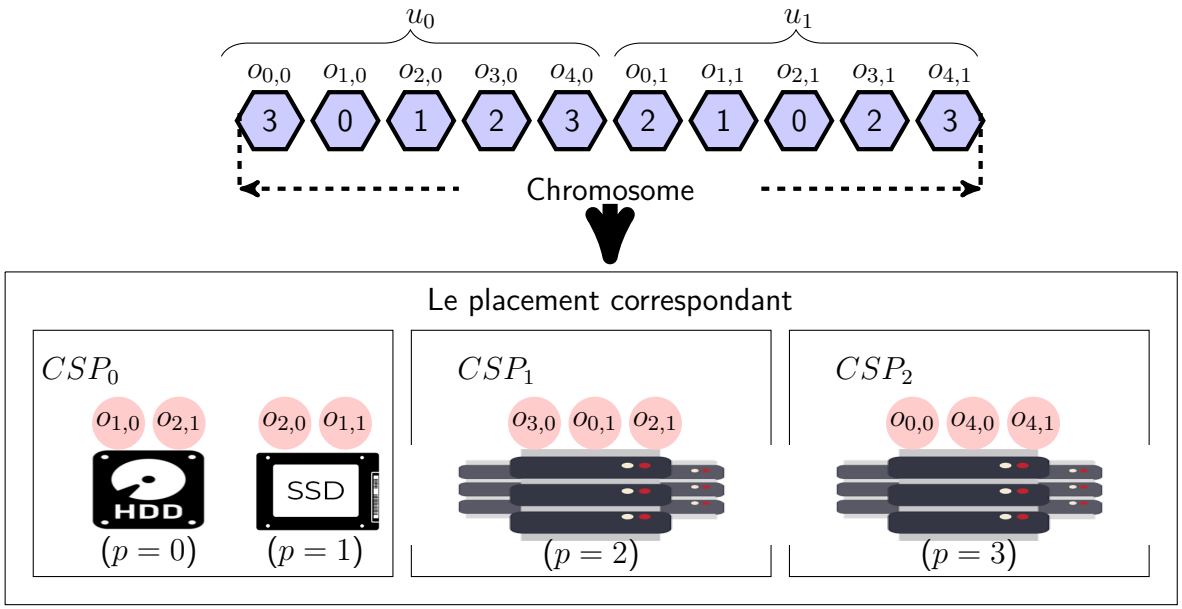


FIGURE 5.3 – Chromosome

stockage locales plus le nombre de CSP partenaires $P = D + J - 1$. Ainsi, un chromosome est un ensemble de gènes. Un numéro de gène i dans un chromosome correspond au numéro d'objet i. La valeur d'un gène i correspond à l'emplacement de l'objet i. Il faut noter que dans ce travail, pour des raisons de simplicité, nous avons supposé que chacun des CSP partenaires offre un seul service de stockage. La généralisation de plusieurs services par CSP n'affectera que l'encodage.

Ainsi une configuration de placement est représentée par le vecteur $x = (x_{o_{0,0}}, x_{o_{1,0}}, \cdots, x_{o_{i,k}} \cdots)$ où $x_{o_{i,k}}$ est un emplacement potentiel pour l'objet $o_{i,k}$.

Pour simplifier, dans notre représentation, un emplacement $p \in [0, J[$ correspond à des classes de stockage locales tandis que $p \in [J, D + J - 1[$ correspond à un emplacement dans un *Cloud* partenaire.

Dans l'exemple illustré dans la figure 5.3, CSP_0 est le CSP concerné par l'optimisation de placement des objets de ses clients. Il héberge 2 classes de stockage. Ainsi, $p = 0$ et $p = 1$ correspondent au placements locaux. $p = 2$ correspond à un placement dans CSP_1 tandis que $p = 3$ représente un placement dans CSP_2.

La configuration du placement présentée dans la figure 5.3 correspond au chromosome $x = (x_{0,0}, ..., x_{4,0}, x_{0,1}, ..., x_{4,1})$ et est représentée par le vecteur $x = (3, 0, 1, 2, 3, 2, 1, 0, 2, 3)$. La taille des chromosomes N indique le nombre d'objets de tous les clients. Dans cet exemple, nous avons 2 clients chacun a 5 objets (voir la figure 5.1), ce qui signifie que la longueur du chromosome est $N = 10$. Le gène $x_{o_{0,0}}$ possède une valeur égale à 3, ce qui signifie que l'objet $o_{0,0}$ est affecté à l'emplacement $p = 3$ qui correspond à CSP_2.

5.4.2 Approche basée sur NSGAII

Parmi les stratégies utilisées pour surmonter la complexité du temps d'exécution figurent les heuristiques et les méta-heuristiques. Comme nous sommes confrontés à un problème multi-objectifs, les algorithmes évolutionnaires multi-objectifs MOEA sont des techniques efficaces pour trouver plusieurs solutions dans un temps raisonnable [122] (voir la section 2.3.4).

Les algorithmes évolutionnaires sont des algorithmes stochastiques itératifs basés sur la théorie de la sélection naturelle darwinienne. Leur idée est de faire évoluer un ensemble de solutions (population) en fonction d'un problème donné au fil des générations, afin de trouver les meilleurs résultats. L'évolution s'effectue via un ensemble d'opérateurs. Généralement, l'opérateur de *selection* remplace les individus pour la génération suivante. Les opérateurs de *mutation* et de *crossover* découvrent de nouvelles solutions en transformant respectivement celles existantes ou en mélangeant des parties de différentes solutions.

L'algorithme génétique de tri non dominé (NSGAII) est un MOEA bien connu utilisé dans de nombreux problèmes d'optimisation tels que [122, 123, 124]. Comme pour les autres méta-heuristiques, NSGAII doit être personnalisé pour le problème ciblé. Un encodage et des opérateurs de variation doivent être choisis ou spécifiquement conçus. Dans notre cas, nous avons adapté une codification standard pour représenter les configurations de placement (voir la section 5.4.1). En outre, des opérateurs de mutation et de croisement standards pour les chaînes de nombres sont utilisés. La mutation polynomiale (PM : Polynomial Mutation) [191] et le croisement uniforme (SBX : Simulated Binary Crossover) [192] ont été employé. En effet, les opérateurs de variation standards peuvent produire n'importe quelle combinaison de valeurs, car ils ne prennent pas en compte les contraintes spécifiques du problème.

Par exemple, pour un nombre d'objets $N = 5$ et un nombre d'emplacements $P = 2$, les solutions [00011] et [11000] peuvent être valides (pas de débordement de ressources), mais elles peuvent être mélangées, par exemple, en produisant la configuration [11011] par un opérateur de croisement, conduisant à une solution irréalisable, car la contrainte sur l'emplacement 1 peut ne pas être respectée ou cet emplacement peut ne pas satisfaire les SLA de tous les clients dont les objets sont placés à cet endroit.

Plus généralement, la taille de l'espace de recherche induite par l'encodage est de P^N où P est le nombre d'emplacements et N le nombre total d'objets, alors que l'espace réalisable peut être beaucoup plus petit, en raison des contraintes de ressources et de clients. Cela pourrait rendre la recherche inefficace, en particulier pour les gros problèmes, lorsque la région infaisable explorée inutilement est énorme. Il en résulte un mauvais ensemble de solutions, qui ne converge pas vers l'ensemble de Pareto.

Il existe trois manières possibles de traiter l'infaisabilité des solutions causée par les contraintes du problème (voir la section 2.3.4). La première consiste à modifier les opérateurs d'exploration, la seconde pénalise les solutions non faisables et la troisième repose sur la construction d'une

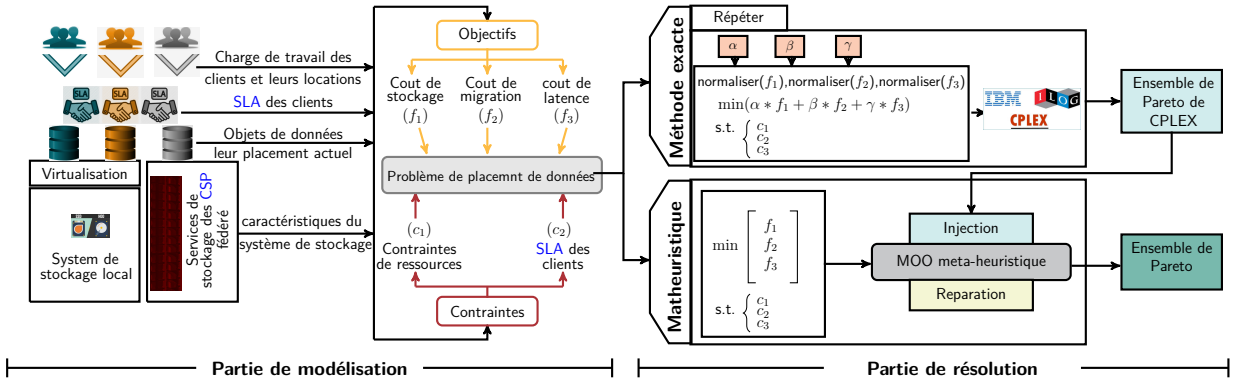


FIGURE 5.4 – Approche d'optimisation matheuristique

fonction de réparation. Nous avons utilisé la solution de réparation pour deux raisons principales : (i) elle peut être assez simple à mettre en œuvre et permet d'utiliser tous les opérateurs standards connus et performants ; (ii) elle ne freine pas l'application de ces opérateurs en vérifiant les contraintes de placement.

En outre, la population initiale de NSGAII est enrichie de quelques solutions calculées par une méthode exacte faisant de NSGAII une matheuristique qui fait l'objet de la section suivante.

5.4.3 CDP-NSGAII$_{IR}$ matheuristique

La figure 5.4 illustre notre approche d'optimisation. La partie modélisation consiste à construire le MOOP en considérant les différentes entrées liées aux caractéristiques des classes de stockage locales et aux spécifications et emplacements des services de stockage des *Clouds* partenaires, la charge de travail, les SLA et la localisation des clients et enfin les spécifications des objets et leur placement actuel.

La seconde partie présente la stratégie de résolution matheuristique (partie de résolution de la figure 5.4). Elle consiste en trois composants.

— Dans le premier composant, un ensemble de solutions initiales est généré à l'aide d'un outil de programmation en nombres entiers mixtes (Mixed Integer Linear Programming : MILP). Nous avons utilisé CPLEX dans ce travail pour résoudre le MILP. Pour l'utilisation d'un tel outil, le MOOP est transformé en un problème mono-objectif en normalisant et en pondérant les différentes fonctions objectifs (voir la partie méthode exacte de la figure 5.4).

— Le deuxième composant est la méta-heuristique évolutive dans laquelle l'ensemble résultant du premier composant est injecté dans la population initiale, complétée par des solutions générées aléatoirement (voir le bloc d'injection dans la figure 5.4). Dans ce travail, nous avons choisi NSGAII.

— Enfin, le dernier composant consiste en la procédure de réparation des individus de chaque génération. Cela se fait en perturbant les individus qui ne verifient pas les contraintes du modèle (bloc de réparation dans la figure 5.4).

Dans ce qui suit, nous expliquons les parties spécifiques que nous avons élaborées et mises en œuvre : (1) comment certaines solutions initiales sont calculées, ensuite, (2) comment fonctionne la fonction de réparation.

5.4.3.1 Méthode exacte basée sur un outil MILP

La méthode exacte dans ce travail consiste à transformer le MOO en un problème mono-objectif et à le résoudre à l'aide d'un outil MILP. La transformation est réalisée en agrégeant et pondérant les différentes fonctions objectifs en une seule fonction. Les poids représentent l'importance des fonctions objectifs associées. Dans notre travail, les poids sont manuellement attribués. Nous les faisons varier plusieurs fois afin d'obtenir plusieurs solutions de l'ensemble de Pareto.

L'algorithme 4 formule le processus de la méthode exacte. Tout d'abord, l'intervalle de valeurs de chaque fonction objectif est calculé (ligne 2 et ligne 3 de l'algorithme 4). Ensuite, les fonctions objectifs sont normalisées (ligne 5 à ligne 7 de l'algorithme 4). Ces fonctions sont pondérées et combinées en une seule fonction objectif. CPLEX est utilisé pour résoudre le problème mono-objectif obtenu (ligne 9 et ligne 10 de l'algorithme 4). Ce processus de résolution est répété plusieurs fois avec des poids différents afin de générer un ensemble de solutions exactes.

Algorithme 4 Calcul de quelques solutions exactes à l'aide d'un solveur MILP

1: **Fonction** CALCULER_SOLUTIONS_EXACTES(n)
2: calculer $minStr$, $minMgr$, $minLtc$
3: calculer $maxStr$, $maxMgr$, $maxLtc$
4: population $\leftarrow\emptyset$
5: $Store_N(x) \leftarrow \frac{Store(x)-minStr}{maxStr-minStr}$
6: $Migrate_N(x) \leftarrow \frac{Migrate(x)-minMgr}{maxMgr-minMgr}$
7: $Latency_N(x) \leftarrow \frac{Latency(x)-minLtc}{maxLtc-minLtc}$
8: **Pour** $i\leftarrow 1$ to n **Faire**
9: lire(α,β, γ) // $\alpha+\beta+\gamma=1$
10: $x \leftarrow CPLEX\_SOLVE(\alpha * Store_N(x) + \beta * Migrate_N(x) + \gamma * Latency_N(x)$ sous toutees les contraintes)
11: population $\leftarrow$ population $\cup$ x
12: **Fin Pour**
13: **Retourner** (population)

5.4.3.2 Fonction de réparation

Lorsqu'une solution ne satisfait pas à toutes les contraintes de placement définies dans les équations (5.7, 5.8, 5.9, 5.10, 5.11), les objets sont déplacés pour aboutir à une solution réalisable. L'algorithme 5 montre comment fonctionne la fonction de réparation.

La fonction de réparation fonctionne selon deux étapes. La première consiste à trouver les objets qui violent les contraintes de placement tandis que la seconde essaie de réviser le placement de ces objets en les déplaçant vers des emplacements adaptés répondant à la fois aux limitations de ressources et aux SLA des clients. Ces étapes fonctionnent comme suit :

Étape 1 : Pour détecter tous les objets qui ne respectent pas les contraintes dans un chromosome (une configuration de placement), nous parcourons le chromosome séquentiellement gène par gène, en vérifiant l'emplacement des objets un par un. Si l'affectation d'un objet donné dans l'emplacement correspondant à la valeur du gène n'implique pas une violation de certaines contraintes, la valeur du gène reste inchangée. Dans l'autre cas, il est ajouté à une liste d'objets infaisables à réparer appelée $listToRepair$.

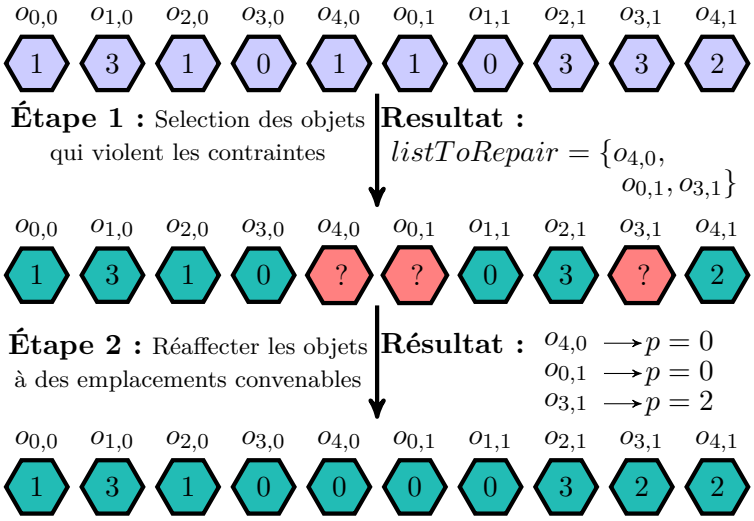


FIGURE 5.5 – Fonction de réparation : exemple

A la fin, $listToRepair$ contient tous les objets qui, avec les valeurs de placement actuelles, ne répondent pas à la capacité ou aux performances des contraintes de ressources de stockage ou dont les emplacements ne satisfont pas les SLA des clients.

Cette étape correspond à l'algorithme 5, de la ligne 2 à la ligne 9.

Prenons la fédération de *Clouds* représentée sur la figure 5.1. Pour simplifier, nous considérons uniquement la contrainte liée aux capacités des classes locales de stockage et des services externes de stockage. Supposons que les capacités des deux classes locales de stockage (csc_0 et csc_1) soient respectivement de 10 et 5 unités. La capacité du service de stockage cs_1 de CSP_1 est égale à 8 et

la capacité cs_2 de CSP_2 est de 5 unités. Nous supposons également que la taille de chaque objet est égale à 2 unités. Une configuration de placement $x = (1, 3, 1, 0, 1, 1, 0, 3, 3, 2)$ décrite dans la figure 5.5 n'est pas réalisable car la capacité des ressources correspondant aux emplacements $p = 1$ (8 unités utilisées sur 5 disponibles) et $p = 3$ (6 unités utilisées sur 5 disponibles) est dépassée.

La première étape de la fonction de réparation recherche les objets qui violent la contrainte de capacité. On vérifie le chromosome objet par objet. Les emplacements des quatre premiers objets $o_{0,0}$, $o_{1,0}$, $o_{2,0}$, $o_{3,0}$ sont conservés inchangés car ils respectent les contraintes de capacité. A ce stade, les capacités de stockage restantes csc_0, csc_1, cs_1, cs_2 sont respectivement (8, 1, 8, 3). Placer l'objet $o_{4,0}$ dans $p = 1$ correspondant à la classe de stockage local sc_1 dépasse sa capacité restante. Nous plaçons donc $o_{0,4}$ dans $listToRepair$. En répétant le même processus sur les objets restants, le résultat de cette $1^{ère}$étape est la liste $listToRepair = \{o_{4,0}, o_{0,1}, o_{3,1}\}$ comme illustré dans la figure 5.5. Les capacités de stockage restantes $(csc_0, csc_1, cs_1, cs_2)$ sont respectivement (6,1,6,1).

Étape 2 : nous parcourons $listToRepair$ dans un ordre aléatoire pour affecter les objets à de nouveaux emplacements. Nous privilégions un traitement dans un ordre aléatoire afin de ne pas privilégier un objet par rapport à un autre. Pour chaque objet, nous construisons une liste LP contenant tous les emplacements existants sauf celui auquel l'objet est actuellement affecté. Ensuite, nous choisissons aléatoirement un nouvel emplacement dans LP et vérifions l'affectation par rapport aux contraintes des ressources et des clients. S'il satisfait toutes les contraintes, nous acceptons l'affectation et définissons la valeur du gène en conséquence. Si ce n'est pas le cas, nous choisissons aléatoirement un autre emplacement parmi LP, et ainsi de suite.

Le choix aléatoire d'un nouvel emplacement a été appliqué pour la réparation d'une solution donnée afin de réduire l'effort de calcul. En effet, vérifier tous les emplacements potentiels pour chaque objet puis choisir un emplacement parmi ceux qui optimisent au mieux les fonctions objectifs augmente considérablement le temps d'exécution.

D'après des tests préliminaires, le temps d'exécution devient considérable lors de la recherche des meilleurs emplacements qui satisfassent les contraintes parmi les emplacements disponibles. En effet, la complexité de choisir aléatoirement un emplacement est dans le pire des cas = $\mathcal{O}(p)$ (quand le dernier emplacement est celui qui satisfait toutes les contraintes ou bien il n'existe pas d'emplacement valide). En revanche, pour choisir le meilleur emplacement, la liste entière des emplacements doit être parcourue pour chaque objets. Dans le pire des cas, le nombres de solutions réalisables est de l'ordre de $\mathcal{O}(p^{|listToRepair|})$. Ensuite, une étape de sélection de solutions non-dominées est appliquée. Enfin, une solution parmi les solutions non-dominées est choisie en appliquant des stratégies dédiées comme la distance euclidienne. Ce processus augmente considérablement le temps d'exécution.

Ensuite, si, pour un objet donné, nous ne trouvons aucun emplacement qui respecte toutes les contraintes, le processus de réparation est interrompu et la configuration initiale du chromosome est conservée en tant que candidat potentiel pour une future mutation. L'algorithme 5 de la ligne 10 à la ligne 25 correspond à cette deuxième étape.

Algorithme 5 Fonction de réparation

1: **Fonction** RÉPARER(*chromosome*)
2: $listToRepair \leftarrow \emptyset$
3: **Pour chaque** gene g dans le chromosome **Faire**
4: $o_{i,k} \leftarrow$ *objet correspondant à* g
5: *Vérifier les contraintes pour* $o_{i,k}$ *en utilisant les équations* {5.7,5.8,5.9,5.10,5.11}
6: **Si** ($o_{i,k}$ *ne respecte pas toutes les contraintes*) **Alors**
7: $listToRepair \leftarrow listToRepair \cup o_{i,k}$
8: **Fin Si**
9: **Fin Pour**
10: **Tant que** ($listToRepair \neq \emptyset$) **Faire**
11: $LP \leftarrow \emptyset$
12: $o_r \leftarrow$ *Choisir aléatoirement un objet parmi listToRepair*
13: *LP.ajouter(tous les emplacements sauf celui affecté à* o_r*)*
14: **Tant que** ($LP \neq \emptyset$) **Faire**
15: $p \leftarrow$ *choisir aléatoirement un emplacement de LP*
16: **Si** (o_r respecte toutes les contraintes des équations. (5.7,5.8,5.9,5.10,5.11) lorsqu'il est affecté à p) **Faire**
17: *Affecter* o_r *à* p
18: $listToRepair$ - $\{o_r\}$
19: *Aller à 10*
20: **Fin Si**
21: LP - $\{p\}$
22: **Fin Tant que**
23: **Si** ($LP = \emptyset$ and o_r is not fixed) **Alors**
24: *Returner le chromosome initial*
25: **Fin Si**
26: **Fin Tant que**
27: *Retourner le nouveau chromosome*

Dans l'exemple précédent, lors de cette 2^{eme} étape, nous parcourons aléatoirement $listToRepair$. Supposons que le premier objet renvoyé aléatoirement soit $o_{0,1}$. Les emplacements potentiels pour cet objet sont $LP = \{0, 2, 3\}$, c'est-à-dire tous les autres emplacements quelle qu'en soit la capacité. Nous recherchons aléatoirement un emplacement dans cette liste pour réparer $o_{0,1}$. Soit $p = 3$ l'emplacement trouvé. Cet emplacement correspond à ss_2. Il ne peut pas stocker $o_{0,1}$ car il n'a pas de capacité suffisante. Ainsi, $p = 3$ sera supprimé de LP qui ne contiendra plus que $\{0, 2\}$. Nous répétons le processus en sélectionnant aléatoirement un emplacement dans LP. Supposons

que $p = 0$ soit récupéré ensuite, il peut stocker $o_{1,1}$ car il a une capacité suffisante. Ainsi, cet objet est réparé et sera placé à l'emplacement $p = 0$. A ce stade, $x = (1, 3, 1, 0, ?, 0, 0, 3, ?, 2)$

Nous supprimons $o_{0,1}$ de $listToRepair$. Nous répétons le même processus pour les objets restants dans $listToRepair$. Comme le montre la figure 5.5, le $x = (1, 3, 1, 0, 0, 0, 0, 3, 2, 2)$ représente une version réparée possible.

La complexité de la fonction de réparation est égale à $\mathcal{O}(N * P)$ où N est le nombre total d'objets et P est le nombre d'emplacements. Pour la première étape, dans le pire cas, les affectations de tous les objets violent toutes les contraintes. Ainsi, les N objets seront dans $listToRepair$. Dans la deuxième étape, le pire des cas se produit lorsque l'objet est affecté au dernier emplacement trouvé dans la liste ou dans le cas où nous ne trouvons pas d'emplacement pour le réparer après $P - 1$ tentatives.

Cette fonction de réparation a été intégrée dans l'algorithme NSGAII juste avant l'évaluation des individus de la population.

La section suivante présente les expérimentations menées dans le but de valider notre approche d'optimisation.

5.5 Évaluation des performances

Dans cette section, nous présentons d'abord la méthodologie utilisée pour évaluer notre approche de placement de données. Ensuite, nous décrivons le cadre des expérimentations en termes de CSP et de caractéristiques des clients utilisés pour évaluer l'efficacité de l'approche proposée. Nous finissons cette section par présenter et discuter des résultats d'expérimentations obtenus.

5.5.1 Méthodologie d'évaluation

Nous avons utilisé l'implantation de la méta-heuristique NSGAII fournie par le cadre MOEA [193]. Nous l'avons étendu à une matheuristique (CDP-NSGAII$_{IR}$) en intégrant une fonction d'injection et une fonction de réparation.

Afin d'évaluer la matheuristique de placement de données CDP-NSGAII$_{IR}$, nous suivons la méthodologie illustrée dans la figure 5.6.

Nous exécutons des expérimentations en faisant varier le nombre d'objets N et le nombre d'emplacements P, c'est-à-dire, le nombre de classes de stockage locales plus le nombre de CSP dans la fédération. Ces deux paramètres définissent la taille de l'espace de recherche dans notre travail. Ils permettent de tester la scalabilité de la matheuristique conçue. Pour les charges de travail des clients, nous avons expérimenté différentes configurations. Il faut souligner que nous avons considéré que le CSP local a deux classes de stockage différentes. Chaque expérience a été réalisée 10 fois.

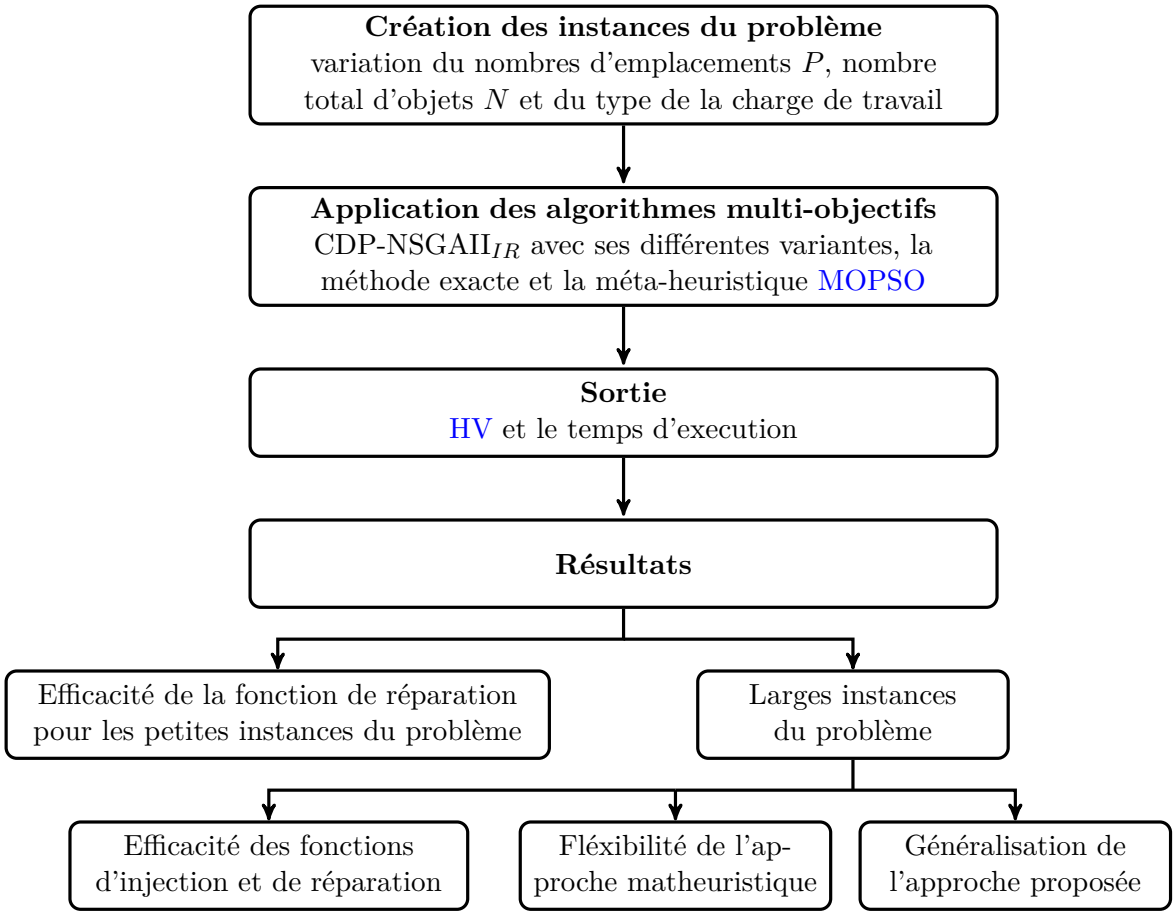


FIGURE 5.6 – Méthodologie d'évaluation

Deux métriques sont considérées pour l'évaluation de notre approche : (i) la qualité des ensembles de Pareto mesurée par le HV. Nous rappelons que le HV est un indicateur consistant à calculer le volume dominé par l'ensemble des valeurs objectives des solutions trouvées. Des valeurs HV plus élevées reflètent de bons fronts de Pareto. Cet indicateur de performance des algorithmes multi-objectifs permet de mesurer à la fois l'exhaustivité, la précision et la diversité de l'ensemble des solutions obtenues. (ii) le temps d'exécution nécessaire pour calculer le placement des objets des clients. Cette mesure permet de vérifier si l'algorithme s'exécute en un temps raisonnable ou non.

Nous évaluons nos algorithmes pour les petites et les larges instances de problème. Dans notre évaluation, une instance est considérée comme petite si elle peut être évaluée en une heure en utilisant la méthode exacte par énumération. Nous comparons les résultats de notre contribution avec la version standard de NSGAII et une version multi-objectifs d'optimisation de type essaim de particules (Particle Swarm Optimization : PSO) utilisée dans [165]. Il faut noter que les méta-heuristiques standard NSGAII et PSO sont mises en œuvre avec la technique de pénalisation.

En effet, ces méta-heuristiques gardent les solutions infaisables lors de l'opération de la sélection. Les solutions infaisables sont ordonnés selon le degré de la satisfaction des contraintes.

Pour les petites instances de problème, nous montrons la pertinence de la fonction de réparation. Pour ce faire, nous comparons notre approche en intégrant uniquement la fonction de réparation (appelée CDP-NSGAII$_R$) avec NSGAII, PSO et une méthode d'énumération exhaustive fournissant le front de Pareto exact, en termes à la fois de HV et de temps d'exécution.

Pour les larges instances de problème, nous commençons d'abord par montrer la pertinence de nos algorithmes en comparant les différentes versions de notre approches, la méta-heuristique PSO et l'ensemble des solutions obtenues par CPLEX. Le but est de voir l'impact des fonctions d'injection et de réparation en termes de HV et de temps d'exécution. Les différentes versions de notre approche sont :

— CDP-NSGAII (version standard de NSGAII adaptée à notre problème),

— CDP-NSGAII$_R$ (NSGAII avec la fonction de réparation),

— CDP-NSGAII$_I$ (NSGAII avec la fonction d'injection),

— CDP-NSGAII$_{IR}$ (NSGAII avec les fonctions d'injection et de réparation)

L'ensemble de solutions de CPLEX est composé de 10 solutions obtenues à l'aide des coefficients du tableau 5.4.

Une fois la pertinence validée, nous généralisons dans une deuxième phase notre approche matheuristique de placement proposée à d'autres méta-heuristiques multi-objectif. Pour ce faire, nous appliquons les fonctions de réparation et d'injection aux méta-heuristiques NSGAII et PSO et les évaluons avec et sans ces deux opérateurs. Les valeurs HV cumulées des différents cas d'utilisation testés sont utilisées comme indicateur qualitatif de comparaison.

Enfin, nous montrons la flexibilité de notre approche. Nous montrons comment l'approche proposée permet de faire un compromis entre la qualité et le temps d'exécution de l'algorithme. Pour cela, nous calculons le HV et le temps d'exécution de CPLEX et CDP-NSGAII$_{IR}$ en variant le nombre de solutions calculées par CPLEX et utilisées dans l'étape d'injection de la matheuristique. Nous varions les coefficients dans la plage [0..1] avec un pas égal à 0,05 pour chaque fonction objectif, ce qui donne 231 combinaisons de coefficients différentes. Nous constituons 11 ensembles de solutions différentes. Le premier contient 21 solutions, le second contient 42 solutions et ainsi de suite jusqu'au 11ème ensemble qui contient toutes les (231) solutions. A chaque fois, nous injectons un ensemble de solutions, nous calculons le HV et le temps d'exécution de CDP-NSGAII$_{IR}$ et CPLEX.

Nous utilisons le modèle de coût présenté dans le chapitre 4 pour l'évaluation des fonctions objectif. Il est à noter que quand les coûts deviennent élevés pour l'une des fonctions objectifs, par exemple, le soft SLA d'E/S n'étant pas satisfait (ce qui entraîne une augmentation de l'objectif *Store*), le HV est également dégradée. Cependant, il n'y a pas de bijection entre la valeur de

HV et un critère particulier tel que la violation du SLA. Autrement dit, si le HV est impactée, alors on ne peut pas savoir quel coût provoque un tel impact. Le HV garantit principalement l'existence de solutions réalisables.

5.5.2 Paramètres expérimentaux

Les expériences ont été conçues sur la base du scénario suivant. Une fédération de *Clouds* contient un ensemble de CSP, chacun composé d'un seul centre de données (pour plus de simplicité). Chaque CSP est situé dans une zone géographique différente. La latence du réseau entre chaque paire de zones est attribuée de manière aléatoire dans un intervalle]100ms, 700ms]. Le CSP considéré gère un ensemble de clients, chacun ayant un objet de données (encore une fois pour plus de simplicité). Le CSP essaie d'optimiser le placement des objets de données de ses clients internes afin de trouver un ensemble de solutions pertinentes. Il héberge un système de stockage hybride composé d'un ensemble de 20 HDD et 20 SSD ayant respectivement les caractéristiques de 1 To Hitachi Deskstar 7K1000.D modèle HDS721010DLE630 7200 RPM HDD [194] et 1 To Samsung 850 PRO SSD [195].

Dans ce travail, nous considérons que le SLA des clients est lié aux métriques de performances de stockage. Si les performances ne sont pas atteintes, une pénalité représentée par un pourcentage du montant total des frais est déduite de la facture du client. Dans notre évaluation, cette pénalité a été fixée à 10% du montant total payé par le client.

Comme dans [23, 189], nous supposons que la latence du réseau des requêtes des clients est facturée. Le prix de la latence est donné dans le tableau 5.1 et sa pénalité associée dans le tableau 5.2.

Tableau 5.1 – Prix de latence/req.

Latency (ms)	Prix par requête ($)
< 100	0.00001
< 300	0.000008
> 300	0

Tableau 5.2 – Penalité de latence/req.

Latence (ms)	Penalilé ($)
< 100	0
< 300	0.000002
> 300	0.00001

Nous avons utilisé la configuration suivante pour les spécifications des centres de données

Tableau 5.3 – Prix et performances du stockage

	Valeur Minimume	Valeure Maximume
Coût d'occupation	0.025 $	0.225 $
Coût d'IOPS	0.055 $	0.075 $
Coût de réseau	0.08 $	0.1 $

(DC), les charges de travail des clients et les paramètres des évaluations.

Spécifications des DC : Les différents CSP de la fédération proposent des services de stockage hétérogènes en termes de prix et de performances. De plus, seul le coût du réseau sortant est pris en compte par les CSP lors de la migration des objets. Nous supposons, comme dans le service de stockage Amazon EBS io1 [196], que le prix du stockage est lié à la capacité, aux performances et à la durée du service. En effet, le prix d'EBS io1 est fixé à 0,125$/GB-mois pour l'occupation et à 0,065$/IOPS-mois [74]. Aussi, dans Amazon, le prix du réseau est de 0,09$Go [197]. Afin d'attribuer aux CSP de la fédération différentes caractéristiques en termes de performances de stockage et de coûts de services, nous avons sélectionné des valeurs parmi les plages spécifiées dans le tableau 5.3 suivant une distribution uniforme comme dans [19, 198, 148]. Dans notre évaluation, les coûts EBS et du réseau d'Amazon ont été définis en tant que valeurs médianes de l'intervalle utilisé. Pour les performances de stockage, la valeur maximale d'IOPS pouvant être fournie par chaque CSP a été définie de manière aléatoire dans [250, 64 000]. Ces limites d'intervalle reflètent les IOPS minimales et maximales pouvant être fournies par les disques EBS d'Amazon. Ce faisant, les différents CSP auront des caractéristiques différentes en termes de coûts et de performances des services de stockage.

Comme dans une fédération, les prix des services peuvent évoluer dans le temps [18] pour favoriser la fédération, on suppose que les prix d'occupation des stockages évoluent en fonction du taux d'occupation comme dans [18, 19]. Dans notre travail, nous supposons que le taux d'occupation est corrélé à la période de la journée car la charge générée a un cycle quotidien relatif aux heures de pointe [18]. Nous supposons qu'il suit une distribution normale ($\mathcal{N}(12, 6)$) avec une moyenne égale à 12 heures et un écart-type de 6 heures. Ainsi, chaque CSP fixe des prix de stockage pondérés selon cette distribution comme dans [19].

Workloads : nous avons expérimenté avec le benchmark YCSB [199]. Nous avons utilisé différentes configurations en termes de ratio de requêtes de lecture, de mise à jour, d'analyse et d'insertion. Nous avons varié le nombre de types de requêtes par client. Nous avons également varié le nombre de requêtes soumises par heure.

Deux charges de travail ont été évaluées dans cette étude. La charge de travail 1 avec 100% de lecture et la charge de travail 2 avec 50% de lecture et 50% de mise à jour.

Les charges de travail génèrent différents modèles d'opérations d'E/S que nous avons capturés à l'aide de l'outil Blktrace [200]. Comme pour le taux d'occupation, le nombre de requêtes par client a été pondéré selon la distribution normale précitée. Comme dans [189], la charge de travail a été définie en fonction du nombre de clients en ligne qui évolue selon une distribution normale.

Coefficients de la méthode exacte : Les coefficients du tableau 5.4 ont été choisis pour calculer les solutions avec la méthode exacte pour construire la fonction objectif comme c'est la somme pondérée des trois coûts de stockage, migration et latence du réseau ($\alpha * Store_N + \beta *$

Tableau 5.4 – Coefficients de la méthode exacte

α	1	0	0	0.5	0.5	0	0.6	0.2	0.2	0.4
β	0	1	0	0.5	0	0.5	0.2	0.6	0.2	0.3
γ	0	0	1	0	0.5	0.5	0.2	0.2	0.6	0.3

Paramètre	Valeur
Taille de la population	500
Nombre de génération	100
Taux de croisement	0.8
Taux de mutation	0.1

Tableau 5.5 – Paramètres de NSGAII

$Migrate_N + \gamma * Latency_N$). Ainsi, 10 solutions peuvent être calculées selon les coefficients du tableau 5.4, chaque colonne correspond à une solution. Les 3 premières correspondent au calcul de la solution optimale pour seulement un des objectifs. Comme CPLEX ne peut pas obtenir de solutions optimales dans un délai raisonnable pour certaines instances, nous avons limité son temps d'exécution à 60 secondes. Ce choix a été fait compte tenu du temps d'exécution de la méta-heuristique qui est de quelques secondes (environs 10 secondes pour les plus larges instances évaluées).

Paramètres de NSGAII : Dans les résultats, les paramètres de NSGAII sont indiqués dans le tableau 5.5.

5.5.3 Résultats

Dans cette section, nous allons d'abord évaluer l'efficacité en termes de HV et de temps d'exécution de la fonction de réparation pour les petites instances du problème. Ensuite, nous passons aux larges instances du problèmes. Pour ce type d'instances, nous évaluons dans un premier lieu l'efficacité de la matheuristique proposée (fonctions d'injection et de réparation). Après cela, nous montrons les résultats de la généralisation de l'approche proposée sur d'autres méta-heuristiques. Enfin, nous évaluons la flexibilité de l'approche proposée en donnant différents compromis entre le temps d'exécution et le HV en fonction du nombre de solutions calculées par la méthode exacte.

5.5.3.1 Petites instances du problème : efficacité de la fonction de réparation

L'objectif de cette évaluation est d'étudier l'efficacité de la fonction de réparation pour des problèmes de petites instances en terme de HV et de temps d'exécution.

Tableau 5.6 – HV et temps d'exécution pour des petites instances

Objets	10		13		15	
Algorithmes	HV	Temps (s)	HV	Temps (s)	HV	Temps (s)
Méthode exacte	0.269	2.4	0.311	154	0.131	2886
CDP-NSGAII$_R$	0.269	0.3	0.310	0.3	0.129	0.3
NSGAII	0.269	0.8	0.311	0.6	0.131	0.6
PSO	0.269	0.2	0.310	0.3	0.130	0.2

Pour cela, nous avons calculé le HV et le temps d'exécution pour 3 petites instances du problème qui peuvent être traitées par la méthode exacte en une heure. Pour ces instances, nous n'avons considéré que 3 CSP dans la fédération, 2 classes de stockage locales qui constituent 4 emplacements différents pour stocker les objets. L'expérience a été répétée avec 10, 13 et 15 objets. Les résultats sont présentés dans le tableau 5.6.

Comme on peut le voir dans le tableau 5.6, le temps d'exécution de la méthode exacte subit une croissance exponentielle. Cela prend environ 48 minutes pour seulement 15 objets et 4 emplacements. Ainsi, par extrapolation, il faudrait environ 8 ans pour évaluer un problème avec seulement 5 emplacements et 20 objets donnant 5^{20} de solutions possibles.

La méthode exacte calcule le front de Pareto exact, et donc l'hypervolume exact. Tous les autres algorithmes évalués obtiennent plus ou moins le même HV (différence entre 0 et 1.5%) en peu de temps (moins de 1 seconde). Par conséquent, contrairement aux algorithmes évolutionnaires évalués, la méthode exacte est inutilisable pour les larges instances ni hors ligne ni en ligne, en raison de la nature combinatoire du problème de placement.

Pour les algorithmes évolutionnaires, NSGAII donne le HV exact mais double le temps requis par CDP-NSGAII$_R$ et PSO pour une amélioration de HV non significative. PSO a le temps d'exécution le plus petit avec une erreur de HV qui ne dépasse pas 0,7%.

Il faut noter que comme les les méta-heuristiques obtiennent le résultat optimal, nous n'avons pas évalué les versions de notre approche avec l'opérateur d'injection. En effet, c'est triviale de conclure que NSGAII avec l'opérateur d'injection prennent plus de temps NSGAII avec et sans l'opérateur de réparation pour le même résultats car les versions avec l'opérateur d'injection nécessitent une étape supplémentaire pour le calcul des solutions initiales avec CPLEX.

Le reste des évaluations concernes les larges instances du problème. Comme ces instances ne peuvent pas être résolues dans un temps raisonnable en utilisant un algorithme par énumération, nous excluons cette méthode de la comparaison.

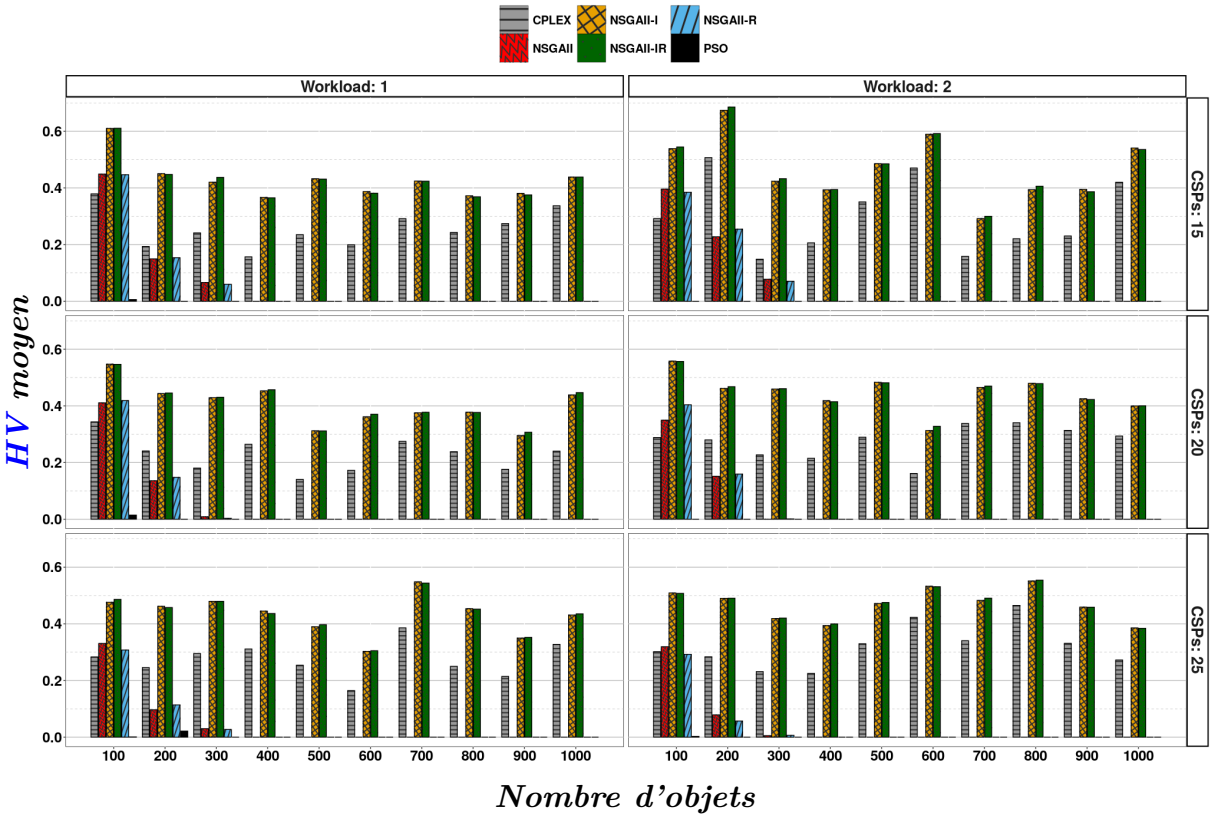


FIGURE 5.7 – Comparaison des hypervolumes pour les larges instances du problème

5.5.3.2 Larges instances du problème

Efficacité des fonctions d'injection et de réparation : la figure 5.7 montre, pour chaque algorithme, l'hypervolume dérivé de son front Pareto résultant en fonction du nombre d'objets, du nombre de CSP et du type de charge de travail. A partir de cette figure, nous remarquons que CDP-NSGAII$_{IR}$ et CDP-NSGAII$_{I}$ donnent toujours les meilleurs résultats en terme de HV. CDP-NSGAII$_{IR}$ améliore le HV de CPLEX de 15% à 60% et la méta-heuristique NSGAII de 4% à 94%. Alors que CDP-NSGAII$_{I}$ améliore le HV de CPLEX de 14% à 60% et la méta-heuristique NSGAII de 4% à 94%.

Dans la plupart des cas évalués, les méta-heuristiques standards NSGAII et PSO, en particulier PSO conduisent à un front presque vide avec un HV négligeable. En moyenne, NSGAII et PSO obtiennent respectivement 21.3% et 2.72% du HV obtenu par CDP-NSGAII$_{IR}$. Ces résultats s'expliquent par les limitations des méta-heuristiques standard lorsque l'espace de recherche augmente et que les solutions valides deviennent proportionnellement rares.

De plus, nous remarquons également que dans les cas où NSGAII et PSO pourraient trouver

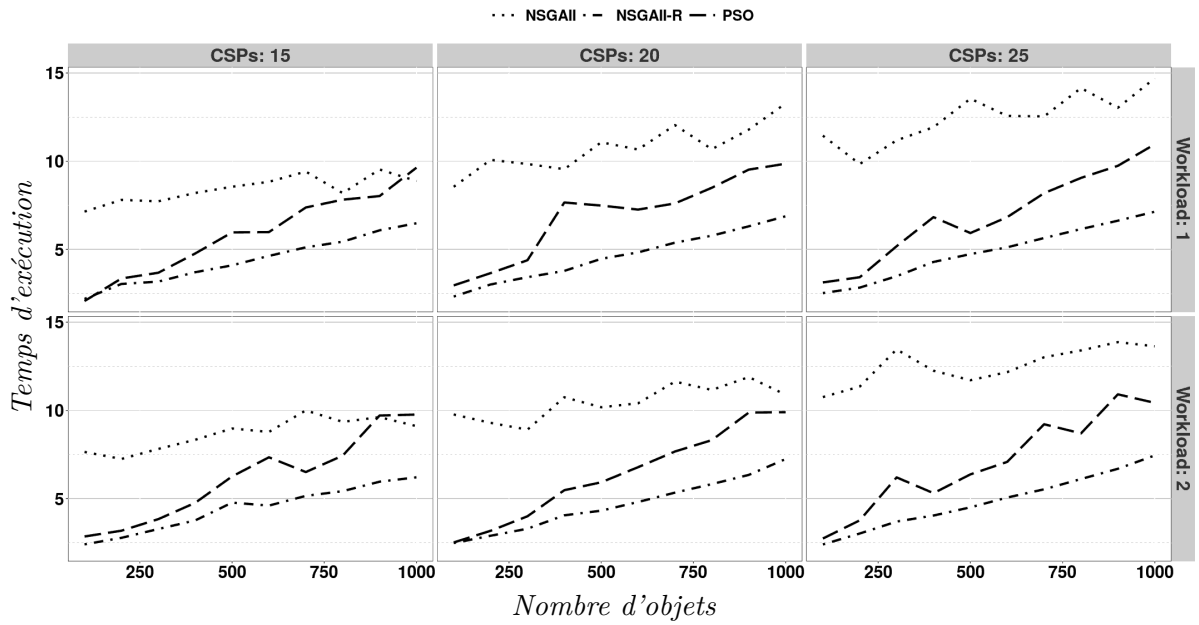


FIGURE 5.8 – Évolution du temps d'exécution

des solutions, la méta-heuristique NSGAII améliore PSO de 6% a 79% en terme de HV.

Pour CDP-NSGAII$_R$, il a à peu près le même HV que CDP-NSGAII avec une différence de -4% à +9%. Néanmoins, à partir de la figure 5.8, nous remarquons que CDP-NSGAII$_R$ améliore fortement le temps d'exécution de NSGAII. Cette amélioration est comprise entre 40% et 86%. En effet, NSGAII prend plus de temps car il doit trier les individus selon le degré de violation des contraintes (pour pénaliser les individus invalides). Comme CDP-NSGAII$_R$ répare toutes les solutions invalides, il n'effectue pas ce traitement supplémentaire. PSO permet d'aller plus vite que NSGAII mais moins vite que CDP-NSGAII$_R$.

De plus, à partir de cette figure, on voit que le temps d'exécution de CDP-NSGAII$_R$ augmente linéairement avec l'augmentation du nombre d'objets alors que le nombre de CSP impacte légèrement le temps d'exécution. Ceci est dû au fait que nous avons considéré un grand nombre d'objets par rapport au nombre de CSP ce qui est cohérent avec la complexité théorique.

Il est à noter que nous n'avons pas pris en compte le temps d'exécution des versions avec l'opérateur d'injection de notre approche comme elles dépendent aussi du temps d'exécution de CPLEX pour calculer leurs solutions initiales, le résultat n'est donc pas significatif.

En définitive, nous remarquons que l'opérateur d'injection a un grand impact sur le HV résultant tandis que l'opérateur de réparation a un grand impact sur le temps d'exécution. Nous pouvons également observer que la méta-heuristique NSGAII surpasse la méta-heuristique PSO pour le problème traité.

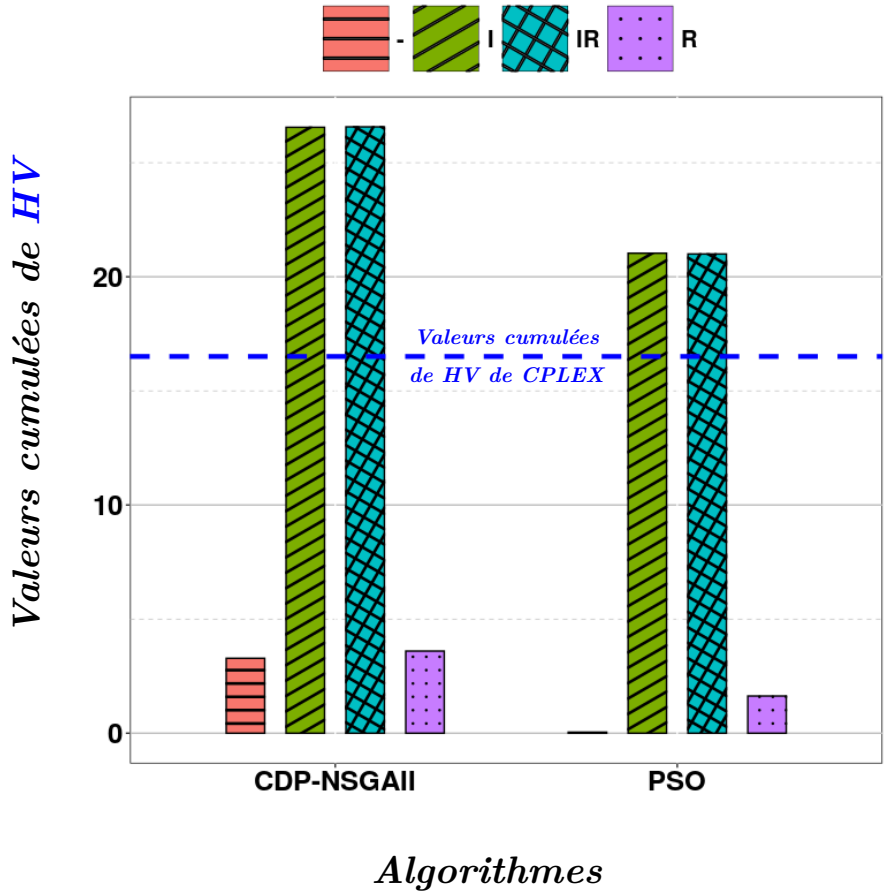


FIGURE 5.9 – Généralisation de la matheuristique

Généralisation de l'approche proposée sur la méta-heuristique PSO : l'approche matheuristique proposée dans ce travail peut être généralisée en exploitant d'autres méta-heuristiques. Dans cette évaluation, nous appliquons les fonctions de réparation et d'injection aux méta-heuristiques NSGAII et PSO et les évaluons avec et sans ces deux opérateurs. Les valeurs HV cumulées des différents cas d'utilisation testés sont utilisées comme indicateur qualitatif de comparaison.

En plus des résultats de l'approche proposée avec la méta-heuristique NSGAII, la figure 5.9 montre les résultats obtenus en la couplant avec la méta-heuristique PSO. Le symbole de légende **(-)** signifie la version standard de l'algorithme, tandis que les symboles **(R, I, IR)** signifient que les algorithmes sont mis à jour respectivement par les opérateurs de réparation, d'injection et les deux en même temps. La ligne (bleue) horizontale représente le HV cumulé de CPLEX. Il faut noter que pour cette expérience, l'ensemble de solutions de CPLEX est également composé

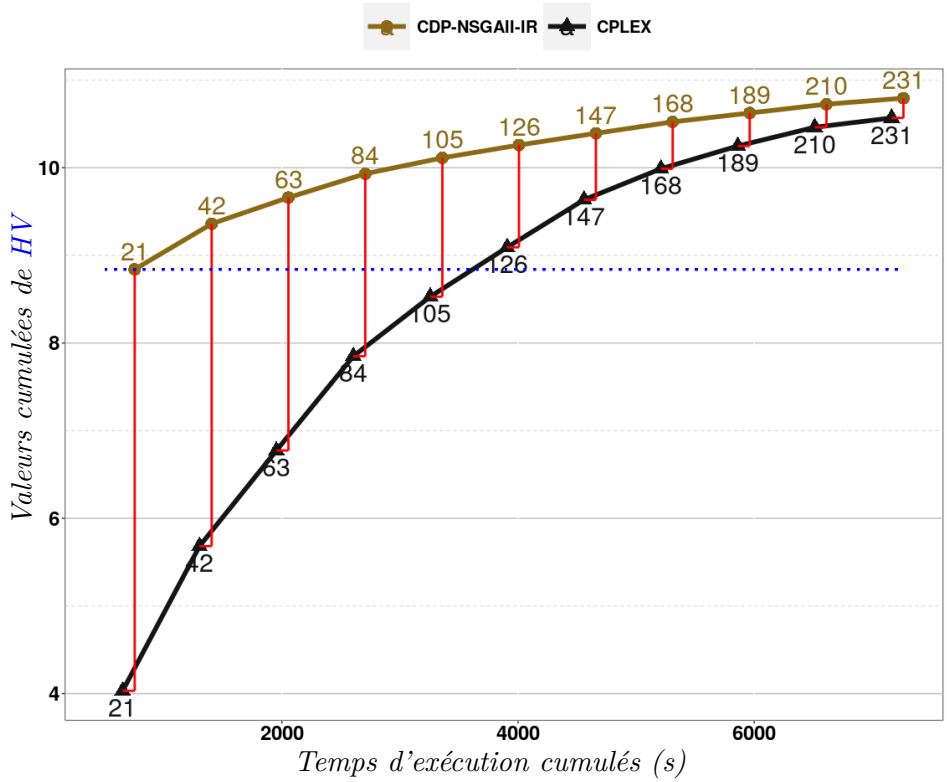


FIGURE 5.10 – Flexibilité de la matheuristique

de dix solutions calculées à l'aide des combinaisons de coefficients du tableau 5.4.

A partir de cette figure, nous remarquons que l'ensemble des solutions calculées par CPLEX surpasse les HV des versions standard de NSGAII et PSO respectivement de 46% et 57%. CPLEX surpasse également les deux méta-heuristiques augmentées par la fonction de réparation. Il surpasse NSGAII$_R$ de 44% et PSO$_R$ de 51%. Nous observons également que NSGAII et PSO mis à niveau avec les fonctions d'injection et de réparation surpassent CPLEX et toutes les autres variantes d'algorithmes. En fait, NSGAII$_{IR}$ améliore le HV du CPLEX de 35% tandis que PSO$_{IR}$ améliore CPLEX de 16%.

Flexibilité de l'approche proposée : dans cette évaluation, nous avons pris en compte à la fois les valeurs cumulées de temps d'exécution et de HV. Ceci est fait pour différentes instances du problème. Les instances du problème se composent de 15 CSP et nous avons fait varier le nombre d'objets entre 100 et 1000 objets. Nous avons considéré à la fois les charges de travail 1 et 2.

Dans la figure 5.10, les petits triangles et points correspondent respectivement au temps d'exécution et au HV de CPLEX et CDP-NSGAII$_{IR}$ et leurs étiquettes représentent les nombres

de solutions calculées par CPLEX. Tout d'abord, nous observons que CDP-NSGAII$_{IR}$ prend un peu plus de temps à s'exécuter (en moyenne, 5 secondes) car il augmente le temps de calcul du CPLEX initial qu'il utilise lui même pour calculer un sous-ensemble de sa population initiale du temps de calcul de la phase heuristique. Néanmoins, ce temps supplémentaire de CDP-NSGAII$_{IR}$ est en grande partie constant car il dépend du nombre d'évaluations de la méta-heuristique NSGAII et n'a aucun rapport avec le nombre de solutions injectées. Ce temps correspond au léger décalage vers la droite des points CDP-NSGAII$_{IR}$ sur l'axe des x par rapport aux points CPLEX (mieux observés sur le point 231).

A partir de cette figure, nous avons remarqué que le HV (cumulé) de CDP-NSGAII$_{IR}$ est toujours supérieur à celui de CPLEX pour un temps d'exécution donné. La matheuristique améliore CPLEX jusqu'à 2.2 fois (pour 21 solutions injectées dans la figure). Nous remarquons que la différence de HV est plus significative pour les petits ensembles de CPLEX et lorsque nous augmentons le nombre de solutions calculées par CPLEX, la différence de HV diminue.

Aussi, on peut observer sur cette figure qu'en traçant une ligne horizontale pour un CDP-NSGAII$_{IR}$ donné, disons 21 solutions injectées, CPLEX a besoin de plus de 100 solutions pour obtenir un HV équivalent (voir la ligne bleue). De plus, avec un tel nombre de solutions, le temps d'exécution de CPLEX est environ 10 fois plus important que CDP-NSGAII$_{IR}$.

Ainsi, réduire le nombre de solutions fournies par CPLEX est un moyen d'améliorer la scalabilité de l'approche matheuristique car lorsque la taille du problème augmente, CPLEX prend beaucoup plus de temps pour trouver les solutions exactes requises tandis que le temps d'exécution de la partie heuristique de CDP-NSGAII$_{IR}$ reste inchangé. Bien entendu, cela se fait au détriment de HV et un compromis est à trouver en fonction du contexte d'utilisation de l'algorithme, notamment si les calculs doivent se faire en ligne.

5.6 Conclusion

Pour un fournisseur de services *Cloud* membre d'une fédération, le choix d'un bon placement entre les ressources locales et les services externes fournis par d'autres CSP est un grand défi, en particulier pour les charges de travail variables dans le temps. Dans ce travail, nous avons modélisé le problème de placement des objets des clients dans une fédération de *Cloud* comme un problème d'optimisation multi-objectifs. Le modèle de placement des données prend en compte les différentes caractéristiques du système de stockage local et des services de stockage externes, la charge de travail variable des clients et leur SLA.

Pour résoudre le problème multi-objectifs proposé, nous avons conçu la matheuristique CDP-NSGAII$_{IR}$. CDP-NSGAII$_{IR}$ étend NSGAII par une étape de pré-traitement consistant à calculer un ensemble de solutions exactes qui est injecté ensuite dans la population initiale de NSGAII. De plus, une fonction de réparation est conçue afin de fixer un placement donné s'il n'est pas

réalisable dans les populations de NSGAII.

L'évaluation effectuée a prouvé l'efficacité de la matheuristique proposée. CDP-NSGAII$_{IR}$ améliore l'hypervolume de NSGAII et de CPLEX jusqu'à 94% et 60% respectivement. Le mécanisme d'injection permet d'améliorer la qualité des solutions pour une effort de calcul donné, tandis que la fonction de réparation améliore le temps d'exécution de NSGAII de 68% en moyenne.

Dans ce chapitre, le placement de données a été traité sans prendre en considération la réplication des objets des clients ce qui fera l'objet du chapitre suivant.

Chapitre 6

Optimisation multi-objectif de placement et de sélection des répliques dans un *Cloud* fédéré

Le chapitre précédent a abordé le problème de placement de données pour un CSP membre d'une fédération de *Clouds*. Le problème a été modélisé sous forme d'un problème d'optimisation multi-objectif. La matheuristique CDP-NSGAII$_{IR}$ proposée pour résoudre le problème de placement de données s'avère efficace en termes de qualité de solutions et de temps d'exécution. Néanmoins, la contribution précédente ne prend pas en considération la réplication de données qui est indispensable pour l'augmentation de la disponibilité de données, la réduction des coûts d'accès aux données et l'assurance d'une meilleure tolérance aux pannes. La réplication consiste à créer plusieurs copies identiques de la donnée. Une bonne stratégie de réplication doit répondre à quatre questions fondamentales : (i) combien de répliques doivent être générées pour fournir la disponibilité nécessaire du système ? (ii) comment synchroniser les répliques afin d'assurer leur cohérence ? (iii) où stocker les répliques ? et (iv) quelles répliques doivent être sélectionnées pour accéder rapidement aux données requises ? Bien que les deux premiers défis sont principalement liés aux exigences des applications, les deux derniers sont critiques et doivent être résolus en ligne pour permettre un placement et une sélection dynamique des données dans un environnement variable d'une fédération de *Clouds* .

Dans ce chapitre, nous augmentons la contribution introduite dans le chapitre précédent en incluant la réplication. Ainsi, le problème de placement et de sélection des répliques est modélisé sous la forme d'un MOOP ayant comme objectif la minimisation des coûts de stockage, de migration et de latence. Le modèle prend en compte l'hétérogénéité des systèmes de stockage locaux et fédérés, les charges de travail des clients et leur QoS requise. Afin d'apporter au CSP des solutions intéressantes, nous proposons $StorNIR$, une matheuristique combinant une méthode exacte avec une méta-heuristique. En effet, $StorNIR$ reprend le principe de fonctionnement de CDP-NSGAII$_{IR}$. Son principe consiste à produire un ensemble de solutions à l'aide d'un solveur MILP. Ces solutions sont ensuite injectées dans la population initiale d'une métaheuristique évolutionnaire. Dans ce travail, nous utilisons NSGAII. Le modèle de coût proposé dans le chapitre

4 est utilisé pour évaluer les fonctions objectifs. En revanche, l'encodage ainsi que les opérateurs d'évolution ont été adaptés pour prendre en considération la réplication des objets. La fonction de réparation devient également plus complexe car le nombre de choix des emplacements pour les objets augmente vu qu'ils sont répliqués. La fonction de migration est mise a jour en migrant la copie placée dans le CSP ayant le coût de transfert réseau le moins important. En outre, comme les objets sont répliqués, la charge de travail des clients doit être soigneusement affectée aux répliques afin de sélectionner celles qui vérifient à la fois les contraintes qui correspondent à la charge d'E/S de stockage, qui réduisent la latence du réseau et enfin qui diminuent le coût de l'exécution des opérations d'E/S.

Ce travail a fait l'objet d'une publication dans la conférence internationale *Annual ACM Symposium on Applied Computing (SAC)* [201].

Dans les sections suivantes, nous rappelons la problématique adressée dans cette contribution, nous décrivons les différentes étapes de la conception de la stratégie proposée, et enfin, nous présentons les expérimentations et analysons les résultats obtenus.

Sommaire

6.1 Problématique

Les services de stockage dans le *Cloud* sont devenus une technologie prometteuse pour externaliser les données et les traitements associés. La réplication de données est considérée comme l'un des principaux mécanismes des services de stockage permettant d'améliorer les performances en termes de disponibilité, de robustesse et de temps de d'accès aux données [21, 202].

Pour un CSP, membre d'une fédération, un placement et une sélection efficaces des répliques des objets des clients sont importants pour satisfaire les demandes de QoS tout en minimisant ses coûts. Pour optimiser le problème de placement et de sélection des répliques d'objets dans un *Cloud* fédéré, le coût de placement qui se compose des coûts de stockage, de migration et de latence, doit être pris en compte. Ces coûts sont contradictoires dans certains cas. De plus, comme mentionné auparavant, un *Cloud* faisant partie d'une fédération est un environnement hétérogène et dynamique. En effet, un CSP essaie d'utiliser ses propres ressources et celles des autres membres de la fédération pour satisfaire les SLA de ses clients. Ces ressources sont hétérogènes et volatiles. L'hétérogénéité des ressources de stockage implique une plus grande complexité pour considérer les différents dispositifs, tandis que la volatilité implique une forte variabilité des ressources disponibles.

La problématique adressée dans ce chapitre comporte les deux aspects suivants : **(i) comment placer les répliques ? et (ii) comment choisir la réplique à accéder pour un client donné ?** afin de satisfaire la QoS des clients tout en optimisant les coûts de placement pour le CSP.

6.2 Formulation du problème

Pour optimiser le placement des répliques des objets des clients dans un *Cloud* fédéré, la minimisation du coût de placement est l'objectif pris en compte. Nous rappelons qu'il se compose des coûts de stockage, de migration et de latence du réseau. Comme discuté précédemment, ces coûts sont dans certains cas contradictoires. Ainsi, une formulation multi-objectifs est recommandée pour modéliser ce problème de placement et offrir un ensemble de solutions au décideur qui pourra choisir une solution finale en fonction des métriques modélisées et de facteurs externes telle que la réputation des CSP. Le modèle de coût proposé dans le chapitre 4 est utilisé pour construire les fonctions objectifs.

Dans ce travail, nous supposons que chaque objet possède 3 répliques comme dans HDFS [203]. Cependant, notre modèle permet de faire varier ce paramètre. L'objectif principal est de trouver une configuration de placement de répliques qui répond aux exigences des clients et minimise le coût global de placement. Ce problème est soumis à un ensemble de contraintes liées aux ressources limitées, au nombre de répliques d'objets et aux SLA des clients.

Ainsi, nous définissons le problème de placement et de sélection de répliques comme un

problème d'optimisation multi-objectifs dont le but est de minimiser le coût de stockage (*Store*), le coût de migration (*Migrate*) et le coût de latence du réseau (*Latency*) (voir la section 5.3.1 pour la définition des fonctions objectifs) comme suit :

$$\min \begin{bmatrix} Store \\ Migrate \\ Latency \end{bmatrix} \tag{6.1a}$$

$$S.T. \quad \sum_{u_k} \sum_{o_{i,k} \in sc_j} S_{o_{i,k}} \leq csc_j \qquad \forall j < J \tag{6.1b}$$

$$\sum_{u_k} \sum_{o_{i,k} \in ss_d} S_{o_{i,k}}] \leq css_d \qquad \forall d \geq J \tag{6.1c}$$

$$\sum_{op \in OP} \frac{\sum_{u_k} \sum_{o_{i,k} \in sc_j} io_{o_{o_{i,k}}}(op)}{io_j(op)} \leq 1, \qquad \forall j < J \tag{6.1d}$$

$$\sum_{op \in OP} \frac{\sum_{u_k} \sum_{o_{i,k} \in ss_d} io_{o_{o_{i,k}}}(op)}{io_d} \leq 1, \qquad \forall d \geq J \tag{6.1e}$$

$$io\text{-}offered_k >= ioHard_k \qquad \forall u_k \in U \tag{6.1f}$$

$$\sum_{sc_j} (o_{i,k} \in sc_j) + \sum_{ss_d} (o_{i,k} \in ss_d) = rep_{o_{i,k}}, \forall o_{i,k} \in O \tag{6.1g}$$

$$\sum_{sc_j} (o_{i,k} \in sc_j) = 1, \qquad \forall o_{i,k} \in O \tag{6.1h}$$

L'équation (6.1a) représente la fonction objectif. Les deux premières contraintes (équation (6.1b) et équation (6.1c)) expriment la limitation des capacités des ressources de stockage internes et externes. L'équation (6.1d) et l'équation (6.1e) indiquent que ces ressources de stockage ont une performance finie. L'équation (6.1f) garantit que le SLA (stockage) doit être limité. Enfin, l'équation (6.1g) indique qu'il existe un nombre de répliques $rep_{o_{i,k}}$ pour l'objet $o_{i,k}$ et l'équation (6.1h) garantit qu'une seule réplique est stockée localement par objet. Ces deux dernières équations font la différence avec le travail présenté dans le Chapitre 5. En outre, le calcul du coût de migration a été mis à jour. En effet, lors de la migration d'un objet vers un nouvel emplacement, on déplace sa copie placée dans l'emplacement avec le plus faible coût de transfert réseau. De plus, comme nous avons imposé qu'une copie de chaque objet doit être stockée localement, le coût de rapatriement est nul.

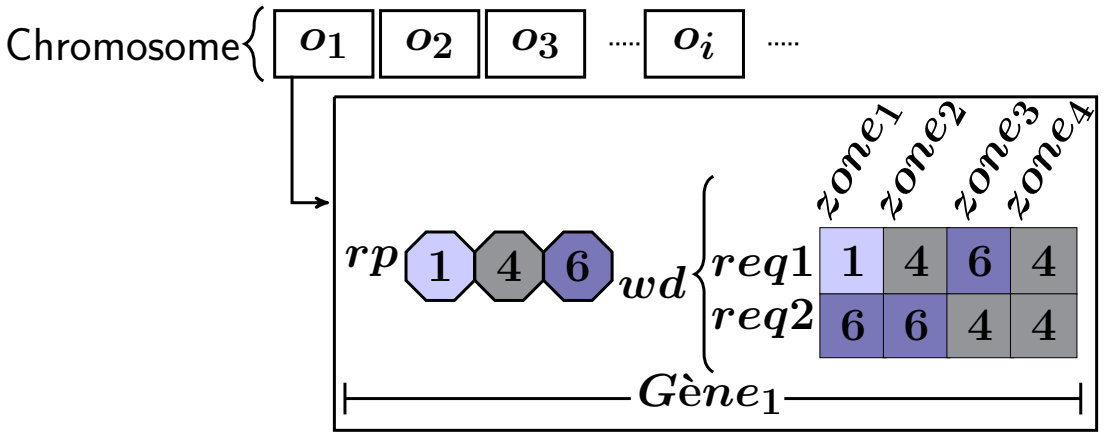


FIGURE 6.1 – Encodage des solutions

6.3 *StorNIR* une approche de placement et de sélection des répliques

NSGAII constitue l'un des MOEA les plus populaires [122, 123, 124]. Cette méta-heuristique se distingue par sa complexité polynomiale qui vaut $\mathcal{O}(MN^2)$ où M est le nombre d'objectifs et N est la taille de la population. Cependant, appliqué à notre problème, NSGAII souffre de problèmes liés à l'infaisabilité des solutions lorsque la taille de l'instance du problème augmente. Pour résoudre ce problème, nous avons conçu ***StorNIR***, qui met à niveau NSGAII vers une matheuristique en l'hybridant avec une méthode exacte qui calcule certaines solutions pour la population initiale. De plus, comme les opérateurs de réparation sont également connus pour résoudre le problème d'infaisabilité, nous avons conçu une fonction de réparation adaptée à notre problème.

Nous présentons dans ce qui suit le schéma d'encodage et les opérateurs évolutifs. Ensuite, la matheuristique ***StorNIR*** est décrite.

6.3.1 Encodage

Les chromosomes ou les solutions doivent représenter le placement et la sélection des répliques pour chaque objet. Par placement de réplique, nous signifions à quel endroit une réplique serait placée tandis que la sélection de réplique signifie quelles répliques d'objet seraient choisies pour satisfaire les requêtes des clients. Chaque gène d'un chromosome représente les caractéristiques de l'objet. Il est codé en deux vecteurs qui sont appelés ***rp*** et ***wd***. Le vecteur ***rp*** est utilisé pour le placement des répliques et ***wd*** est utilisé pour la sélection des répliques pour l'exécution de la charge de travail des clients.

Les valeurs ***rp*** appartiennent à l'intervalle $[\mathbf{0}, \boldsymbol{D} + \boldsymbol{J} - \mathbf{1}[$ puisque le nombre total d'emplacements est égal au nombre de classes de stockage locales $\boldsymbol{J}$ plus le nombre de CSP partenaires

$(\boldsymbol{D-1})$. La figure 6.1 montre un exemple de gène ($\boldsymbol{Gène_1}$) représentant l'objet $\boldsymbol{o_1}$ qui a trois répliques placées dans les emplacements : $\boldsymbol{p=1}$, $\boldsymbol{p=4}$ et $\boldsymbol{p=6}$.

$\boldsymbol{wd}$ est un vecteur 2D. Chaque ligne correspond à la répartition de la charge de travail des requêtes d'un type donné provenant de différentes zones et ciblant les différentes répliques. Ainsi, les valeurs $\boldsymbol{wd}$ doivent appartenir à $\boldsymbol{rp}$ car la charge de travail doit être exécutée sur une réplique existante. La figure 6.1 montre un exemple simple où nous supposons qu'il y a deux types de requête (par exemple, la requête *get* et la requête *update*). $\boldsymbol{wd}$ de $\boldsymbol{o_1}$ montre que la charge de travail de type $\boldsymbol{req1}$ provenant de $\boldsymbol{zone_1}$ est exécutée sur la première réplique placée à l'emplacement $\boldsymbol{p=1}$. De plus, la charge de travail de type $\boldsymbol{req1}$ provenant de $\boldsymbol{zone_2}$ est exécutée sur la réplique $\boldsymbol{2^{nd}}$ placée dans $\boldsymbol{p=4}$, et ainsi de suite.

6.3.2 Opérateurs d'évolution

La phase de reproduction des algorithmes génétiques repose généralement sur l'utilisation d'opérateurs de croisement et de mutation. Dans ce qui suit, les opérateurs utilisés sont expliqués.

6.3.2.1 Croisement

Pour cet opérateur, nous avons adopté l'opérateur de croisement populaire SBX [192]. Les opérateurs de croisement sont classés en deux catégories [204] (i) les opérateurs fonctionnant sur les variables où chaque gène des solutions parentes est recombiné indépendamment avec une probabilité pré-spécifiée pour créer de nouvelles valeurs, et (ii) les opérateurs fonctionnant sur les vecteurs effectuant généralement une combinaison linéaire des vecteurs. SBX est un opérateur qui fonctionne sur les variables. Dans notre travail, les vecteurs $\boldsymbol{rp}$ de la nouvelle population sont générés à l'aide du SBX tandis que les vecteurs $\boldsymbol{wd}$ sont conservés et mis à jour avec les nouvelles valeurs des nouveaux vecteurs $\boldsymbol{rp}$ de la nouvelle population. En effet, quand $\boldsymbol{rp[i]}$ change, toutes les valeurs du vecteur $\boldsymbol{wd}$ qui étaient égales à l'ancienne valeur de $\boldsymbol{rp[i]}$ valent sa nouvelle valeur. Nous avons suivi cette stratégie afin d'affecter la charge de travail à un emplacement où il y a une copie de l'objet voulu car si on applique le SBX sur les vecteurs $\boldsymbol{wd}$, il se peut que des nouvelles valeurs générées n'appartiennent pas aux valeurs du vecteur $\boldsymbol{rp}$. En faisant ainsi, nous assurons que la charge de travail des clients est affectée à des emplacements où il existe des copies.

La figure 6.2 montre un exemple de croisement de deux gènes parents $\boldsymbol{Gène_1}$ et $\boldsymbol{Gène_2}$. Le croisement de ces deux gènes génère les deux enfants $\boldsymbol{Enfant_1}$ et $\boldsymbol{Enfant_2}$ en appliquant le SBX sur les vecteurs $\boldsymbol{rp}$ des parents et en mettant à jours les vecteurs $\boldsymbol{wd}$ avec les nouvelles valeurs des vecteurs $\boldsymbol{rp}$ résultants. Pour $\boldsymbol{Gène_1}$, par exemple, la charge de travail provenant de $\boldsymbol{zone_3}$ est exécutée sur la $3^{\text{ème}}$ réplique placée dans l'emplacement $\boldsymbol{p=6}$. Pour l'enfant $\boldsymbol{Enfant_1}$ la charge de travail provenant de cette zone est exécutée sur la $3^{\text{ème}}$ réplique qui est, dans ce cas, placée dans l'emplacement $\boldsymbol{p=5}$ après le croisement.

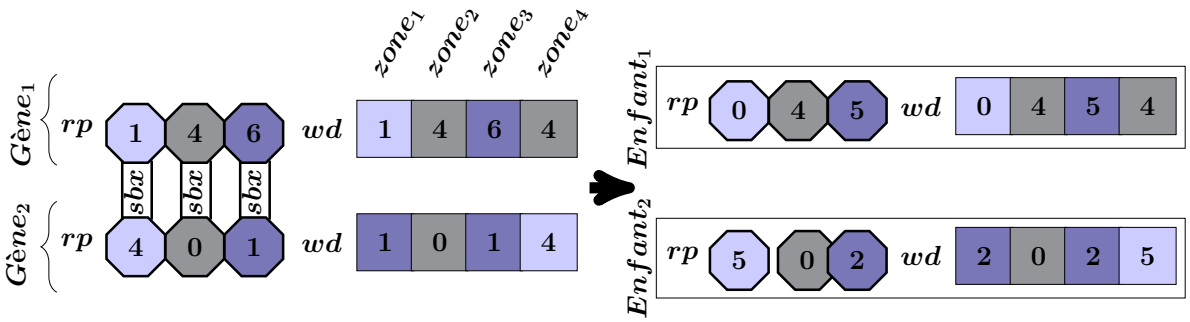


FIGURE 6.2 – Exemple de croisement

6.3.2.2 Mutation

Nous avons utilisé la mutation polynomiale PM comme utilisée dans de nombreux algorithmes évolutionnaires résolvant les MOOP [191]. Tout comme l'opérateur de croisement, la mutation opère sur les vecteurs ***rp***. Si un emplacement de réplique du vecteur ***rp*** est muté, toutes ses occurrences dans le vecteur ***wd*** sont mises à jour en conséquence. Aussi, les gènes respectent les contraintes ((6.1h) et (6.1g)) qui assurent le stockage d'une copie localement et qu'au plus, une réplique d'un objet est stockée dans un emplacement donné.

6.3.3 *StorNIR* matheuristique

L'approche matheuristique de ***StorNIR*** est la même que celle de CDP-NSGAII$_{IR}$. La méthode exacte permet d'obtenir un ensemble de solutions initiales exactes non dominées (sous-ensemble de Pareto) en transformant le MOOP défini dans la section 6.2 en problème d'optimisation mono-objectif et en utilisant le solveur CPLEX pour sa résolution. L'algorithme 4, introduit dans le chapitre précédant (voir la section 5.4.3.1, chapitre 5), est utilisé dans cette méthode exacte. La sortie de cette étape peut être utilisée telle quelle dans l'étape *Execute* de la boucle MAPE-K ou envoyée à l'étape évolutionaire de la matheuristique pour améliorer la qualité de l'ensemble de Pareto. Dans ce dernier cas, les solutions obtenues par la méthode exacte sont injectées dans la population initiale de NSGAII et une fonction de réparation (voir la section 6.3.4) intervient avant l'évaluation de chaque individu de la population. A chaque itération, les conditions de terminaison sont vérifiées, si elles ne sont pas satisfaites, la population est triée selon un opérateur de comparaison et une nouvelle population est générée en utilisant les opérateurs d'évolution décrits dans la section 6.3.2.

Dans le reste de cette section, nous décrivons la fonction de réparation et celle de sélection.

6.3.4 Fonction de réparation

La méta-heuristique NSGAII ne s'adapte pas très bien aux grandes instances de problèmes en raison de l'initialisation aléatoire et des opérateurs d'évolution. Pour faire face à ce problème, nous avons conçu une fonction de réparation permettant de corriger les solutions afin de respecter les contraintes du problème. Cette fonction consiste d'abord à trouver les objets qui violent les contraintes du système. Ensuite, le placement et la sélection de ces répliques sont modifiées en les déplaçant vers des emplacements appropriés répondant à la fois aux limitations de ressources et aux SLA des clients. L'algorithme 6 formule cette fonction de réparation et est expliqué comme suit.

Dans la première phase (lignes 3 à 9 de l'algorithme 6), chaque chromosome est parcouru gène par gène. Si le gène courant ne respecte pas toutes les contraintes, alors on le met dans une liste $\boldsymbol{LR}$ pour une réparation ultérieure. A la fin de cette première phase, $\boldsymbol{LR}$ contient tous les gènes qui ne respectent pas les contraintes du système.

Algorithme 6 Fonction de réparation

1: **Fonction** RÉPARER(***chromosome***)
2: $\boldsymbol{LR} \leftarrow \emptyset$
3: **Pour chaque** (Gene $\boldsymbol{g}$ dans le chromosome) **Faire**
4: $\boldsymbol{o_{i,k}} \leftarrow$ *l'objet correspondant au* $\boldsymbol{g}$
5: *Verifier les contraintes pour* $\boldsymbol{o_{i,k}}$ en utilisant ses vecteurs **rp et wd**
6: **Si** (*$\boldsymbol{o_{i,k}}$ ne respecte pas toutes les contraintes*) **Alors**
7: $\boldsymbol{LR} \leftarrow \boldsymbol{LR} \cup \boldsymbol{o_{i,k}}$
8: **Fin Si**
9: **Fin Pour**
10: **Tant que** ($\boldsymbol{LR} \neq \emptyset$) **Faire**
11: $\boldsymbol{LP} \leftarrow \emptyset$
12: $\boldsymbol{o_{i,k}} \leftarrow$ *Choisir aléatoirement un objet à réparer de la liste* $\boldsymbol{LR}$
13: $\boldsymbol{LP}$*.ajouter(toutes les combinaisons des emplacements* $\boldsymbol{r_{o_{i,k}}}$ *sauf celle qui est déja affectée à* $\boldsymbol{o_{i,k}}$ *et celles qui ne respectent pas les contraintes des équations (6.1g, 6.1h)*
14: **Tant que** ($\boldsymbol{LP} \neq \emptyset$) **Faire**
15: $\boldsymbol{rp} \leftarrow$ trouver une combinaison $\boldsymbol{LP}$ qui satisfait les équations (6.1b et 6.1c)
16: **Si** ($\boldsymbol{rp}$ est trouvé) **Alors**
17: $\boldsymbol{wd} \leftarrow$ GÉNÉRERWD$(\boldsymbol{rp}, \boldsymbol{w^k})$
18: **Si** ($\boldsymbol{wd}$ est valide) **Faire**
19: Assigner $\boldsymbol{rp}$ et $\boldsymbol{wd}$ à $\boldsymbol{o_{i,k}}$
20: $\boldsymbol{LR} \leftarrow \boldsymbol{LR} - \{\boldsymbol{o_{i,k}}\}$
21: *Aller à 10*
22: **Fin Si**
23: $\boldsymbol{LP}$ - $\{\boldsymbol{rp}\}$
24: **Fin Si**
25: **Fin Tant que**
26: **Si** ($\boldsymbol{o_{i,k}}$ n'est pas réparer) **Alors**
27: *Retourner le chromosome initial*
28: **Fin Si**
29: **Fin Tant que**
30: *Retourner le nouveau chromosome*

Ensuite, nous essayons de réparer tous les gènes[1] de la liste $\boldsymbol{LR}$ (lignes 10 à 25 de l'algo-

1. Nous rappelons que les gènes représentent les copies des objets

rithme 6). Nous la parcourons aléatoirement gène par gène. Pour chaque gène sélectionné, nous construisons une liste d'emplacements cibles potentiels $\boldsymbol{LP}$. Cette liste contient toutes les combinaisons des emplacements de répliques correspondants. Nous vérifions que les valeurs de chaque combinaison $\boldsymbol{LP}$ contiennent un seul emplacement local et que les emplacements doivent être différents (contraintes : équation (6.1h) et équation (6.1g)). Ensuite, nous recherchons à partir de $\boldsymbol{LP}$ une combinaison de $\boldsymbol{rp}$ emplacements qui satisfait les contraintes d'occupation locales et externes (équation (6.1b) et équation (6.1c)). Si $\boldsymbol{rp}$ est trouvé, nous essayons de construire le vecteur $\boldsymbol{wd}$ en utilisant l'algorithme 7 (décrit ci-dessous). Si $\boldsymbol{wd}$ satisfait toutes les contraintes d'E/S (équation (6.1d), équation (6.1e) et équation (6.1f)), nous acceptons les affectations de répliques ($\boldsymbol{rp}$) et de sélection ($\boldsymbol{wd}$).

Il est à noter que si, pour un gène donné, on ne trouve pas de vecteurs $\boldsymbol{rp}$ et $\boldsymbol{wd}$ valides satisfaisant toutes les contraintes, le processus de réparation est interrompu et la configuration initiale du chromosome est conservée. Ceci est exprimé dans les lignes 26 à 28 de l'algorithme 6.

Le but de l'algorithme 7 (appelé dans l'algorithme 6, la ligne 17) est de trouver un schéma de sélection de répliques optimisé (vecteur $\boldsymbol{wd}$) qui minimise à la fois l'exécution de la charge de travail d'E/S de stockage et la latence du réseau et satisfait les contraintes d'E/S (équation. 6.1d, équation 6.1e et équation 6.1f). Pour cela, pour chaque zone $\boldsymbol{z}$, pour chaque type de requête $\boldsymbol{r}$, nous vérifions s'il existe une charge de travail $\boldsymbol{w}^{\boldsymbol{k}}_{\boldsymbol{r},\boldsymbol{z}}$ correspondant au type de requête $\boldsymbol{r}$ et provenant de la zone $\boldsymbol{z}$. Si c'est le cas, on recherche à partir du vecteur $\boldsymbol{rp}$ l'emplacement le plus proche de la zone $\boldsymbol{z}$ qui minimise le coût de la charge de stockage d'E/S et respecte les contraintes d'E/S. Si un emplacement est trouvé, nous l'affectons à l'exécution de $\boldsymbol{w}^{\boldsymbol{k}}_{\boldsymbol{r},\boldsymbol{z}}$, sinon nous choisissons aléatoirement un emplacement dans le vecteur $\boldsymbol{rp}$.

Algorithme 7 Géneration de $\boldsymbol{wd}$

```
1: Fonction GénérerWD(rp : vecteur de placement de répliques, w^k : la charge de travail du client k )
2:     Pour (z ← 1 à M) Faire
3:         Pour (r ← 1 à R) Faire
4:             Si ( w^k_{r,z} n'est pas nul) Alors
5:                 Faire
6:                     p ← l'emplacement le plus proche avec un prix d'IOPS minimum et qui satisfait les contraintes (6.1d),
                       (6.1e) et (6.1f)
7:                 Tant que (p trouvé ou tous les rp testés)
8:                 Si p trouvé Alors
9:                     wd[r][z] ← p ;
10:                Sinon
11:                    wd[r][z] ← un emplacement aléatoire à partir de rp ;
12:                Fin Si
13:            Sinon
14:                wd[r][z] ← un emplacement aléatoire de rp ;
15:            Fin Si
16:        Fin Pour
17:    Fin Pour
```

La prochaine partie décrit la méthode et les résultats d'évaluation de l'approche présentée.

6.4 Évaluation des performances

Cette section présente l'évaluation de l'approche d'optimisation proposée. Notre objectif est triple : (i) prouver la pertinence de l'approche proposée en la comparant à la méta-heuristique MOPSO [205] qui est un algorithme évolutionnaire multi-objectifs mature utilisé dans plusieurs études comme [77, 206], (ii) Généraliser l'approche proposée (injection et réparation) en utilisant d'autres méta-heuristiques, et enfin (iii) Montrer la flexibilité de notre approche qui permet de proposer un ensemble de compromis entre le temps d'exécution et la qualité des solutions obtenues.

6.4.1 Méthodologie d'évaluation

Nous évaluons ***StorNIR*** en suivant la même méthodologie décrite dans le chapitre précédent (voir la figure 5.6).

En effet, nous varions les instances du problème de placement et nous considérons le HV et le temps d'exécutions comme métriques pour l'évaluation de ***StorNIR***. Il convient de souligner que nous avons évalué les grandes instances de problème. Pour chaque instance, les valeurs médianes du HV et les valeurs moyennes du temps d'exécution ont été considérées sur 10 exécutions.

Dans un premier temps, nous évaluons l'efficacité des fonctions d'injection et de réparation lorsqu'elles sont couplées avec la méta-heuristique NSGAII pour l'optimisation du problème de placement. Pour le montrer, nous comparons HV de différentes variantes de notre approche, MOPSO et l'ensemble des solutions obtenues par CPLEX. Les différentes variantes de notre approche sont ***StorN*** (uniquement la méta-heuristique NSGAII), ***StorNI*** (NSGAII et la fonction d'injection), ***StorNR*** (NSGAII et la fonction de réparation) et ***StorNIR*** (NSGAII avec les deux fonctions d'injection et de réparation).

Ensuite, nous montrons que les fonctions de réparation et d'injection conçues peuvent également être appliquées avec succès à d'autres méta-heuristiques. Nous appliquons le même processus sur la méta-heuristique MOPSO et l'évaluons avec et sans ces deux opérateurs. Les valeurs cumulées des HV médianes des différents cas d'utilisation testés ont été prises en compte.

Enfin, nous montrons comment l'approche proposée permet au décideur de faire un compromis entre la qualité de HV et le temps d'exécution de la matheuristique. Pour cela, nous calculons le HV et le temps d'exécution de CPLEX et de ***StorNIR*** en variant le nombre de solutions calculées par CPLEX et utilisées dans l'étape d'injection de la matheuristique. Les paramètres des coefficients sont définis conformément au tableau 6.3 pour résoudre le problème à l'aide de CPLEX.

6.4.2 Paramètres expérimentaux

Les expériences ont été conçues sur la base du scénario suivant. Une fédération de *Clouds* contient un ensemble de CSP. Les CSP appartiennent à des zones géographiques différentes. La latence du réseau entre chaque paire de zones est attribuée de manière aléatoire dans un intervalle]100ms, 700ms]. Le CSP considéré gère un ensemble de clients, chacun ayant un objet de données. Les clients génèrent des charges de travail. Dans notre travail, nous avons utilisé YCSB [199] pour générer les charges de travail des clients. Nous avons utilisé l'outil Blktrace [200] pour capturer différentes opérations d'E/S générées par ces requêtes. Comme dans [189], le nombre de requêtes par client a été pondéré selon une distribution normale ($\boldsymbol{\mathcal{N}(12, 6)}$) avec une moyenne égale à **12** heures et un écart type de **6** heures.

Chaque objet est répliqué trois fois. Ce facteur de réplication est largement utilisé dans les systèmes de stockage comme HDFS et GFS et il assure une haute disponibilité [21].

Le CSP concerné héberge un système de stockage hybride composé d'un ensemble de 50 HDD et 50 SSD ayant respectivement les caractéristiques de 1 To Hitachi Deskstar 7K1000.D modèle HDS721010DLE630 7200 RPM HDD [194] et 1 To Samsung SSD 970 EVO Plus [207]. Le CSP peut également utiliser les services de stockage proposés par ses CSP partenaires qui ont des prix et des performances différents. Nous supposons que les services de stockage proposés sont définis selon trois services de stockage d'Amazon *Cloud* à savoir EBS io1, EBS gp2 et *magnetic disks* [208]. Nous supposons également que seul le débit réseau sortant est facturé et son prix est fixé à 0,09$/Go [197] comme chez Amazon. Pour motiver l'utilisation des ressources de la fédération, on suppose que les prix d'occupation des stockages évoluent en fonction du taux d'occupation comme dans [18, 19] et que ce dernier est corrélé à la période de temps pendant la journée car la charge générée a un cycle quotidien [18]. Nous supposons que le taux d'occupation suit une distribution normale ($\boldsymbol{\mathcal{N}(12, 6)}$) avec une moyenne égale à **12**am et un écart de **6** heures. Ainsi, chaque CSP fixe ses prix de stockage des ressources fédérées selon cette distribution [19].

Dans notre évaluation, la pénalité de violation du SLA de stockage est fixée à 10% du montant total payé par le client.

Enfin, comme dans [23, 189], la latence réseau des requêtes des clients est facturée. La correspondance entre le prix et la latence est donnée dans le tableau 6.1 et la pénalité associée est illustrée dans le tableau 6.2.

Les coefficients du tableau 6.3 ont été choisis pour calculer les solutions avec la méthode exacte. Comme CPLEX ne peut pas obtenir de solutions optimales dans un délai raisonnable pour certaines instances, le temps d'exécution a été limité à 60 secondes.

Les tests expérimentaux dans le travail présenté dans cette contribution ont été exécutés pour plusieurs scénarios, en considérant différents nombres de CSP et d'objets. Nous avons fait varier le nombre de CSP entre 15 et 25 avec un pas de 5 et le nombre d'objets entre 100 et 500. Chaque cas d'utilisation a été exécuté 10 fois.

Latence (ms)	Prix par requête ($)
< 200	0.0000001
< 400	0.00000005
< 600	0.00000002
> 600	0

Tableau 6.1 – Prix de latence par requête

Latence (ms)	Penalité ($)
< 200	0
< 400	0.00000005
< 600	0.00000008
> 600	0.0000001

Tableau 6.2 – Pénalité de latence par requete

α	1	0	0	0.5	0.5	0	0.6	0.2	0.2	0.4
β	0	1	0	0.5	0	0.5	0.2	0.6	0.2	0.3
γ	0	0	1	0	0.5	0.5	0.2	0.2	0.6	0.3

Tableau 6.3 – Coefficients de la méthode exacte

Les résultats expérimentaux ont été comparés en fonction du temps d'exécution et de HV. Nous rappelons que cet indicateur consiste à calculer le volume dominé par l'ensemble des valeurs des fonctions objectifs de la solution trouvée.

6.4.3 Résultats de l'évaluation

6.4.3.1 Impact des opérateurs d'injection et de réparation

L'ensemble de solutions de CPLEX est composé de trois solutions, chacune d'entre elles correspondant au minimum de chaque fonction objectif (les trois premières combinaisons de coefficients du tableau 6.3).

La figure 6.3 montre, pour chaque algorithme, le HV dérivé de son front de Pareto résultant en fonction de la taille du problème. Nous observons que ***StorN*** est plus performant que MOPSO avec des HV plus élevés. Cependant, lorsque la taille du problème augmente, ni ***StorN*** ni MOPSO ne parviennent à trouver des solutions faisables. L'évaluation montre également que les procédures d'injection et de réparation améliorent le HV de NSGAII. Pour les instances où ***StorN*** est capable de trouver des solutions, le HV de ***StorNR*** est de 2 à 7 fois supérieure au HV de ***StorN*** tandis que le HV de ***StorNI*** est 6 à 17 fois supérieur à celui de ***StorN***. De plus, l'amélioration est plus importante lors de l'utilisation de ***StorNIR***. ***StorNIR*** augmente le HV de ***StorN*** de 16 à 34 fois. En moyenne, ***StorNIR*** améliore le HV de CPLEX de 17 fois tandis que ***StorNI*** n'améliore le HV de CPLEX que de 4,8 fois.

Pour le temps d'exécution, nous avons comparé ***StorN***, ***StorNR*** et MOPSO afin de voir l'impact de l'opérateur de réparation sur le temps d'exécution. Nous n'avons pas pris en compte ***StorNI*** et ***StorNIR*** car ils dépendent aussi du temps d'exécution de CPLEX pour calculer leurs solutions initiales. En effet, le temps d'exécution de CPLEX et ceux des meta-heuristiques n'ont pas le même ordre de grandeur. Bien que, les heuristiques prennent quelques secondes,

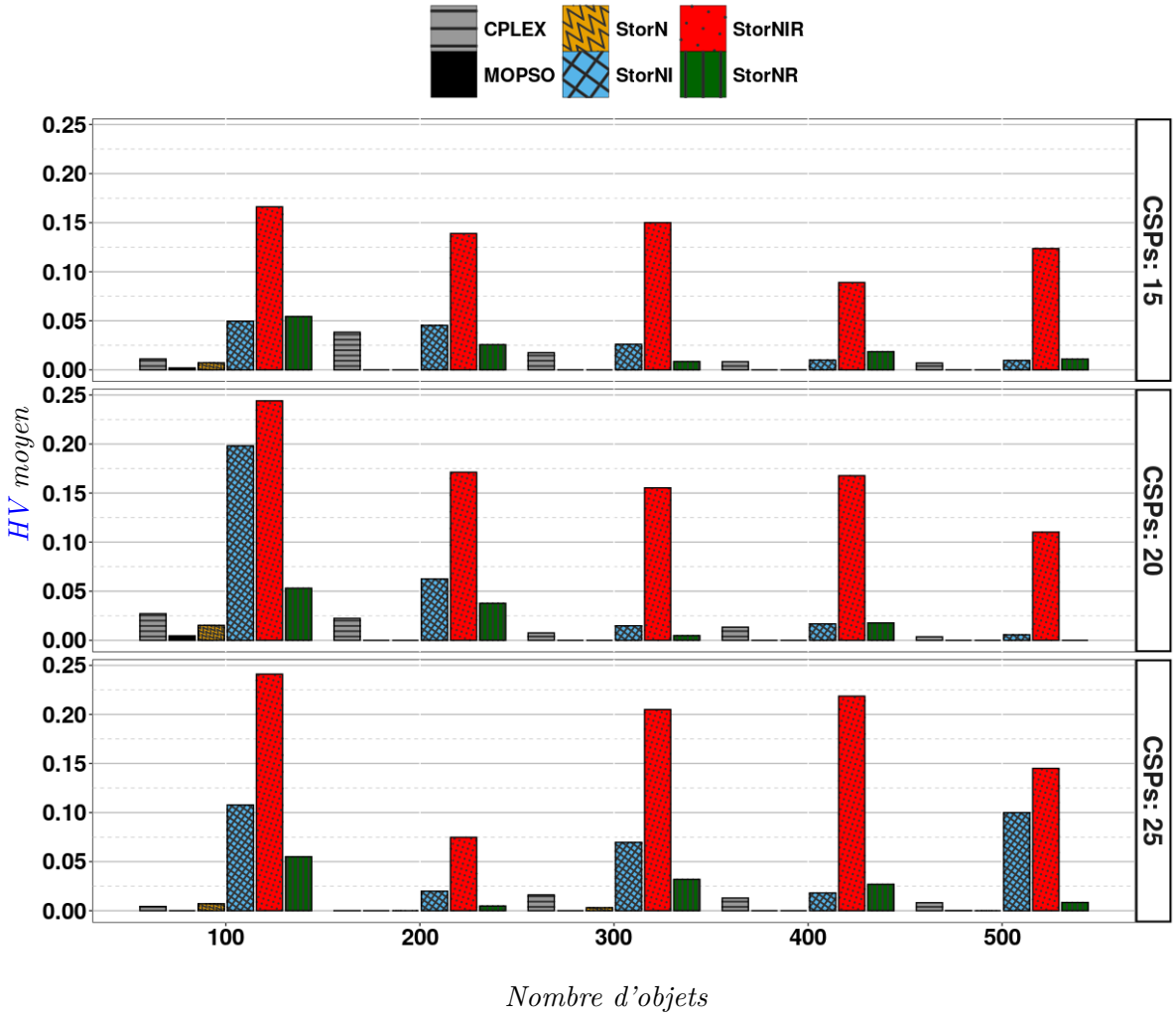


FIGURE 6.3 – Comparaison de HV

le temps nécessaire pour le calcule des solutions avec CPLEX est environ cent secondes pour seulement trois solutions.

À partir de la figure 6.4, nous remarquons que les trois temps d'exécution sont proches les uns des autres. Le temps d'exécution cumulé de ***StorNR*** est légèrement supérieur à celui des autres algorithmes. En effet, ***StorNR*** prend 7,4% de temps (cumulé) de plus que ***StorN*** et 1,8% de temps (cumulé) de plus que MOPSO. On remarque aussi que le temps d'exécution des différents algorithmes augmente avec l'augmentation de la taille du problème qui dépend du nombre d'objets et de CSP. Pour ***StorNR***, cela est dû à la fonction de réparation qui nécessite des efforts supplémentaires pour la correction des individus de la population, alors que ***StorN***

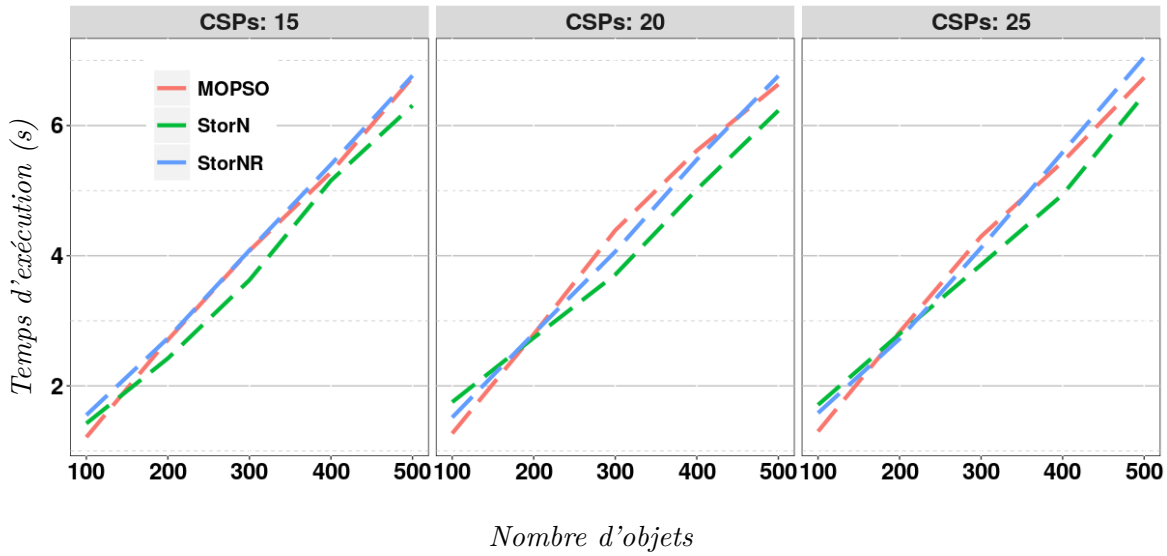


FIGURE 6.4 – Évolution du temps d'exécution

et MOPSO prennent plus de temps lorsque la taille du problème augmente car ils doivent classer les individus selon le degré de violations des contraintes (pour pénaliser les individus invalides) et nous savons par la formulation du problème que le nombre de contraintes dépend à la fois du nombre d'objets et du nombre de CSP (emplacements). La figure 6.4 montre également que le temps d'exécution est plus impacté par le nombre d'objets que par le nombre de CSP, car nous avons considéré un grand nombre d'objets par rapport au nombre de CSP.

Pour récapituler, à partir de la figure 6.3 et de la figure 6.4, la fonction de réparation améliore le HV de NSGAII de 2 à 7 fois avec 7,4% de temps d'exécution supplémentaire (cumulé).

6.4.3.2 Généralisation de l'approche proposée à d'autres méta-heuristiques

La figure 6.5 montre les valeurs cumulées de HV des variantes ***StorN**∗* et ***MOPSO**∗*. Le symbole de la légende **(-)** signifie la version classique de l'algorithme, tandis que les symboles **(R, I, IR)** signifient que les algorithmes sont mis à jour respectivement par les fonctions de réparation, d'injection et à la fois d'injection et de réparation. La ligne bleue horizontale représente le HV cumulé de CPLEX. Notez que pour cette expérience, l'ensemble de solutions de CPLEX est également composé de trois solutions calculées à l'aide des trois premières combinaisons de coefficients du tableau 6.3.

Nous remarquons sur la figure 6.5 que les fonctions d'injection et de réparation peuvent être intégrées avec succès dans la méta-heuristique MOPSO. Les résultats montrent que MOPSO sans injection ni réparation peut rarement trouver des solutions faisables. MOPSO avec seulement la

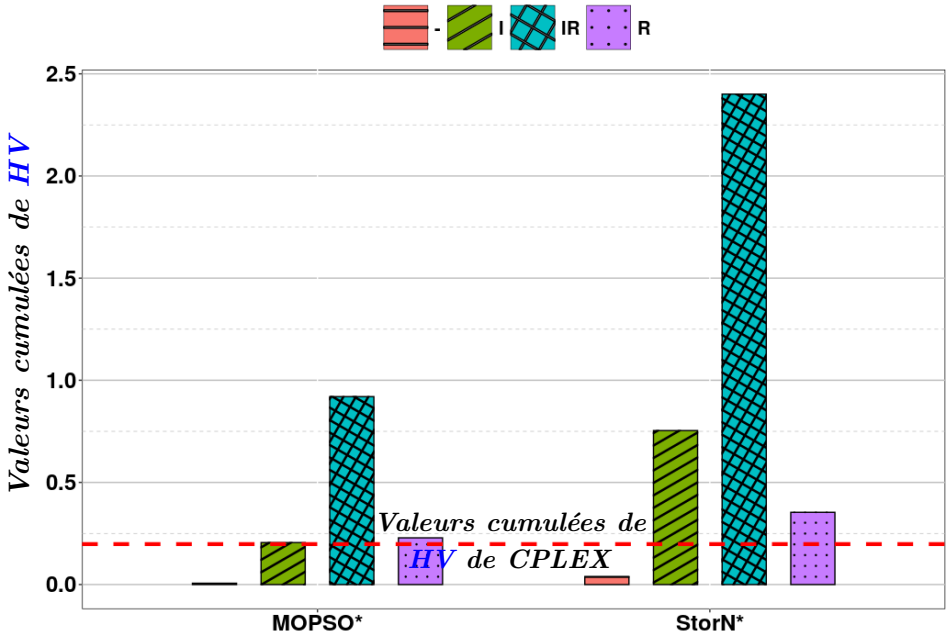


FIGURE 6.5 – Généralisation de la matheuristique

réparation ou seulement l'injection, on peut obtenir un HV plus ou moins égal à celui de CPLEX. Ainsi, MOPSO avec injection fait plus d'efforts sans améliorer nettement le HV de l'ensemble initialement injecté pour les cas d'utilisation testés. Sans surprise, les résultats montrent que MOPSO avec les procédures d'injection et de réparation a le meilleur HV par rapport à ses autres variantes. En effet, la matheuristique basée sur MOPSO améliore 4 fois le HV de CPLEX. Cependant, l'amélioration est encore plus grande pour la matheuristique ***StorNIR*** que pour ***MOPSO-IR***. En fait, le HV de ***StorNIR*** est supérieur à celui de ***MOPSO-IR*** d'environ 2,6 fois.

6.4.3.3 Flexibilité de *StorNIR*

Dans cette évaluation, nous avons pris en compte, à la fois, le temps d'exécution cumulé et les HV pour différentes instances de problème. Ces valeurs sont obtenues pour ***StorNIR*** et CPLEX en fonction du nombre de solutions calculées par CPLEX et injectées dans ***StorNIR***. Dans la figure 6.6, chaque point rouge et vert correspond respectivement au temps d'exécution et au HV de CPLEX et ***StorNIR***. Les étiquettes de points correspondent au nombre de solutions calculées par CPLEX. Tout d'abord, nous observons que ***StorNIR*** prend un peu plus de temps d'exécution car il augmente l'ensemble de solutions de CPLEX par le calcul évolutif requis par NSGAII et la fonction de réparation. Néanmoins, ce temps supplémentaire dans ***StorNIR***

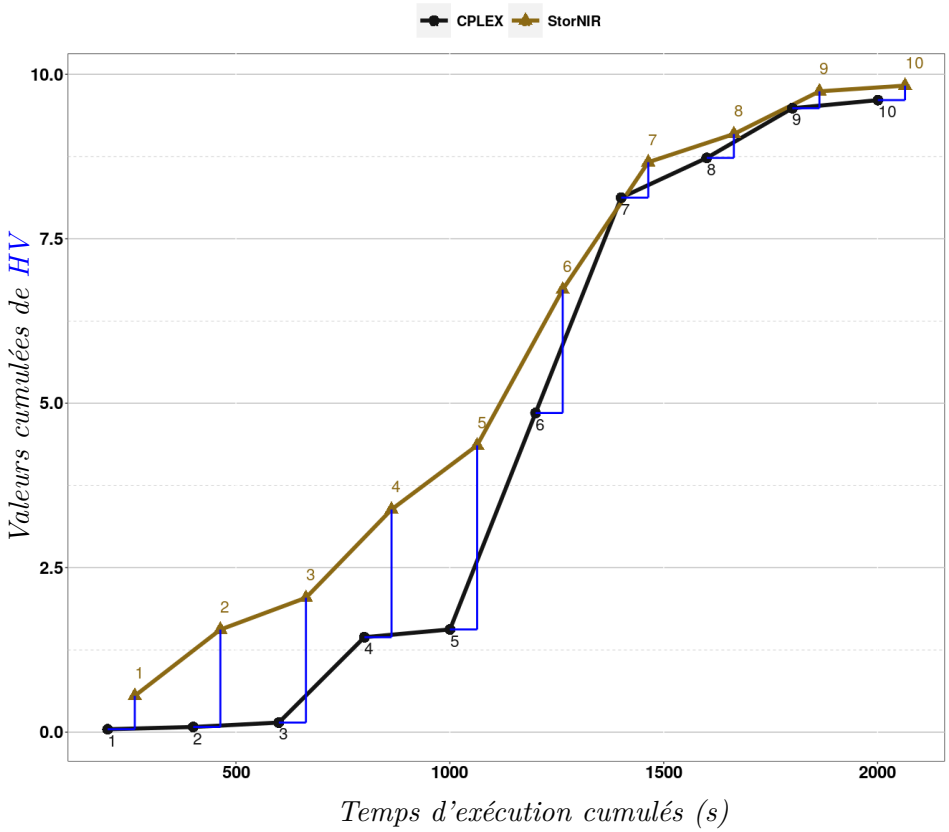


FIGURE 6.6 – Flexibilité de la matheuristique

est en grande partie constant car il dépend du nombre d'évaluations de la méta-heuristique NSGAII et n'est pas corrélé au nombre de solutions injectées. Par exemple, lors de l'injection de 3 solutions à partir de CPLEX, le temps d'exécution (cumulé) de CPLEX est de 600 s, tandis que ***StorNIR*** s'exécute en 663 s et le HV (cumulé) de CPLEX est égal à 0,14 tandis que celui de ***StorNIR*** est égal à 2,043. Cela signifie que ***StorNIR*** améliore le HV de CPLEX de 14 fois avec 10% de temps supplémentaire.

Pour le HV, nous remarquons que le HV (cumulé) de ***StorNIR*** est toujours supérieur à celui de CPLEX. ***StorNIR*** améliore CPLEX jusqu'à 19 fois (2 solutions injectées dans la figure). La différence de HV est plus importante lorsque le nombre de solutions calculées par CPLEX est inférieur ou égal à 6. En moyenne, dans un tel cas, le HV moyen (cumulé) de ***StorNIR*** est 8,8 fois supérieur à celui de CPLEX. Ensuite, la différence devient plus petite avec plus d'effort requis par la partie heuristique. En effet, réduire le nombre de solutions calculées par CPLEX est un moyen d'améliorer la scalabilité de l'approche matheuristique car lorsque la taille du problème augmente, CPLEX prend beaucoup plus de temps pour trouver des solutions exactes tandis que le temps d'exécution de la partie heuristique de ***StorNIR*** reste inchangé.

6.5 Conclusion

Dans ce travail, nous avons abordé le problème de placement et de sélection des répliques pour un CSP membre d'une fédération. Ce dernier doit répondre à la QoS de ses clients et minimiser le coût de placement en utilisant ses ressources de stockage locales et celles de ses partenaires. Nous avons modélisé ce problème sous forme d'un MOOP prenant en compte les différentes caractéristiques du système de stockage local, des services de stockage externes et des clients. Pour résoudre le MOOP proposé, nous avons conçu ***StorNIR***, une matheuristique qui consiste d'abord à calculer des solutions à l'aide d'un outil MILP. Le solveur CPLEX a été utilisé à cette fin. L'ensemble des solutions calculées par CPLEX est ensuite injecté dans la population initiale de la méta-heuristique NSGAII. Enfin, un opérateur de réparation est utilisé à chaque génération pour fixer les solutions qui ne satisfont pas les contraintes du système.

L'évaluation a prouvé l'efficacité de la matheuristique proposée. ***StorNIR*** fournit de meilleures solutions que NSGAII et de CPLEX avec des temps d'exécution intéressants. En effet, le temps d'exécution de l'étape évolutionaire conséquente augmente légèrement le temps d'exécution de NSGAII (7.4%) pour une amélioration du HV (jusqu'à 34 fois). ***StorNIR*** est flexible car cette matheuristique permet de faire un compromis entre la qualité des solutions (HV) et le temps d'exécution en ajustant la fonction d'injection. De plus, l'approche proposée s'est avérée être généralisable avec succès à d'autres méta-heuristiques telles que MOPSO en permettant d'améliorer également les résultats de cette meta-heuristique avec les fonctions d'injection et de réparation.

CONCLUSION

Chapitre 7

Conclusion

Le *Cloud computing* représente une révolution dans le domaine de l'informatique. En effet, il est devenu une partie intégrante de la plupart des applications Internet.

Un grand effort sur les architectures de *Cloud computing*, a été consacré au déploiement de grands centres de données centralisés avec des milliers voire des centaines de milliers de serveurs. Même si ce modèle reste largement utilisé, il pose plusieurs défis de gestion des ressources. Aussi, il pose un problème lorsqu'une rupture de service se produit. De plus, cette architecture centralisée engendre des temps de latence très élevés. La fédération de *Clouds*, permettant la collaboration de plusieurs *Clouds*, se présente comme une solution clé pour surmonter les obstacles liés à l'architecture centralisée du *Cloud* traditionnel. La fédération de *Clouds* permet non seulement de réduire la latence perçue par l'utilisateur en mettant les ressources à sa proximité, mais également, elle permet d'offrir une plus grande disponibilité des données et une meilleure tolérance aux pannes en répliquant les données sur plusieurs sites. Elle offre aussi la possibilité de mettre en place un traitement parallèle. De plus, la fédération permet d'améliorer la productivité des CSP sans investissement dans la construction de nouveaux centres de données.

Les fournisseurs de *Cloud* essaient d'optimiser leurs performances en agissant sur plusieurs paramètres comme les coûts énergétiques, les coûts de virtualisation, etc. Une des options d'optimisation consiste à choisir un bon placement des données sur différents types de support de stockage afin de minimiser les coûts tout en garantissant une meilleure qualité de service.

Dans le contexte d'une fédération de *Clouds*, le problème de placement de données pour un CSP membre d'une fédération est très complexe dû, d'une part, à l'hétérogénéité des ressources locales et de services de stockage de données proposés par les *Clouds* partenaires, et à l'environnement variable de la fédération et d'autre part aux charges de travail variables des clients et leur différent SLA. Ainsi, selon les exigences de qualité de service des clients, un CSP peut utiliser ses différentes classes de stockage ainsi que les services de stockage proposés par les autres CSP partenaires.

Cette thèse traite l'optimisation du placement de données dans un *Cloud* fédéré. L'objectif global est de fournir des stratégies efficaces pour trouver des solutions de placement pertinentes qui permettent d'obtenir les meilleurs compromis entre la satisfaction des utilisateurs et la minimisation des coûts pour le fournisseur.

Ce dernier chapitre récapitule les travaux de recherche réalisés dans le cadre de cette thèse

et décrit quelques perspectives qui nous semblent prometteuses pour des travaux futurs.

Sommaire

7.1 Résumé des contributions

Nous avons apporté trois contributions dans le cadre de cette thèse. La première est consacrée à l'évaluation du coût de placement des données dans un *Cloud* fédéré à base d'un système de stockage hybride. La deuxième contribution consiste en une stratégie de placement de données prenant en compte les contraintes liées à l'environnement dynamique d'un *Cloud* fédéré. Enfin, la troisième contribution augmente la deuxième par le mécanisme de réplication.

7.1.1 Modèle de coût pour l'évaluation du placement dans un Cloud fédéré

Minimiser les coûts d'exploitation pour les fournisseurs de services *Cloud* représente généralement un grand défi notamment lorsqu'il s'agit d'un environnement complexe comme une fédération de *Clouds*. À cet égard, nous avons proposé un modèle de coût pour l'évaluation du coût de placement de données dans un *Cloud* faisant partie d'une fédération. Le CSP peut utiliser ses différentes classes de stockage ainsi que les services de stockage proposés par les autres CSP partenaires pour migrer ou répliquer les données de ses clients.

Le modèle de coût proposé inclut 1) les coûts de placement des données des clients internes, y compris les coûts de placement local, d'externalisation, de rapatriement et de pénalité, et 2) les coûts de placement des données des clients externes, y compris les coûts d'internalisation et de renvoi des données de clients. Le modèle de coût prend en compte différents paramètres liés aux caractéristiques des classes de stockage internes et aux services proposés par les autres CSP. Il considère également la charges de travail des clients, leurs exigences en termes de QoS décrites dans le SLA et les pénalités associées.

Les résultats des évaluations ont montré la pertinence des sous-coûts considérés. Ils ont aussi montré qu'une non prise en compte de certains sous-coûts peut entraîner une erreur de coût de 95 % pour le placement des données des clients externes et de 80 % pour l'externalisation des clients. Cela peut entraîner des pertes financières importantes. Le modèle de coût permet d'étudier si l'externalisation est pertinente dans une fédération en fonction du coût des ressources et des propriétés de charge de travail.

Même si notre modèle est orienté stockage, il peut être utilisé et intégré dans un modèle de coût plus large en tenant compte d'autres ressources comme le processeur et la mémoire.

7.1.2 Stratégie multi-objectifs de placement de données dans un *Cloud* fédéré

L'objectif principal du CSP est de minimiser le coût de placement tout en respectant les exigences des clients en terme de performances d'E/S de stockage et en minimisant la latence réseau.

Le coût de placement des objets des clients est constitué de trois parties liées à trois opérations qui sont 1) le stockage (ce coût a une relation avec le placement futur des objets), 2) la latence

du réseau (a une relation avec le placement futur des objets et la localisation future des clients), et 3) la migration (a une relation avec le placement actuel et futur des objets). Ces coûts sont interdépendants et dans certain cas contradictoires. Par exemple, le placement d'un objet peut minimiser le coût de la latence réseau et le coût de stockage mais peut également générer un coût de migration important. D'autres part, respecter les performances d'E/S de stockage exigées par les clients et minimiser la latence réseau sont parfois contradictoires. En effet, il se peut qu'il existe un placement qui vérifie les contraintes d'E/S mais qui génère de mauvaises latences réseau. Afin de mettre à la disposition du CSP un ensemble de solutions pertinentes, nous avons fait le choix d'une optimisation multi-objectifs.

Nous avons modélisé le problème de placement de données sous forme d'un problème d'optimisation multi-objectifs. Les coûts de stockage, de migration et de latence réseau constituent les fonctions objectifs. Le modèle est soumis à différentes contraintes portant sur l'hétérogénéité des classes locales et services fédérés de stockage, la charge de travail des clients et leur SLA.

La matheuristique CDP-NSGAII$_{\boldsymbol{IR}}$ a été proposée pour résoudre le problème de placement. CDP-NSGAII$_{\boldsymbol{IR}}$ combine une méthode exacte et une méta-heuristique (NSGAII). En effet, le processus d'optimisation passe par deux étape.

Dans la première étape, la méthode exacte calcule un ensemble de solutions exactes à l'aide d'un solveur MILP. Le problème a été transformé en problème d'optimisation mono-objectif en normalisant et en pondérant les différentes fonctions objectifs afin de les agréger en une seule. CPLEX a été utilisé pour résoudre le problème mono-objectif.

La deuxième étape est évolutionnaire, elle est basée sur la méta-heuristique NSGAII. Tout d'abord, l'ensemble de solutions précédemment obtenues est injecté dans la population initiale de NSGAII. Le processus évolutif de NSGAII est exécuté avec une fonction de réparation qui consiste à corriger les individus à chaque itération. La fonction d'injection vise à améliorer la qualité des solutions tandis que la fonction de réparation garantit que les solutions obéissent aux contraintes du problème et empêche ainsi d'explorer de grands ensembles de solutions irréalisables.

Les résultats de l'évaluation ont prouvé l'efficacité de la matheuristique. Ils montrent que la fonction d'injection améliore le HV du NSGAII et la méthode exacte jusqu'à 94% et 60% respectivement tandis que la fonction de réparation réduit le temps d'exécution de 68% en moyenne. La matheuristique proposée est généralisable sur d'autres algorithmes évolutionnaires et permet de faire un compromis entre la qualité des solutions et le temps d'exécution.

7.1.3 Stratégie multi-objectifs de placement et de sélection des répliques

Fournir des services fiables avec une haute disponibilité des données et des performances adéquates sont des exigences clés qui doivent être satisfaites. Le concept de réplication est utilisé pour garantir de telles exigences.

Dans ce travail, nous avons proposé ***StorNIR***, une matheuristique de placement de répliques pour un *Cloud* fédéré. En effet, le modèle précédent a été mis à jour pour inclure la réplication. Lorsque les objets sont répliqués, la latence du réseau des requêtes d'accès doit être optimisée. Par conséquent, dans la solution matheuristique, nous avons proposé un algorithme pour optimiser la latence. Il consiste à affecter les requêtes à l'emplacement le plus proche qui satisfasse les performances d'E/S de stockage.

L'évaluation a prouvé l'efficacité de la matheuristique proposée. ***StorNIR*** fournit de meilleures solutions que NSGAII et CPLEX avec des temps d'exécution intéressants. En effet, la fonction de réparation améliore la méta-heuristique NSGAII jusqu'à 7 fois avec 7,4% de temps d'exécution supplémentaire. Par ailleurs, ***StorNIR*** améliore la qualité de NSGAII jusqu'à 34 fois selon le nombre de solutions injectées. De plus, en moyenne, ***StorNIR*** augmente de 17 fois la qualité des solutions initiales obtenues par CPLEX lorsque les solutions initiales représentent le minimum de chaque fonction objectif.

7.2 Perspectives

Notre travail s'est appliqué au problème du placement de données dans un *Cloud* fédéré. Dans cette section, nous exposons quelques pistes potentielles de recherches futures.

7.2.1 Mécanismes de détection des *Clouds* égoïstes

La fédération de *Clouds* permet de renforcer la coopération entre plusieurs *Clouds* déjà déployés, permettant ainsi aux organisations d'atteindre divers objectifs commerciaux. L'aspect volontaire des membres de la fédération et le maintien de l'intégrité de l'organisation est un aspect important à prendre en considération. Cependant, dans un tel environnement de partage de ressources, il se peut qu'il existe des *Clouds* malveillants appelés *Clouds* "égoïstes" (*selfish Clouds* en anglais) qui essaient par tous les moyens de maximiser leur propre profit. Par exemple, ils utilisent les ressources de leurs partenaires à moindre prix sans partager leurs ressources. Un tel comportement peut nuire aux autres fournisseurs car cela diminue la quantité de ressources dans l'environnement et par conséquent, la fédération devient peu attrayante pour les CSP bien intentionnés [209]. Ainsi, des mécanismes pour éviter ce comportement malveillant sont nécessaires. Certaines travaux ont été proposées dans la littérature pour éviter le comportement égoïste de CSP dans les environnements de partage de ressources [209, 210, 211]. Les solutions proposées sont, de manière générale, basées sur la théorie des jeux. Il serait intéressant de proposer une stratégie de partage de ressources basée sur la théorie de jeux à base d'algorithme de classification d'apprentissage automatique comme **SVM** [212] ou **ANN** [213] pour la classification des membres de la fédération en CSP sains et CSP égoïstes.

7.2.2 Compromis entre disponibilité, cohérence et coût de placement

Dans notre travail portant sur la réplication de données, nous avons un facteur de réplication fixe (3 copies pour chaque objet). Cela peut engendrer une gaspillage de ressources pour les données peu populaires, dites données froides ou en anglais *cold data* et un risque d'une disponibilité insuffisante pour les données très populaires, dites données chaudes (*hot data*). Une solution à cette problématique consiste à hybrider la réplication avec le code correcteur (*erasure coding*) plus précisément la solution consiste à utiliser la réplication pour les données chaudes et la technique *erasure coding* pour les données froides. Cette stratégie est déjà adoptée dans [178]. Néanmoins, les données traitées sont en lecture seule. Il serait intéressant de proposer une heuristique combinant ces deux stratégies de disponibilité de données en prenant en considération des charges de travail avec des opérations d'écriture. Dans ce cas, il est nécessaire de prendre en compte le problème de cohérence de données. En effet, la gestion de la cohérence complexifie davantage le système [214]. Différents niveaux de cohérence de données existent (fort, causal, éventuel, etc.) selon les besoins des applications. Fournir une cohérence forte dégrade la latence d'accès aux données notamment si une forte disponibilité est requise. L'approche à proposer consiste à explorer comment choisir les facteurs de réplication pour les données chaudes et les paramètres de *erasure coding* pour les données froides ainsi que les emplacements des données afin de minimiser le coût tout en répondant aux exigences de la disponibilité et de la cohérence des données des clients.

La tendance actuelle dans le monde de l'informatique, du fait de la complexité des problèmes traités, consiste à explorer des solutions d'apprentissage automatique (*machine learning*) aux divers problèmes, et cela se reflète dans les solutions de placement de données. Ainsi, une des directions naturelles pour étendre le travail actuel est d'exploiter l'apprentissage automatique pour l'identification des données chaudes et froides qui ont une influence directe sur les facteurs de réplication et les paramètres de *erasure coding*. Des travaux actuels offrent de bonnes idées sur la façon dont l'apprentissage automatique peut être exploré pour le problème de placement [215, 216, 217, 218, 219, 220, 221]. Pour prédire les facteurs de réplication et les paramètres de *erasure coding* de manière efficace, il serait intéressant de proposer une approche d'apprentissage automatique se basant sur le seuil de la marge de sécurité [215, 222] pour déterminer un seuil pour chaque paramètre.

7.2.3 Stratégies avancées de placement de données

Jusqu'à présent, nous n'avons considéré que les ressources de stockage. Bien que cela soit raisonnable pour le problème de placement des données, nous pensons qu'il est important d'étendre le travail pour prendre en charge d'autres types de ressources telle que le processeur et la mémoire. Il serait important d'intégrer notre approche de placement dans des systèmes de gestion de données comme *MapReduce*. L'infrastructure Hadoop est une implantation open source de

MapReduce qui utilise le système de fichiers HDFS. Dans cette infrastructure, la localité des données n'a pas été prise en compte pour le lancement des tâches *Map*, cependant, l'ignorance de la localité des données peut réduire sensiblement les performances de *MapReduce* [223]. Un placement de données qui rapproche les données de leurs consommateurs est considéré comme une solution prometteuse pour les applications *Mapreduce* [224]. Certains travaux ont été proposés pour optimiser le placement des données afin d'exploiter efficacement le système *MapReduce* comme [225, 223, 224]. Il serait intéressant d'explorer comment exécuter les tâches de *MapReduce* dans un *Cloud* fédéré utilisant notre approche de placement de données tout en optimisant le calcul, la mémoire et le trafic réseau. Une façon de résoudre ce problème consiste à proposer une heuristique d'optimisation multi-objectifs basée sur la formulation de graphes afin d'optimiser les trois métriques qui sont : le calcul, la mémoire et le trafic réseau.

BIBLIOGRAPHIE

[1] Magic quadrant for cloud infrastructure and platform services, accessed Febuary, 2021. "https://www.gartner.com/doc/reprints?id=1-1ZDZDMTF&ct=200703&st=sb".

[2] Ankita Desai, Rachana Oza, Pratik Sharma, and Bhautik Patel. Hypervisor : A survey on concepts and taxonomy. *International Journal of Innovative Technology and Exploring Engineering*, 2(3) :222–225, 2013.

[3] Roberto Morabito, Jimmy Kjällman, and Miika Komu. Hypervisors vs. lightweight virtualization : a performance comparison. In *2015 IEEE International Conference on Cloud Engineering*, pages 386–393. IEEE, 2015.

[4] Aditya Bhardwaj and C Rama Krishna. Virtualization in cloud computing : Moving from hypervisor to containerization—a survey. *Arabian Journal for Science and Engineering*, pages 1–17, 2021.

[5] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1) :41–50, 2003.

[6] Peter B Galvin, Greg Gagne, Abraham Silberschatz, et al. *Operating system concepts*, volume 10. John Wiley & Sons, 2003.

[7] Marco Indaco. Service oriented non volatile memories. 2020.

[8] K Lamamra, K Belarbi, A Belhani, and S Boukhtini. Nsga2 based of multi-criteria decision analysis for multi-objective optimization of fuzzy logic controller for non linear system. *Journal of Next Generation Information Technology*, 5(1) :57, 2014.

[9] David Villegas, Norman Bobroff, Ivan Rodero, Javier Delgado, Yanbin Liu, Aditya Devarakonda, Liana Fong, S Masoud Sadjadi, and Manish Parashar. Cloud federation in a layered service model. *Journal of Computer and System Sciences*, 78(5) :1330–1344, 2012.

[10] Hongxing Li, Chuan Wu, Zongpeng Li, and Francis CM Lau. Profit-maximizing virtual machine trading in a federation of selfish clouds. In *2013 Proceedings IEEE INFOCOM*, pages 25–29. IEEE, 2013.

[11] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[12] Amir Taherkordi, Feroz Zahid, Yiannis Verginadis, and Geir Horn. Future cloud systems design : challenges and research directions. *IEEE Access*, 6 :74120–74150, 2018.

[13] Wentao Liu. Research on cloud computing security problem and strategy. In *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pages 1216–1219. IEEE, 2012.

[14] The cloud 100, accessed January, 2021. "https://www.forbes.com/cloud100/#5e45e0855f94".

[15] Cloud computing market, accessed Febuary, 2021. "https://www.marketsandmarkets.com/Market-Reports/cloud-computing-market-234.html".

[16] Peter M Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing, 2011.

[17] Positioning technology players within a specific market, accessed Febuary, 2021. "https://www.gartner.com/en/research/methodologies/magic-quadrants-research".

[18] Adel Nadjaran Toosi, Rodrigo N Calheiros, Ruppa K Thulasiram, and Rajkumar Buyya. Resource provisioning policies to increase iaas provider's profit in a federated cloud environment. In *2011 IEEE International Conference on High Performance Computing and Communications*, pages 279–287. IEEE, 2011.

[19] Salma Rebai, Makhlouf Hadji, and Djamal Zeghlache. Improving profit through cloud federation. In *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, pages 732–739. IEEE, 2015.

[20] Marcio RM Assis and Luiz Fernando Bittencourt. A survey on cloud federation architectures : Identifying functional and non-functional properties. *Journal of Network and Computer Applications*, 72 :51–71, 2016.

[21] Yaser Mansouri, Adel Nadjaran Toosi, and Rajkumar Buyya. Data storage management in cloud environments : Taxonomy, survey, and future directions. *ACM Computing Surveys (CSUR)*, 50(6) :1–51, 2017.

[22] Mohamed A Sharaf, Panos K Chrysanthis, Alexandros Labrinidis, and Cristiana Amza. Optimizing i/o-intensive transactions in highly interactive applications. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 785–798, 2009.

[23] Douglas B Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K Aguilera, and Hussam Abu-Libdeh. Consistency-based service level agreements

for cloud storage. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 309–324, 2013.

[24] Binbing Hou, Feng Chen, Zhonghong Ou, Ren Wang, and Michael Mesnier. Understanding i/o performance behaviors of cloud storage from a client's perspective. *ACM Transactions on Storage (TOS)*, 13(2) :1–36, 2017. https://doi.org/10.1145/3078838.

[25] Darren Quick and Kim-Kwang Raymond Choo. Digital droplets : Microsoft skydrive forensic data remnants. *Future Generation Computer Systems*, 29(6) :1378–1394, 2013.

[26] Chengyu Hu, Yuqin Xu, Pengtao Liu, Jia Yu, Shanqing Guo, and Minghao Zhao. Enabling cloud storage auditing with key-exposure resilience under continual key-leakage. *Information Sciences*, 520 :15–30, 2020.

[27] Cloud storage market, accessed Febuary, 2021. "https://www.alliedmarketresearch.com/cloud-storage-market".

[28] Djillali Boukhelef, Jalil Boukhobza, Kamel Boukhalfa, Hamza Ouarnoughi, and Laurent Lemarchand. Optimizing the cost of dbaas object placement in hybrid storage systems. *Future Generation Computer Systems*, 93 :176–187, 2019.

[29] Pengwei Wang, Caihui Zhao, Wenqiang Liu, Zhen Chen, and Zhaohui Zhang. Optimizing data placement for cost effective and high available multi-cloud storage. *Computing and Informatics*, 39(1-2) :51–82, 2020.

[30] William Sentosa, Balakrishnan Chandrasekaran, P Brighten Godfrey, Haitham Hassanieh, Bruce Maggs, and Ankit Singla. Accelerating mobile applications with parallel high-bandwidth and low-latency channels. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, pages 1–7, 2021.

[31] Amazon found every 100ms of latency cost them 1% in sales, accessed February, 2021. "https://www.gigaspaces.com/blog/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/".

[32] Amazon study : Every 100ms in added page load time cost 1% in revenue, accessed Octobre, 2021. "https://www.contentkingapp.com/academy/page-speed-resources/faq/amazon-page-speed-study/".

[33] Jalil Boukhobza and Pierre Olivier. *Flash Memory Integration : Performance and Energy Issues*. Elsevier, 2017. https://www.sciencedirect.com/book/9781785481246/flash-memory-integration.

[34] Zhichao Li, Ming Chen, Amanpreet Mukker, and Erez Zadok. On the trade-offs among performance, energy, and endurance in a versatile hybrid drive. *ACM Transactions on Storage (TOS)*, 11(3) :1–27, 2015. https://doi.org/10.1145/2700312.

[35] Boyang Yu and Jianping Pan. Location-aware associated data placement for geo-distributed data-intensive applications. In *2015 IEEE Conference on Computer Communications (INFOCOM)*, pages 603–611. IEEE, 2015. http://dx.doi.org/10.1109/INFOCOM.2015.7218428.

[36] Carlos A. Coello Coello. Multi-objective optimization. In Rafael Martí, Panos M. Pardalos, and Mauricio G. C. Resende, editors, *Handbook of Heuristics*, pages 177–204. Springer, 2018.

[37] Anirban Mukhopadhyay, Ujjwal Maulik, Sanghamitra Bandyopadhyay, and Carlos Artemio Coello Coello. A survey of multiobjective evolutionary algorithms for data mining : Part i. *IEEE Transactions on Evolutionary Computation*, 18(1) :4–19, 2013.

[38] Antonio Regalado. Who coined 'cloud compuitng' ? mit technologyreview2. https://www.technologyreview.com/2011/10/31/257406/who-coined-cloud-computing/.

[39] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *2008 grid computing environments workshop*, pages 1–10. Ieee, 2008.

[40] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds : towards a cloud definition, 2008.

[41] Cloud computing : les nouveaux modèles économiques, accessed February, 2021. "https://syntec-numerique.fr/cloud-computing".

[42] Mathijs Jeroen Scheepers. Virtualization and containerization of application infrastructure : A comparison. In *21st twente student conference on IT*, volume 21, 2014.

[43] Zheng Li, Maria Kihl, Qinghua Lu, and Jens A Andersson. Performance overhead comparison between hypervisor and container based virtualization. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 955–962. IEEE, 2017.

[44] Michael Eder. Hypervisor-vs. container-based virtualization. *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, 1, 2016.

[45] Roberto Morabito. Power consumption of virtualization technologies : an empirical investigation. In *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, pages 522–527. IEEE, 2015.

[46] MinSu Chae, HwaMin Lee, and Kiyeol Lee. A performance comparison of linux containers and virtual machines using docker and kvm. *Cluster Computing*, 22(1) :1765–1775, 2019.

[47] Docker, accessed Mai, 2021. "https://www.docker.com/".

[48] David Bernstein. Containers and cloud : From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3) :81–84, 2014.

[49] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. The datacenter as a computer : An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3) :1–154, 2013.

[50] Thomas Erl, Ricardo Puttini, and Zaigham Mahmood. *Cloud computing : concepts, technology, & architecture*. Pearson Education, 2013.

[51] Ian Foster and Carl Kesselman. *The Grid 2 : Blueprint for a new computing infrastructure*. Elsevier, 2003.

[52] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing : issues and challenges. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 27–33. Ieee, 2010.

[53] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software : Practice and Experience*, 32(2) :135–164, 2002.

[54] Hausi A Muller. Bits of history, challenges for the future and autonomic computing technology. In *2006 13th Working Conference on Reverse Engineering*, pages 9–18. IEEE, 2006.

[55] Markus C Huebscher and Julie A McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys (CSUR)*, 40(3) :1–28, 2008.

[56] Paul Horn. Autonomic computing : Ibm's perspective on the state of information technology. 2001.

[57] Tobias Anstett, Frank Leymann, Ralph Mietzner, and Steve Strauch. Towards bpel in the cloud : Exploiting different delivery models for the execution of business processes. In *2009 Congress on Services-I*, pages 670–677. IEEE, 2009.

[58] Cor-Paul Bezemer, Andy Zaidman, Bart Platzbeecker, Toine Hurkmans, et al. Enabling multi-tenancy : An industrial experience report. In *2010 IEEE International Conference on Software Maintenance*, pages 1–8. IEEE, 2010.

[59] Judith S Hurwitz and Daniel Kirsch. *Cloud computing for dummies*. John Wiley & Sons, 2020.

[60] William Voorsluys, James Broberg, Rajkumar Buyya, et al. Introduction to cloud computing. *Cloud computing : Principles and paradigms*, pages 1–44, 2011.

[61] Sukhpal Singh and Inderveer Chana. A survey on resource scheduling in cloud computing : Issues and challenges. *Journal of grid computing*, 14(2) :217–264, 2016.

[62] Lin Dai, Xin Gao, Yan Guo, Jingfa Xiao, and Zhang Zhang. Bioinformatics clouds for big data manipulation. *Biology direct*, 7(1) :1–7, 2012.

[63] Mohammed Alhamad, Tharam Dillon, and Elizabeth Chang. Conceptual sla framework for cloud computing. In *4th IEEE International Conference on Digital Ecosystems and Technologies*, pages 606–610. IEEE, 2010.

[64] Jiao Zhang, Fengyuan Ren, and Chuang Lin. Survey on transport control in data center networks. *IEEE Network*, 27(4) :22–26, 2013.

[65] Vikas Kumar and Kavindra Kumar Garg. Migration of services to the cloud environment : Challenges and best practices. *International Journal of Computer Applications*, 55(1), 2012.

[66] Amazon found every 100ms of latency cost them 1% in sales, accessed Mai, 2021. "https://www.gigaspaces.com/blog/amazon-found-every-100ms-of-latency-cost-them-1-in-sales".

[67] Rajat Chaudhary and Archika Kansal. A perspective on the future of the magnetic hard disk drive (hdd) technology. *International Journal of Technical Research and Applications*, 3(3) :63–74, 2015.

[68] Arezki Laga, Jalil Boukhobza, Michel Koskas, and Frank Singhoff. Lynx : A learning linux prefetching mechanism for ssd performance model. In *2016 5th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pages 1–6. IEEE, 2016.

[69] Multi-layer ssds : What are slc, mlc, tlc, qlc, and plc ?, accessed August, 2021. "https://www.howtogeek.com/444787/multi-layer-ssds-what-are-slc-mlc-tlc-qlc-and-mlc/".

[70] Hamza Ouarnoughi. *Placement autonomique de machines virtuelles sur un système de stockage hybride dans un cloud IaaS*. PhD thesis, Université de Bretagne occidentale-Brest, 2017.

[71] Jalil Boukhobza. Flashing in the cloud : Shedding some light on nand flash memory storage systems. In *Data Intensive Storage Services for Cloud Environments*, pages 241–266. IGI Global, 2013.

[72] Ssd vs hdd, accessed Mai, 2021. "https://www.digitaltrends.com/computing/ssd-vs-hdd/".

[73] Persistent disk, accessed Mai, 2021. "https://cloud.google.com/persistent-disk".

[74] Amazon elastic block store, accessed May, 2020. "https://aws.amazon.com/ebs".

[75] Managed disks pricing, accessed Mai, 2021. "https://azure.microsoft.com/en-us/pricing/details/managed-disks/".

[76] Adel Nadjaran Toosi, Ruppa K Thulasiram, and Rajkumar Buyya. Financial option market model for federated cloud environments. In *2012 IEEE Fifth International Conference on Utility and Cloud Computing*, pages 3–12. IEEE, 2012.

[77] Yu Gu, Dongsheng Wang, and Chuanyi Liu. Dr-cloud : Multi-cloud based disaster recovery service. *Tsinghua Science and Technology*, 19(1) :13–23, 2014.

[78] Marcio RM Assis, Luiz Fernando Bittencourt, Rafael Tolosana-Calasanz, and Craig A Lee. Cloud federations : Requirements, properties. *Developing Interoperable and Federated Cloud Architecture*, page 1, 2016.

[79] Le Guan, Xu Ke, Meina Song, and Junde Song. A survey of research on mobile cloud computing. In *2011 10th IEEE/ACIS international conference on computer and information science*, pages 387–392. IEEE, 2011.

[80] Mohammed Islam Naas, Laurent Lemarchand, Jalil Boukhobza, and Philippe Raipin. A graph partitioning-based heuristic for runtime iot data placement strategies in a fog infrastructure. In *Proceedings of the 33rd annual ACM symposium on applied computing*, pages 767–774, 2018.

[81] Lisbeth Rodríguez-Mazahua, Cristian-Aarón Rodríguez-Enríquez, José Luis Sánchez-Cervantes, Jair Cervantes, Jorge Luis García-Alcaraz, and Giner Alor-Hernández. A general perspective of big data : applications, tools, challenges and trends. *The Journal of Supercomputing*, 72(8) :3073–3113, 2016.

[82] Dimitrios G Kogias, Michael G Xevgenis, and Charalampos Z Patrikakis. Cloud federation and the evolution of cloud computing. *Computer*, 49(11) :96–99, 2016.

[83] Onapp federation, accessed Avril, 2021. "http://onapp.com/federation".

[84] Egi federated cloud, accessed Avril, 2021. "http://www.egi.eu/services/cloud-compute/".

[85] Cern.openlab project, accessed Avril, 2021. "http://openlab.web.cern.ch".

[86] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing : state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1) :7–18, 2010.

[87] Benay Kumar Ray, Avirup Saha, Sunirmal Khatua, and Sarbani Roy. Toward maximization of profit and quality of cloud federation : solution to cloud federation formation problem. *The Journal of Supercomputing*, 75(2) :885–929, 2019.

[88] Adel Nadjaran Toosi, Rodrigo N Calheiros, and Rajkumar Buyya. Interconnected cloud computing environments : Challenges, taxonomy, and survey. *ACM Computing Surveys (CSUR)*, 47(1) :7, 2014. http://dx.doi.org/10.1145/2593512.

[89] George Darzanos, Iordanis Koutsopoulos, and George D Stamoulis. Cloud federations : Economics, games and benefits. *IEEE/ACM Transactions on Networking*, 27(5) :2111–2124, 2019.

[90] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. Intercloud : Utility-oriented federation of cloud computing environments for scaling of application services. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 13–31. Springer, 2010.

[91] Misbah Liaqat, Victor Chang, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Toseef, Umar Shoaib, and Rana Liaqat Ali. Federated cloud resource management : Review and discussion. *Journal of Network and Computer Applications*, 77 :87–105, 2017.

[92] Dana Petcu. Portability and interoperability between clouds : challenges and case study. In *European conference on a service-based internet*, pages 62–74. Springer, 2011.

[93] Andy Edmonds, Thijs Metsch, Alexander Papaspyrou, and Alexis Richardson. Toward an open cloud standard. *IEEE Internet Computing*, 16(4) :15–25, 2012.

[94] Open virtualization format, accessed Mai, 2021. "https://www.dmtf.org/standards/ovf".

[95] One interface to rule them all, accessed November, 2020. "http://libcloud.apache.org/".

[96] Grace A Lewis. Role of standards in cloud-computing interoperability. In *2013 46th Hawaii international conference on system sciences*, pages 1652–1661. IEEE, 2013.

[97] Inigo Goiri, Jordi Guitart, and Jordi Torres. Characterizing cloud federation for enhancing providers' profit. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 123–130. IEEE, 2010.

[98] Íñigo Goiri, Jordi Guitart, and Jordi Torres. Economic model of a cloud provider operating in a federated cloud. *Information Systems Frontiers*, 14(4) :827–843, 2012.

[99] Lokesh Chouhan, Pavan Bansal, Bimalkant Lauhny, and Yash Chaudhary. A survey on cloud federation architecture and challenges. In *Social Networking and Computational Intelligence*, pages 51–65. Springer, 2020.

[100] Francisco Luna and Enrique Alba. Parallel multiobjective evolutionary algorithms. In *Springer Handbook of Computational Intelligence*, pages 1017–1031. Springer, 2015.

[101] Nadia Nedjah and Luiza de Macedo Mourelle. Evolutionary multi–objective optimisation : a survey. *International Journal of Bio-Inspired Computation*, 7(1) :1–25, 2015. http://dx.doi.org/10.1504/IJBIC.2015.067991.

[102] Laurent Lemarchand, Damien Massé, Pascal Rebreyend, and Johan Håkansson. Multiobjective optimization for multimode transportation problems. *Advances in Operations Research*, 2018, 2018. http://dx.doi.org/10.1155/2018/8720643.

[103] Anirban Mukhopadhyay, Ujjwal Maulik, Sanghamitra Bandyopadhyay, and Carlos Artemio Coello Coello. A survey of multiobjective evolutionary algorithms for data mining : Part i. *IEEE Transactions on Evolutionary Computation*, 18(1) :4–19, 2014. http://dx.doi.org/10.1109/TEVC.2013.2290086.

[104] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer, 2007. https://link.springer.com/book/10.1007/978-0-387-36797-2.

[105] Efrén Mezura-Montes, Margarita Reyes-Sierra, and Carlos A Coello Coello. Multi-objective optimization using differential evolution : a survey of the state-of-the-art. In *Advances in differential evolution*, pages 173–196. Springer, 2008.

[106] Warisa Wisittipanich and Voratas Kachitvichyanukul. An efficient pso algorithm for finding pareto-frontier in multi-objective job shop scheduling problems. *Industrial Engineering and Management Systems*, 12(2) :151–160, 2013.

[107] Hui Bai, Jinhua Zheng, Guo Yu, Shengxiang Yang, and Juan Zou. A pareto-based many-objective evolutionary algorithm using space partitioning selection and angle-based truncation. *Information Sciences*, 478 :186–207, 2019.

[108] Jay Prakash and TV Vijay Kumar. A multi-objective approach for materialized view selection. In *Research Anthology on Multi-Industry Uses of Genetic Programming and Algorithms*, pages 512–533. IGI Global, 2021.

[109] Hisao Ishibuchi and Tadahiko Murata. A multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 28(3) :392–403, 1998.

[110] Yacov Haimes. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE transactions on systems, man, and cybernetics*, 1(3) :296–297, 1971.

[111] Carlos A Coello. An updated survey of ga-based multiobjective optimization techniques. *ACM Computing Surveys (CSUR)*, 32(2) :109–143, 2000.

[112] J David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the first international conference on genetic algorithms and their applications, 1985*. Lawrence Erlbaum Associates. Inc., Publishers, 1985.

[113] Michael P Fourman. Compaction of symbolic layout using genetic algorithms. In *Proceedings of the 1st international conference on genetic algorithms*, pages 141–153, 1985.

[114] Tapan P Bagchi. Pareto-optimal solutions for multi-objective production scheduling problems. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 458–471. Springer, 2001.

[115] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm : Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2) :182–197, 2002. http://dx.doi.org/10.1109/4235.996017.

[116] David E Golberg. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989(102) :36, 1989.

[117] Kalyanmoy Deb and David E Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the third international conference on Genetic algorithms*, pages 42–50, 1989.

[118] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms : a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4) :257–271, 1999.

[119] CA Coello Coello and Maximino Salazar Lechuga. Mopso : A proposal for multiple objective particle swarm optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 2, pages 1051–1056. IEEE, 2002.

[120] Sancho Salcedo-Sanz. A survey of repair methods used as constraint handling techniques in evolutionary algorithms. *Computer science review*, 3(3) :175–192, 2009. http://dx.doi.org/10.1016/j.cosrev.2009.07.001.

[121] Carlos A Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms : a survey of the state of the art. *Computer methods in applied mechanics and engineering*, 191(11-12) :1245–1287, 2002.

[122] Hemant Kumar and Shiv Prasad Yadav. Fuzzy rule-based reliability analysis using nsga-ii. *International Journal of System Assurance Engineering and Management*, 10(5) :953–972, 2019. https://doi.org/10.1007/s13198-019-00826-5.

[123] A Sathya Sofia and P GaneshKumar. Multi-objective task scheduling to minimize energy consumption and makespan of cloud computing using nsga-ii. *Journal of Network and Systems Management*, 26(2) :463–485, 2018. https://doi.org/10.1007/s10922-017-9425-0.

[124] Xiaolong Xu, Shucun Fu, Yuan Yuan, Yun Luo, Lianyong Qi, Wenmin Lin, and Wanchun Dou. Multiobjective computation offloading for workflow management in cloudlet-based mobile cloud using nsga-ii. *Computational Intelligence*, 35(3) :476–495, 2019. https://doi.org/10.1111/coin.12197.

[125] Hisao Ishibuchi, Naoya Akedo, and Yusuke Nojima. Behavior of multiobjective evolutionary algorithms on many-objective knapsack problems. *IEEE Transactions on Evolutionary Computation*, 19(2) :264–283, 2014.

[126] Hui Li and Qingfu Zhang. Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii. *IEEE transactions on evolutionary computation*, 13(2) :284–302, 2008.

[127] Hemant Kumar and Shiv Prasad Yadav. Nsga-ii based fuzzy multi-objective reliability analysis. *International Journal of System Assurance Engineering and Management*, 8(4) :817–825, 2017.

[128] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. Performance assessment of multiobjective optimizers : An analysis and review. *IEEE Transactions on evolutionary computation*, 7(2) :117–132, 2003.

[129] Nery Riquelme, Christian Von Lücken, and Benjamin Baran. Performance metrics in multi-objective optimization. In *2015 Latin American Computing Conference (CLEI)*, pages 1–11. IEEE, 2015.

[130] Charles Audet, J Bigeon, D Cartier, Sébastien Le Digabel, and Ludovic Salomon. Performance indicators in multiobjective optimization. *Optimization Online*, 2018.

[131] Rahma Bouaziz, Laurent Lemarchand, Frank Singhoff, Bechir Zalila, and Mohamed Jmaiel. Multi-objective design exploration approach for ravenscar real-time systems. *Real-Time Systems*, 54(2) :424–483, 2018.

[132] Lyndon While, Philip Hingston, Luigi Barone, and Simon Huband. A faster algorithm for calculating hypervolume. *IEEE transactions on evolutionary computation*, 10(1) :29–38, 2006.

[133] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms—a comparative case study. In *International conference on parallel problem solving from nature*, pages 292–301. Springer, 1998.

[134] Yaser Mansouri and Rajkumar Buyya. To move or not to move : Cost optimization in a dual cloud-based storage architecture. *Journal of Network and Computer Applications*, 75 :223–235, 2016. https://doi.org/10.1016/j.jnca.2016.08.029.

[135] Richard L Moore, Jim D'Aoust, Robert H McDonald, and David Minor. Disk and tape storage cost models. In *Archiving Conference*, volume 2007, pages 29–32. Society for Imaging Science and Technology, 2007.

[136] Youngjae Kim, Aayush Gupta, Bhuvan Urgaonkar, Piotr Berman, and Anand Sivasubramaniam. Hybridstore : A cost-efficient, high-performance storage system combining ssds and hdds. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pages 227–236. IEEE, 2011. http://dx.doi.org/10.1109/mascots.2011.64.

[137] Ning Zhang, Junichi Tatemura, Jignesh M Patel, and Hakan Hacigümüş. Towards cost-effective storage provisioning for dbmss. *Proceedings of the VLDB Endowment*, 5(4) :274–285, 2011. http://dx.doi.org/10.14778/2095686.2095687.

[138] Amit Kumar Dutta and Ragib Hasan. How much does storage really cost ? towards a full cost accounting model for data storage. In *International Conference on Grid Economics and Business Models*, pages 29–43. Springer, 2013.

[139] Jorge Guerra, Himabindu Pucha, Joseph Glider, Wendy Belluomini, and Raju Rangaswami. Cost effective storage using extent based dynamic tiering. In *Proceedings of the 9th USENIX conference on File and stroage technologies*, pages 20–20, 2011.

[140] Lin Lin, Yifeng Zhu, Jianhui Yue, Zhao Cai, and Bruce Segee. Hot random off-loading : A hybrid storage system with dynamic data migration. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pages 318–325. IEEE, 2011. http://dx.doi.org/10.1109/mascots.2011.41.

[141] Oliver Schiller, Nazario Cipriani, and Bernhard Mitschang. Prorea : live database migration for multi-tenant rdbms with snapshot isolation. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 53–64, 2013.

[142] Gong Zhang, Lawrence Chiu, and Ling Liu. Adaptive data migration in multi-tiered storage based cloud environment. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 148–155. IEEE, 2010.

[143] Zhichao Li, Amanpreet Mukker, and Erez Zadok. On the importance of evaluating storage systems'$ costs. In *6th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 14)*, 2014.

[144] Hamza Ouarnoughi, Jalil Boukhobza, Frank Singhoff, and Stéphane Rubini. A cost model for virtual machine storage in cloud iaas context. In *Parallel, Distributed, and Network-Based Processing (PDP), 2016 24th Euromicro International Conference on*, pages 664–671. IEEE, 2016. http://dx.doi.org/10.1109/pdp.2016.119.

[145] Djillali Boukhelef, Jalil Boukhobza, and Kamel Boukhalfa. A cost model for dbaas storage. In *International Conference on Database and Expert Systems Applications*, pages 223–239. Springer, 2016. http://dx.doi.org/10.1007/978-3-319-44403-1_14.

[146] Makhlouf Hadji and Djamal Zeghlache. Mathematical programming approach for revenue maximization in cloud federations. *IEEE transactions on cloud computing*, 5(1) :99–111, 2017. http://dx.doi.org/10.1109/tcc.2015.2402674.

[147] Zhenyu Wen, Jacek Cała, Paul Watson, and Alexander Romanovsky. Cost effective, reliable and secure workflow deployment over federated clouds. *IEEE Transactions on Services Computing*, 10(6) :929–941, 2017. http://dx.doi.org/10.1109/tsc.2016.2543719.

[148] Linquan Zhang, Chuan Wu, Zongpeng Li, Chuanxiong Guo, Minghua Chen, and Francis CM Lau. Moving big data to the cloud : An online cost-minimizing approach. *IEEE Journal on Selected Areas in Communications*, 31(12) :2710–2721, 2013. http://dx.doi.org/10.1109/jsac.2013.131211.

[149] Wenhua Xiao, Weidong Bao, Xiaomin Zhu, and Ling Liu. Cost-aware big data processing across geo-distributed datacenters. *IEEE Transactions on Parallel and Distributed Systems*, 28(11) :3114–3127, 2017. http://dx.doi.org/10.1109/tpds.2017.2708120.

[150] Yaser Mansouri, Adel Nadjaran Toosi, and Rajkumar Buyya. Cost optimization for dynamic replication and migration of data in cloud data centers. *IEEE Transactions on Cloud Computing*, 2017. http://dx.doi.org/10.1109/tcc.2017.2659728.

[151] Atefeh Khosravi, Lachlan LH Andrew, and Rajkumar Buyya. Dynamic vm placement method for minimizing energy and carbon cost in geographically distributed cloud data centers. *IEEE Transactions on Sustainable Computing*, 2(2) :183–196, 2017. http://dx.doi.org/10.1109/tsusc.2017.2709980.

[152] Kien Le, Ricardo Bianchini, Jingru Zhang, Yogesh Jaluria, Jiandong Meng, and Thu D Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 22. ACM, 2011. http://dx.doi.org/10.1145/2063384.2063413.

[153] Haitao Yuan, Jing Bi, Wei Tan, and Bo Hu Li. Cawsac : Cost-aware workload scheduling and admission control for distributed cloud data centers. *IEEE Transactions on Automation Science and Engineering*, 13(2) :976–985, 2016. http://dx.doi.org/10.1109/tase.2015.2427234.

[154] Makhlouf Hadji, Benjamin Aupetit, and Djamal Zeghlache. Cost-efficient algorithms for critical resource allocation in cloud federations. In *Cloud Networking (Cloudnet), 2016 5th IEEE International Conference on*, pages 1–6. IEEE, 2016. http://dx.doi.org/10.1109/cloudnet.2016.11.

[155] Caesar Wu and Rajkumar Buyya. *Cloud Data Centers and Cost Modeling : A complete guide to planning, designing and building a cloud data center*. Morgan Kaufmann, 2015. https://www.elsevier.com/books/cloud-data-centers-and-cost-modeling/wu/978-0-12-801413-4.

[156] Yuan Xiaoyong, Tang Hongyan, Li Ying, Jia Tong, Liu Tiancheng, and Wu Zhonghai. A competitive penalty model for availability based cloud sla. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 964–970. IEEE, 2015.

[157] Mostafa Mahi, Omer Kaan Baykan, and Halife Kodaz. A new approach based on particle swarm optimization algorithm for solving data allocation problem. *Applied Soft Computing*, 62 :571–578, 2018. https://doi.org/10.1016/j.asoc.2017.11.019.

[158] Kapali P. Eswaran. Placement of records in a file and file allocation in a computer. In Jack L. Rosenfeld, editor, *Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974*, pages 304–307. North-Holland, 1974.

[159] Yizi Wu and Youtao Zhang. Ga based placement optimization for hybrid distributed storage. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 198–203. IEEE, 2015.

[160] Hyun Jin Moon, Yun Chi, and Hakan Hacigümüş. Performance evaluation of scheduling algorithms for database services with soft and hard slas. In *Proceedings of the second international workshop on Data intensive computing in the clouds*, pages 81–90, 2011.

[161] Wanhao Yang and Yan Hu. A replica management strategy based on moea/d. In *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 2154–2159. IEEE, 2018.

[162] Elena Kakoulli and Herodotos Herodotou. Octopusfs : A distributed file system with tiered storage management. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 65–78. ACM, 2017. http://dx.doi.org/10.1145/3035918.3064023.

[163] Lipeng Wan, Zheng Lu, Qing Cao, Feiyi Wang, Sarp Oral, and Bradley Settlemyer. Ssd-optimized workload placement with adaptive learning and classification in hpc environments. In *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–6. IEEE, 2014.

[164] Xin Liu and Kenneth Salem. Hybrid storage management for database systems. *Proceedings of the VLDB Endowment*, 6(8) :541–552, 2013.

[165] Xiyang Liu, Lei Fan, Liming Wang, and Sha Meng. Pso based multiobjective reliable optimization model for cloud storage. In *2015 IEEE International Conference on Computer and Information Technology ; Ubiquitous Computing and Communications ; Dependable, Autonomic and Secure Computing ; Pervasive Intelligence and Computing*, pages 2263–2269. IEEE, 2015.

[166] Lei Jiao, Jun Lit, Wei Du, and Xiaoming Fu. Multi-objective data placement for multi-cloud socially aware services. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 28–36. IEEE, 2014.

[167] Zhenyu Wen, Jacek Cała, Paul Watson, and Alexander Romanovsky. Cost effective, reliable and secure workflow deployment over federated clouds. *IEEE Transactions on Services Computing*, 10(6) :929–941, 2016.

[168] Kwangsung Oh, Abhishek Chandra, and Jon Weissman. Trips : Automated multi-tiered data placement in a geo-distributed cloud environment. In *Proceedings of the 10th ACM International Systems and Storage Conference*, pages 1–11, 2017.

[169] Zhe Wu, Michael Butkiewicz, Dorian Perkins, Ethan Katz-Bassett, and Harsha V Madhyastha. Spanstore : Cost-effective geo-replicated storage spanning multiple cloud services. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 292–308, 2013.

[170] Rashed Salem, Mustafa Abdul Salam, Hatem Abdelkader, and Ahmed Awad Mohamed. An artificial bee colony algorithm for data replication optimization in cloud environments. *IEEE Access*, 8 :51841–51852, 2019.

[171] Pengwei Wang, Caihui Zhao, and Zhaohui Zhang. An ant colony algorithm-based approach for cost-effective data hosting with high availability in multi-cloud environments. In *2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC)*, pages 1–6. IEEE, 2018.

[172] Chunlin Li, YaPing Wang, Hengliang Tang, and Youlong Luo. Dynamic multi-objective optimized replica placement and migration strategies for saas applications in edge cloud. *Future Generation Computer Systems*, 100 :921–937, 2019.

[173] Reza Salkhordeh, Hossein Asadi, and Shahriar Ebrahimi. Operating system level data tiering using online workload characterization. *The Journal of Supercomputing*, 71(4) :1534–1562, 2015.

[174] Woon-Hak Kang, Sang-Won Lee, and Bongki Moon. Flash as cache extension for online transactional workloads. *The VLDB Journal*, 25(5) :673–694, 2016.

[175] Sai-Qin Long, Yue-Long Zhao, and Wei Chen. Morm : A multi-objective optimized replication management strategy for cloud storage cluster. *Journal of Systems Architecture*, 60(2) :234–244, 2014.

[176] Li Renhou LuoYinsheng, Zhang Lei, and Liu Fang. Application of artificial immune algorithm to function optimization. *Journal of Xi'an Jiaotong University*, 8(3) :840–843, 2003.

[177] Cplex optimizer, accessed December, 2018. "https://www.ibm.com/fr-fr/analytics/cplex-optimizer".

[178] Quanlu Zhang, Shenglong Li, Zhenhua Li, Yuanjian Xing, Zhi Yang, and Yafei Dai. Charm : A cost-efficient multi-cloud data hosting scheme with high availability. *IEEE Transactions on Cloud computing*, 3(3) :372–386, 2015.

[179] Chandrakant D Patel and Amip J Shah. Cost model for planning, development and operation of a data center. *Hewlett-Packard Laboratories Technical Report*, 107 :1–36, 2005.

[180] Amina Chikhaoui, Kamel Boukhalfa, and Jalil Boukhobza. A cost model for hybrid storage systems in a cloud federations. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1025–1034. IEEE, 2018.

[181] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D Davis, Mark S Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *USENIX Annual Technical Conference*, volume 57. Boston, USA, 2008.

[182] Amazon compute service level agreement, accessed November, 2021. "https://aws.amazon.com/compute/sla/".

[183] Rahul Garg, Huzur Saran, Ramandeep Singh Randhawa, and Manpreet Singh. A sla framework for qos provisioning and dynamic capacity allocation. In *IEEE 2002 Tenth IEEE International Workshop on Quality of Service (Cat. No. 02EX564)*, pages 129–137. IEEE, 2002.

[184] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software : Practice and experience*, 41(1) :23–50, 2011. http://dx.doi.org/10.1002/spe.995.

[185] Amina Chikhaoui, Laurent Lemarchand, Kamel Boukhalfa, and Jalil Boukhobza. Multi-objective optimization of data placement in a storage-as-a-service federated cloud. *ACM Transactions on Storage (TOS)*, 17(3) :1–32, 2021.

[186] Amazon cloudwatch, accessed November, 2020. "https://aws.amazon.com/fr/cloudwatch/".

[187] Openstack watcher project, accessed November, 2020. "https://wiki.openstack.org/wiki/Watcher".

[188] Mohammed Islam Naas, François Trahay, Alexis Colin, Pierre Olivier, Stéphane Rubini, Frank Singhoff, and Jalil Boukhobza. Eziotracer : unifying kernel and user space i/o tracing for data-intensive applications. In *Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems*, pages 1–11, 2021.

[189] Masoud Saeida Ardekani and Douglas B Terry. A self-configurable geo-replicated cloud storage system. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 367–381, 2014.

[190] Stefan Voss, V Maniezzo, and T Stützle. Matheuristics : Hybridizing metaheuristics and mathematical programming (annals of information systems). 2009.

[191] Mohammad Hamdan. On the disruption-level of polynomial mutation for evolutionary multi-objective optimisation algorithms. *Computing and Informatics*, 29(5) :783–800, 2012.

[192] Kalyanmoy Deb, Ram Bhushan Agrawal, et al. Simulated binary crossover for continuous search space. *Complex systems*, 9(2) :115–148, 1995.

[193] Moea framework. http://moeaframework.org/.

[194] Deskstar 7k1000.d, accessed May, 2020. "https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/product/hgst/deskstar-7k-series/data-sheet-deskstar-7k1000-d.pdf".

[195] Samsung ssd 850 pro, accessed May, 2020. "https://images-eu.ssl-images-amazon.com/images/I/81tHEOsrKRS.pdf".

[196] Amazon ebs volume types, accessed January, 2022. "https://aws.amazon.com/ebs/volume-types/".

[197] Amazon data transfer, accessed May, 2020. "https://aws.amazon.com/s3/pricing/".

[198] Jörn Altmann and Mohammad Mahdi Kashef. Cost model based service placement in federated hybrid clouds. *Future Generation Computer Systems*, 41 :79–90, 2014. https://doi.org/10.1016/j.future.2014.08.014.

[199] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[200] Brunelle Alan D. blktrace user guide. 2008.

[201] Amina Chikhaoui, Laurent Lemarchand, Kamel Boukhalfa, and Jalil Boukhobza. Stornir, a multi-objective replica placement strategy for cloud federations. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 50–59, 2021.

[202] Najme Mansouri and Mohammad Masoud Javidi. A review of data replication based on meta-heuristics approach in cloud computing and data grid. *Soft Computing*, pages 1–28, 2020.

[203] Hdfs architecture guide, accessed September, 2020. "https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html".

[204] Joel Chacón and Carlos Segura. Analysis and enhancement of simulated binary crossover. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018.

[205] Carlos A Coello Coello, Gregorio Toscano Pulido, and M Salazar Lechuga. Handling multiple objectives with particle swarm optimization. *IEEE Transactions on evolutionary computation*, 8(3) :256–279, 2004.

[206] Miao Zhang, Huiqi Li, Li Liu, and Rajkumar Buyya. An adaptive multi-objective evolutionary algorithm for constrained workflow scheduling in clouds. *Distributed and Parallel Databases*, 36(2) :339–368, 2018.

[207] Samsung v-nand ssd 970 evo plus, accessed May, 2020. "https://semiconductor.samsung.com/resources/data-sheet/Samsung_NVMe_SSD_970_EVO_Plus_Data_Sheet_Rev.3.0.pdf".

[208] Ebs magnetic volumes, accessed December, 2020. "https://aws.amazon.com/ebs/previous-generation/".

[209] Marcio Roberto Miranda Assis and Luiz Fernando Bittencourt. Avoiding free riders in the cloud federation highways. In *CLOSER*, pages 169–179, 2017.

[210] Praveen Khethavath, Johnson Thomas, Eric Chan-Tin, and Hong Liu. Introducing a distributed cloud architecture with efficient resource discovery and optimal resource allocation. In *2013 IEEE Ninth World Congress on Services*, pages 386–392. IEEE, 2013.

[211] Lena Mashayekhy, Mahyar Movahed Nejad, and Daniel Grosu. Cloud federations in the sky : Formation game and mechanism. *IEEE Transactions on Cloud Computing*, 3(1) :14–27, 2014.

[212] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[213] Bayya Yegnanarayana. *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[214] Mohammed Islam Naas, Laurent Lemarchand, Philippe Raipin, and Jalil Boukhobza. Iot data replication and consistency management in fog computing. *Journal of Grid Computing*, 19(3) :1–25, 2021.

[215] Mohamed Handaoui, Jean-Emile Dartois, Laurent Lemarchand, and Jalil Boukhobza. Salamander : a holistic scheduling of mapreduce jobs on ephemeral cloud resources. In *2020*

20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), pages 320–329. IEEE, 2020.

[216] Mohamed Handaoui, Jean-Emile Dartois, Jalil Boukhobza, Olivier Barais, and Laurent d'Orazio. Releaser : A reinforcement learning strategy for optimizing utilization of ephemeral cloud resources. In *2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 65–73. IEEE, 2020.

[217] Jean-Emile Dartois, Heverson B Ribeiro, Jalil Boukhobza, and Olivier Barais. Cuckoo : Opportunistic mapreduce on ephemeral and heterogeneous cloud resources. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 396–403. IEEE, 2019.

[218] Jean-Emile Dartois, Jalil Boukhobza, Anas Knefati, and Olivier Barais. Investigating machine learning algorithms for modeling ssd i/o performance for container-based virtualization. *IEEE transactions on cloud computing*, 2019.

[219] Ta-Yuan Hsu and Ajay D Kshemkalyani. A proactive, cost-aware, optimized data replication strategy in geo-distributed cloud datastores. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, pages 143–153, 2019.

[220] Abdullah Algarni and Daniel Kudenko. Distribution data across multiple cloud storage using reinforcement learning method. In *ICAART (2)*, pages 431–438, 2017.

[221] Zhanyang Xu, Dawei Zhu, Jinhui Chen, and Baohua Yu. Splitting and placement of data-intensive applications with machine learning for power system in cloud computing. *Digital Communications and Networks*, 2021.

[222] Sunirmal Khatua, Anirban Ghosh, and Nandini Mukherjee. Optimizing the utilization of virtual resources in cloud environment. In *2010 IEEE International Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems*, pages 82–87. IEEE, 2010.

[223] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, James Majors, Adam Manzanares, and Xiao Qin. Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In *2010 IEEE international symposium on parallel & distributed processing, workshops and Phd forum (IPDPSW)*, pages 1–9. IEEE, 2010.

[224] Wuhui Chen, Baichuan Liu, Incheon Paik, Zhenni Li, and Zibin Zheng. Qos-aware data placement for mapreduce applications in geo-distributed data centers. *IEEE Transactions on Engineering Management*, 68(1) :120–136, 2020.

[225] Yongqiang He, Rubao Lee, Yin Huai, Zheng Shao, Namit Jain, Xiaodong Zhang, and Zhiwei Xu. Rcfile : A fast and space-efficient data placement structure in mapreduce-based warehouse systems. In *2011 IEEE 27th International Conference on Data Engineering*, pages 1199–1208. IEEE, 2011.

NOTATIONS

Notations utilisées dans le chapitre 4

Le tableau 1 présente les notations utilisées dans le chapitre 4 pour la modélisation du coût de placement.

Symbole	Description
Fédération	
$\boldsymbol{F}$	Fédération d'un ensemble de Cloud Service Provider (CSP)
$\boldsymbol{CSP^d}$	fournisseur $\boldsymbol{d}$
$\boldsymbol{Cap_{max_{rsc}}}$, $\boldsymbol{Cap_{idl_{rsc}}}$,$\boldsymbol{p_{rsc}}$	Capacités totales et inactives de la ressource de stockage $\boldsymbol{rsc}$ et son prix
$\boldsymbol{F^d_{rsc}}$	Prix d'internalisation de la ressource $\boldsymbol{rsc}$ de $\boldsymbol{CSP^d}$
Clients	
$\boldsymbol{U_{int}}$, $\boldsymbol{u_k}$	Ensemble des clients internes et le client $\boldsymbol{kth}$
$\boldsymbol{U_{ext}}$, $\boldsymbol{u^{d'}_{k'}}$	L'ensemble des clients externes et le client $\boldsymbol{k'}$th
$\boldsymbol{wl_k}$	Charge de travail de $\boldsymbol{u_k}$
$\boldsymbol{iops_{sla,k}}$, $\boldsymbol{ltc_{sla,k}}$	Performances demandées en termes d'IOPS et de latence
$\boldsymbol{pn_k}$, $\boldsymbol{pn_{iops,k}}$, $\boldsymbol{pn_{ltc,k}}$	Fonction de penalité et ses parties : pénalité d'Input/Output Operations Per Second (IOPS) et pénalité de latence
$\boldsymbol{iops_{offered}(u_k)}$	IOPS offertes au client $\boldsymbol{u_k}$
$\boldsymbol{ltc_{offered}(u_k)}$	Latence offerte au client $\boldsymbol{u_k}$
Stockage	
$\boldsymbol{SC}$, $\boldsymbol{sc_j}$	Ensemble de classes de stockage, $\boldsymbol{j^{ème}}$ classe
$\boldsymbol{sc_{j,int}}$, $\boldsymbol{sc_{j,ext}}$	Parties de stockage des clients internes et externes
$\boldsymbol{c_{sc_j}}$, $\boldsymbol{p_{sc_j}}$, $\boldsymbol{wo_{sc_j}}$	Papacité, prix et endurance de $\boldsymbol{sc_j}$
Bande passantee	
$\boldsymbol{bw}$, $\boldsymbol{p_{bw}}$	Bande passante et son prix d'achat
$\boldsymbol{bw_{int}}$,$\boldsymbol{bw_{ext}}$	Bandes passantes des clients internes et externes

Objets	
$\boldsymbol{N}$, $\boldsymbol{o_{i,k}}$, $\boldsymbol{s_{o_{i,k}}}$	$\boldsymbol{i^{ème}}$ Nombre total d'objets, objet du client $\boldsymbol{u_k}$ et sa taille
$\boldsymbol{req_{op,o_{i,k}}}$	IOPS moyen de type $\boldsymbol{op}$ émis vers l'objet $\boldsymbol{o_{i,k}}$
Général	
$\boldsymbol{T}$	Periode de temps
$\boldsymbol{T_{sp}}$	La période d'abonnement internet
Coûts	
$\boldsymbol{Cost_{plc,T}}$	Cout de placement totale
$\boldsymbol{Cost_{plc_{int},T}}$	Cout de placement des objets des clients internes
$\boldsymbol{Cost_{plc_{ext},T}}$	Cout de placement des objets des clients externes
$\boldsymbol{Cost_{lcl_{int},T}}$	Cout de placement local des clients internes
$\boldsymbol{Cost_{lcl_{ext},T}}$	Cout de placement local des clients externes
$\boldsymbol{Cost_{pnt_{int},T}}$	Cout de pénalité des clients internes
$\boldsymbol{Cost_{pnt_{ext},T}}$	Cout de pénalité des clients externes
$\boldsymbol{Cost_{pnt_{iops},T}}$	Cout de pénalité d'IOPS
$\boldsymbol{Cost_{pnt_{ltc},T}}$	Cout de pénalité de latence
$\boldsymbol{Cost_{stg_{int},T}}$	Cout de stockage des objets des clients internes
$\boldsymbol{Cost_{stg_{ext},T}}$	Cout de stockage des objets des clients externes
$\boldsymbol{Cost_{mgr_{int},T}}$	Cout de migration interne des objets des clients internes
$\boldsymbol{Cost_{mgr_{ext},T}}$	Cout de migration interne des objets des clients externes
$\boldsymbol{Cost_{out_{src},T}}$	Cout d'externalisation
$\boldsymbol{Cost_{in_{src},T}}$	Cout d'internalisation
$\boldsymbol{Cost_{geo_{mgr},T}}$	Cout de géo-migration
$\boldsymbol{Cost_{bck_{mig},T}}$	Cout de rapatriement
$\boldsymbol{Cost_{re_{mgr},T}}$	Cout de renvoi
$\boldsymbol{Cost_{geo_{rpl},T}}$	Cout de geo-replication
$\boldsymbol{Cost_{add_{rep},T}}$	Cout d'ajout d'une réplique
$\boldsymbol{Cost_{syn_{rep},T}}$	Cout de synchronisation des repliques
$\boldsymbol{Cost_{rd,T}}$	Cout de l'opération de lecture locale
$\boldsymbol{Cost_{rd_{erg},T}}$	Cout d'energie de l'opération de lecture
$\boldsymbol{Cost_{rd_{edr},T}}$	Cout d'endurance de l'opération de lecture
$\boldsymbol{Cost_{wr,T}}$	Cout de l'opération d'écriture

$\boldsymbol{Cost_{wr_{erg},T}}$	Cout d'energie de l'opération d'écriture
$\boldsymbol{Cost_{wr_{edr},T}}$	Cout d'endurance de l'opération d'écriture
$\boldsymbol{Cost_{bw,T}}$	Coût de la bande passante Internet consommée localement
$\boldsymbol{Cost_{bw_{amz},1}}$	Bande passante internet amortie sur une unité de temps
$\boldsymbol{Cost_{ntw_{out},T}}$	Coût du réseau du trafic sortant externe
$\boldsymbol{Cost_{ext_{occ},T}}$	Cout d'occupation des externe
$\boldsymbol{Cost_{ext_{wld},T}}$	Cout de la charge de travail externe
$\boldsymbol{Cost_{ext_{plt},T}}$	Cout de penalité externe

Tableau 1 – Notations utilisées dans le modèle de coût

Notations utilisées dans les chapitres 5 et 6

Comme la plupart des notations utilisées dans le chapitre 5 sont utilisées dans le chapitre 6, nous les avons fusionné dans le tableau 2.

Symbole	Description
M, Z_m	Le nombre de zones géographiques, la zone m
D	Le nombre de CSP dans la fédération
CSP_d, ss_d,css_d, io_d	Le fournisseur d, son service de stockage, la capacité du service (GB) et ses performances en terme d'IOPS
J	Le nombre de classes de stockage local
sc_j, csc_j, $io_j(op)$	La classe de stockag j, sa capacité et ses IOPS par type d'opération op
op	($op \in \{rr, rw, sr, sw\}$) où rr : lecture aléatoire, rw : écriture aléatoire, sr : lecture séquentielle, sw : écriture séquentielle
R, r	Le nombre de types de requêtes, le type de requête r (par exemple, get, put)
K	Le nombre de clients
U, u_k, O_k	L'ensemble de clients, le client k et l'ensemble de ses objets
$ioHard_k$,$ioSoft_k$	SLA hard et soft en terme d'IOPS du client u_k
$ioOffered_k$	Les IOPS livrés au client u_k
$o_{i,k}$,$s_{o_{i,k}}$, $io_{o_{i,k}}(op)$	L'object i du client u_k, sa taille et ses requêtes d'E/S émises par type d'opération
P	Le nombre d'emplacements de stockage possibles, $P = D + J - 1$
$tot_k^{m,o_{i,k}}$	le nombre total de requêtes du client u_k émises depuis la zone Z_m et qui accèdent à l'objet $o_{i,k}$
$w_{r,z}^k$	Le nombre de requêtes de type r pour la charge de travail du client k envoyées depuis la zone z
$l_{m,x_{o_{i,k}}}$	La latence du réseau entre la zone Z_m et l'emplacement de l'objet $o_{i,k}$.
T	Période de temps
x	La variable de décision représentant une configuration de placement
$store$	Le coût de stockage
$migrate$	Le coût de migration
$latency$	Le coût de latence

Tableau 2 – Notations utilisées les chapitres 5 et 6

ملخص

يسمح اتحاد السحابات امكانية توسيع موارد مزودي الخدمات السحابية بسلاسة من أجل توفير خدمات ذات جودة أفضل للعملاء بدون تكاليف اضافية.

تعتبر خدمة التخزين أحد الخدمات السحابية الرئيسية. بالنسبة لمثل هذه الخدمة، يعد أداء الإدخال/الإخراج وزمن استجابة الشبكة من بين أهم المقاييس التي يأخذها العملاء بعين الاعتبار. في الواقع ، تقضي بعض عمليات استعلامات قاعدة البيانات 90% من وقت التنفيذ في عمليات الإدخال / الإخراج. من جهة أخرى،، تقوم بعض الشركات السحابية بتقديم ضمانات خاصة بزمن الاستجابة في اتفاقية مستوى الخدمة " Service Level Agreements " ويمكن للعملاء دفع رسوم إضافية لتقليل أزمنة الاستجابة بشكل أكبر.

تتناول هذه الأطروحة مشكلة وضع البيانات " Data placement " بالنسبة لمزود خدمات ينتمي إلى اتحاد سحابات. في الواقع ، تقديم خدمات جذابة وغير مكلفة يعتبر تحديًا كبيرًا لمزود الخدمات. هدفنا هو توفير استراتيجيات ذكية لتحسين عملية وضع البيانات من شأنها تقليل التكلفة للمزود مع تلبية متطلبات جودة الخدمة للعملاء. يجب أن تأخذ هذه الاستراتيجيات في الاعتبار عدم تجانس موارد التخزين الداخلية والخارجية من حيث السعة، الأداء والتسعير وكذلك خصائص العملاء ومتطلباتهم. على الرغم من وجود العديد من استراتيجيات وضع البيانات الخاصة بأنظمة التخزين الهجينة، إلا أنها غير قابلة للتعميم على كل بنية. في الواقع ، يجب تصميم الاستراتيجيات وفقًا لبنية النظام المدروس والأهداف المنشودة.

تعتمد استراتيجيات التحسين في البيئات السحابية بشكل عام على التكلفة. في الواقع ، يعد خفض التكاليف التشغيلية مع الحفاظ على مستويات عالية من الخدمة للعملاء من العوامل المهمة لمزودي الخدمات لزيادة إيراداتهم والحفاظ على قدرتهم التنافسية. وبالتالي ، فإن مساهمتنا الأولى

تتعلق بتقييم تكلفة وضع البيانات لمزود خدمات ينتمي الى اتحاد سحابات. يأخذ النموذج المقترح في الاعتبار العوامل المختلفة التي تشكل الخصائص الفيزيائية والوظيفية للنظام المتحد.

من أجل تحسين وضع البيانات ، يجب مراعاة تكاليف التخزين، الترحيل وزمن الاستجابة. هذه التكاليف مترابطة ومتناقضة في بعض الحالات. ومن ثم فمن الضروري إيجاد حل وسط بينها. و بالتالي، يعد التحسين متعدد الأهداف ضروريًا. في هذا السياق ، تتمثل مساهمتنا الثانية في اقتراح استراتيجية متعددة الأهداف تعتمد على الخوارزمية الجينية للفرز الوراثي التطوري ، الإصدار الثاني "NSGAII" . قمنا بتطوير IR CDP-NSGAII " Constraint Data Placement matheuristic based on NSGAII with Injection and Repair functions " . هي عبارة عن تهجين طريقة دقيقة تعتمد على البرمجة الخطية و NSGAII . تم تصميم دالة إصلاح هدفها جعل الحلول تحترم قيود النظام.

أخيرًا ، المساهمة الثالثة في هذا البحث تتعلق بتكرار البيانات " data replication " و ذلك بخلق نسخ متماثلة. في الواقع ، يعد توفير خدمات موثوقة مع توافر عالي للبيانات وأداء مناسب من المتطلبات الرئيسية التي يعنى بها العميل. يتم استخدام مفهوم التكرار لضمان مثل هذه المتطلبات. لحل مشكلة وضع النسخ المتماثلة و توزيع الاعمال، اقترحنا StorNIR " a cost-efficient data object Storing scheme based on NSGAII upgraded with Injection and Reparation operators " . تم تناول نفس المنطق للمساهمة الثانية في هذا العمل. في الواقع، مزجنا طريقة دقيقة مع NSGAII لحل مشكلة وضع واختيار نسخ البيانات لتنفيذ الأعمال. تجدر الإشارة إلى أننا قمنا بتحديث عوامل التطور لـ NSGAII لتكييفهم مع مشكلة النسخ المتماثلة. بالإضافة إلى ذلك ، عندما يتم نسخ البيانات، يمكن تحسين زمن استجابة الشبكة. لذلك، اقترحنا خوارزمية لتحسين هذا الزمن تتمثل في تعيين الطلبات إلى أقرب موقع يحترم أداء الإدخال/الإخراج للتخزين .

DOCTORAT BRETAGNE LOIRE / MATHSTIC

Université de Bretagne Occidentale

جامعة هواري بومدين للعلوم والتكنولوجيا
USTHB

Titre : Vers une approche intelligente de placement de données dans un Cloud distribué basé sur un système de stockage hybride

Mots clés : système de stockage hybride

Résumé : La fédération de Clouds permet d'´etendre de manière transparente les ressources des fournisseurs de services Cloud (Cloud service Providers: CSP) afin de fournir une meilleure qualité de service (Quality of Service: QoS) aux clients sans frais de d ´eploiement suppl´ementaires. Le stockage en tant que service (Storage as a Service: StaaS), constitue l'un des principaux services Cloud offerts aux clients. Pour un tel service, la performance des Entr´ees/Sorties (E/S) des supports de stockage et la latence du réseau sont parmi les métriques les plus importantes considérées par les clients. En effet, le système d'E/S prend environ 90% du temps d'exécution d'une transaction pour certaines requêtes de base de données. Afin de rassurer les clients, certaines sociétés de Cloud incluent déjà des garanties de latence dans leurs contrats de niveau de service (Service Level Agreements: SLA) et les clients peuvent payer des frais supplémentaires pour réduire davantage les latences. Cette thèse traite du problème de placement de données pour un CSP faisant partie d'une fédération.
En effet, offrir des services attractifs et peu coûteux est un grand défi pour les CSP. Notre but est de proposer des approches intelligentes pour un meilleur placement des données qui minimise le coût pour le fournisseur tout en satisfaisant les clients. Cette approche doit prendre en compte l'hétérogénéité des ressources de stockage interne et externe en terme de plusieurs paramètres (comme la capacité, les performances, la tarification) ainsi que les caractéristiques des clients et leurs exigences.

Title : Towards an intelligent approach for data placement in a distributed Cloud based on a hybrid storage system

Keywords : Cloud, hybrid storage system

Abstract : Cloud federation makes it possible to seamlessly extend the resources of Cloud Service Providers (CSP) in order to provide a better Quality of Service (QoS) to customers without addi-tional deployment costs. Storage as a Service (StaaS), is one of the main Cloud services offered to customers. For such a service, storage Input/ Output (I/O) performance and network latency are among the most important metrics considered by customers. In effect, transactions for some database queries spend 90% of the execution time in I/O operations. In order to satisfy customers, some Cloud companies already include latency guarantees in their Service Level Agreements (SLA) and customers can pay additional fees to further reduce latency. This thesis addresses the data placement problem for a CSP that is part of a federation. services is a big challenge for CSP. Indeed, offering attractive and inexpensive. Our goal is to pro-vide intelligent approaches for a better data placement that minimizes the cost of placement for the provider while satisfying the customers QoS requirements. This approach must take into account the heterogeneity of internal and external storage resources in terms of several parameters (such as capacity, performance, pricing) as well as customer characteristics and requirements. Despite the fact that many data placement strategies have been proposed for hybrid storage systems, they are not generalizable to every architecture. Indeed, a placement strategy must be designed according to the system architecture for which it is proposed and the target objectives.