N° d'ordre: 86/2019 - C/INF

République algérienne démocratique et populaire
Ministère de l'enseignement supérieur et de la recherche
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'électronique et d'informatique

THÈSE DE DOCTORACT

PRESENTÉ POUR OBTENIR LE GRADE DE DOCTEUR

EN : INFORMATIQUE

Spécialité : **Systèmes informatiques**

Présentée et soutenue par

Djillali BOUKHELEF

VERS UNE METHODOLOGIE D'OPTIMISATION DU PLACEMENT DES OBJETS DBAAS DANS UN ENVIRONNEMENT DE CLOUD COMPUTING

Soutenu publiquement le 29/09/2019 devant les jurys :

Mme. ALIMAZIGHI Zaia	Professeur à l'USTHB	Présidente
M. BOUKHALFA Kamel	Professeur à l'USTHB	Directeur de thèse
Mme. NACER Hassina	Maitre de conférence/A à l'USTHB	Examinatrice
M. ZITOUNI Abdelhafid	Professeur à l'U. Constantine 2	Examinateur
M. HIDOUCI Walid-Khaled	Professeur à l'ESI	Examinateur
M. BOUKHOBZA Jalil	Maitre de conférence/A à l'UBO Brest France	Invité

Résumé

Les opérateurs de *cloud DBaaS* (*Database as a Service*) proposent à leurs clients des bases de données sous forme d'un service. L'explosion du marché du cloud les a contraints à optimiser très finement l'utilisation de leurs centres de données afin de proposer des services attractifs à moindre coût. Dans cette perspective, plusieurs technologies ont été utilisées pour optimiser les coûts tels que le coût de stockage (e.g. l'utilisation des systèmes de stockage hybride), le coût de calcul (e.g. la virtualisation) et le coût de maintenance (e.g. les applications multi-locataire). Dans ce thèse, nous nous intéressons au système de stockage hybride HDD-SSD. En effet, les disques durs magnétiques (HDD) sont des supports de stockage à la fois énergivores et peu performants comparés aux unités de calcul. Néanmoins, leur prix par gigaoctet et leur longévité peuvent jouer en leur faveur. Contrairement aux HDD, les disques SSD à base de mémoire flash sont plus performants et consomment peu d'énergie. Leur prix élevé par gigaoctet et leur courte durée de vie (comparés aux HDD) représentent leurs contraintes majeures. De ce fait, les chercheurs ont suggéré l'utilisation de systèmes de stockage hybride HDD-SSD afin d'avoir les performances des SSD au prix des HDD.

Cependant, l'adoption d'une telle technologie sans l'utilisation d'une stratégie de placement des objects adaptée pour les système de stockage hybride pourra seulement augmenter les coûts sans forcément aboutir à amélioration des performances. Il est donc nécessaire de concevoir des stratégies de placement qui gèrent le placement des objects et décident où et quand doit-on placer un objet dans une classe de stockage. Ces stratégies doivent être capables de gérer le mouvement des données entre les différentes classes de stockage en cas de fluctuation de la charge de travail. Nous prenons dans ce travail les bases de données relationnelles comme cas d'étude. Toutefois, nos approches sont applicables pour les bases de données NoSQL.

Dans cette thèse nous introduisons un nouveau modèle autonomique d'optimisation de placement des objets de DBaaS inspiré de MAPE-K (*Monitor, Analyze, Plan, Execute, Knwoledge*), et prenant en compte en plus des caractéristiques du Cloud, les E/S des objets ainsi que les systèmes de stockage associés. Notre première contribution consiste à proposer un modèle de coût étendu dont l'originalité consiste à prendre en compte le profil d'accès des objets, les caractéristiques du système de stockage, ainsi que les contraintes économiques de l'environnement cloud. Notre deuxième contribution consiste à proposer une stratégie de placement des objets de DBaaS dans un contexte de Cloud. Deux stratégies ont été proposées dans ce sens : La première, appelée Genetic Cost Based Object Placement Strategie (G-COPS), est basée sur une métaheuristique à savoir un algorithme génétique (GA), la deuxième approche est basée sur un algorithme

heuristique incrémental ad-hoc appelée Heuristic Cost Based Object Placement Strategie (H-COPS).

Abstract

DBaaS cloud providers offer the database as service. The growth and highly competitive nature of this economy has compelled them to optimize the use of their data centers, in order to offer attractive services at a lower cost. From this perspective, Several technologies and techniques have been designed to optimize costs such as the storage cost (e.g. the use of hybrid storage systems), the computing cost (e.g. virtualization) and the maintenance cost (e.g. multi-tenant). In this thesis, we are interested in hybrid storage system HDD-SSD. Indeed, Hard Disk Drives (HDD) represent energy-intensive and inefficient devices compared to compute units. However, their low cost per gigabyte and their long lifetime may constitute positive arguments. Unlike HDD, flash-based Solid-State Disks (SSD) are more efficient and consume less power, but their high cost per gigabyte and their short lifetime (compared to HDD) represent major constraints. The idea is to combine SSD and HDD seeking the performance of SSDs with the price of HDDs.

However, adopting such a technology without the use of a object placement strategy suitable for hybrid storage systems will only increase the costs without necessarily leading to improved performance. It is therefore necessary to design placement strategies that manage the placement of objects and decide where and when to place an object in a storage class. These strategies must be able to manage the movement of data between different storage classes in the case of workloads fluctuating. In this work, we take relational databases as a case study. However, our approaches are applicable for NoSQL databases.

In this thesis we propose a new autonomic model for object placement optimization based on MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) whereby in addition to the key aspects of Cloud computing paradigm, the Object I/O and related storage systems are considered. Our first contribution consist to propose a cost model which takes into account the object I/O profile, the storage system characteristics, and the cloud environment constraints. the second contribution consist to propose a Cost based Object Placement Strategies (COPS) on HSS for relational cloud DBMSs. We propose two strategies. The first one is a meta-heuristic approach based on a genetic algorithm (G-COPS). As meta-heuristic algorithms are not always the best answer for optimization problems, we also propose a specialized heuristic-based solution (H-COPS). The idea of H-COPS consists in computing a first object placement solution and enhancing it incrementally based on the characteristics of the I/O workloads and customer penalties.

Remerciements

Au terme de ce modeste travail, je voudrais adresser mes vifs remerciements à mes parents et à toutes les personnes qui ont contribué, de près ou de loin, à accomplir ce présent travail.

je tiens d'emblée à remercier mon directeur de thèse Mr. Kamel BOUKHALFA pour m'avoir accordé l'honneur de travailler avec lui et de m'avoir supporté pendant toute cette période de thèse. Son soutien, ses conseils, ses encouragements ainsi que ses remarques très pointues m'ont certainement permis de mener à bien mon travail de thèse.

J'exprime ma profonde gratitude à Mr Jalil BOUKHOBZA, enseignant à l'UBO, Brest, France, pour son accueil au sein du laboratoire, pour sa disponibilité, pour le temps précieux qu'il m'a accordé et pour ses observations qui ont contribué à améliorer la qualité de ce travail.

Je tiens à remercier très sincèrement Pr.ALIMAZIGHI Zaia, Pr. ZITOUNI Abdelhafid, Pr. HIDOUCI Walid-Khaled et Dr. NACER Hassina qui me font le grand honneur d'accepté de juger mon travail.

Dédicace

A ma chère famille, en particulier mes parents qui ont sus me supporter et encourager tout au long de ma vie et qui ont fait de moi ce que je suis aujourd'hui.
A mes amis de tout temps et tout bord, à ceux et celles qui ont su trouver les mots pour m'aider à franchir les obstacles et à avancer dans la vie.

A ...
Mon épouse pour son soutien irremplaçable.

PUBLICATIONS

Revues

Djillali Boukhelef, Jalil Boukhobza, Kamel Boukhalfa, Hamza Ouarnoughi, Laurent Lemarchand : "**Optimizing the cost of DBaaS object placement in hybrid storage systems**", *Future Generation Computer Systems*, Volume 93, Avril 2019, Pages 176-187.
DOI : 10.1016/j.future.2018.10.030
URL : https ://www.sciencedirect.com/science/article/pii/S0167739X17325785 ?via%3Dihub

Conférences internationales

1. Djillali Boukhelef, Jalil Boukhobza, Kamel Boukhalfa, Hamza Ouarnoughi, Laurent Lemarchand : "**COPS : Cost Based Object Placement Strategies on Hybrid Storage System for DBaaS Cloud**", In *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2017)*, Pages 659-664 ,madrid, espagne, Mai 2017 .
   DOI : 10.1109/CCGRID.2017.36
   URL : https ://ieeexplore.ieee.org/document/7973754

2. Djillali Boukhelef, Jalil Boukhobza, Kamel Boukhalfa : "**A Cost Model for DBaaS Storage**", In *27th The International Conference on Database and Expert Systems Applications (DEXA 2016)*, Pages 223-239 , Porto, Portugal, Septembre 2016 . **DOI :** 10.1007/978-3-319-44403-1_14
   URL : https ://link.springer.com/chapter/10.1007/978-3-319-44403-1_14

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

ACRONYMES

Cloud : Cloud Computing

CPU : Central Processing Unit

DBaaS : Database As s Service

EEPROM : Electrically Erasable Programmable Read-Only Memory

E/S : Entrée/Sortie

Go : GigaOctet

HDD : Hard Disk Drive

HSS : Hybrid Storage System

I/O : Input/Output

IOPS : Input/Output Operations Per Second

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

PM : Physical Machine

RAM : Random Acess Memory

RR : Random Read

RW : Random write

SLA : Server Level Agreement

SR : sequential Read

SSD : Solid-State Drive

SW : sequential write

VM : Virtual Machine

Chapitre 1

INTRODUCTION

Sommaire

1.1 CONTEXTE GÉNÉRAL

Le *Cloud Computing* est un modèle d'hébergement, de fourniture et d'accès aux services informatique à travers l'internet [133]. Il permet un accès pratique, omniprésent, et à la demande à un ensemble de ressources de calcul partagées et configurables (e.g. réseaux, serveurs, stockage, applications, et services). Ces ressources peuvent être acquises et libérées avec un minimum d'effort de gestion. Cette idée n'est pas tout à fait nouvelle. Elle rappelle des concepts d'informatique utilitaire proposés par John McCarthy en 1961 lors d'une conférence au Massachusetts Institute of Technology (MIT) : « *Computation may someday be organized as a public utility* » [48, 133]. Néanmoins, les technologies n'étaient pas matures pour incarner l'idée et démocratiser un tel service [64].

Toutefois, avec le développement de la technologie de virtualisation et l'évolution de l'internet, du web, des systèmes de stockage et des capacités de traitement, il est devenu possible d'offrir l'informatique comme un service [23]. De nos jours, plusieurs entreprises proposent des services informatiques dans le *Cloud* (e.g. Amazon, Microsoft, etc) [25, 87]. Ces services sont consommés par des clients qui peuvent être soit des entreprises, ou des particuliers [127, 133]. Les services du Cloud sont facturés à l'usage (*pay-as-you-go*) et ils s'adaptent facilement aux fluctuations de la demande. La qualité de service est garantie par

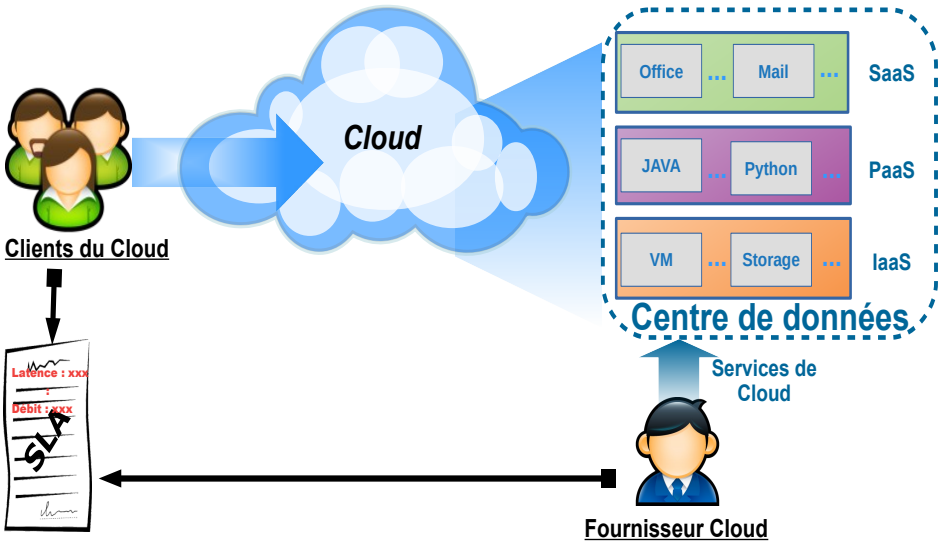

FIGURE 1.1 : Les acteurs de l'écosystème du Cloud Computing.

un contrat appelé *Service Level Agreement* (SLA) regroupant les métriques qui décrivent la qualité de service souhaitée par le client (temps de réponse, latence, etc.) [89]. La figure 1.1 illustre les acteurs du Cloud (fournisseur de Cloud et Client) et la relation entre ces derniers (e.g. service, SLA).

Les services offerts par le Cloud sont généralement classés en trois catégories principales [62] :

1. *Software as a Service (SaaS)* : le fournisseur de Cloud offre des applications prêtes à être utilisées (i.e. Microsoft office live, Dropbox, etc.)

2. *Platform as a Service (PaaS)* : le fournisseur de Cloud fournit aux programmeurs un environnement de développement (i.e. Google's App Engine).

3. *Infrastructure as a Service (IaaS)* : le fournisseur de Cloud offre des ressources matérielles de bas niveau (i.e. AWS de Amazon). Les types de ressources disponibles sont généralement le stockage, le traitement et le réseau.

Néanmoins, il existe d'autres types de service (*XaaS* pour *Everything as a service*) qui sortent de la classification classique citée ci-dessus tels que la *Communication as a Service (CaaS)*.

Dans ce travail, nous nous intéressons au service de base de données (DBaaS pour *DataBase as a Service*). Il consiste à proposer un système de gestion de base de données ainsi que les fonctions associées à sa gestion comme un service. Il permet aux clients d'éviter les coûts liés à l'investissement (e.g. achat de matériel, location des locaux, etc.), l'exploitation, la maintenance, l'administration, la sécurité et la disponibilité du système de base de données. Comme tout service de Cloud, la qualité de service est garantie par l'établissement d'un contrat spécifiant le SLA. Dans le cas d'une violation du SLA, une pénalité est appliquée au

fournisseur de service [127]. La pénalité est considérée comme une remise sur la facture globale du client. Le montant de celle-ci dépend de plusieurs paramètres comme le degré de violation et sa durée [127]. De ce fait, la satisfaction des SLAs des clients représente un défi pour les fournisseurs de Cloud [95].

L'un des objectifs principaux des fournisseurs de Cloud est de satisfaire les SLAs de leurs clients à moindre coût. Minimiser les coûts permet aux fournisseurs d'offrir des services attractifs à des prix compétitifs. Dans cette perspective, plusieurs technologies ont été utilisées pour optimiser les coûts tels que le coût de stockage (e.g. l'utilisation des systèmes de stockage hybride), le coût de calcul (e.g. la virtualisation) et le coût de maintenance (e.g. les applications multi-locataires). Dans ce travail, nous nous intéressons au système de stockage hybride qui est considéré comme l'une des solutions pour avoir un système de stockage performant à un coût réduit. Il faut noter que l'optimisation des performances de système de stockage contribue grandement à l'amélioration du service DBaaS [111].

Jusqu'à la fin du $20^{ème}$ siècle, les systèmes de stockage étaient basés sur des disques durs magnétiques (HDD pour *Hard Disk Drive*) [73]. Cependant, les performances de ceux-ci n'ont pas pu suivre les exigences des applications qui demandent un nombre d'opérations d'E/S très élevé [71] (e.g. les bases de données) d'un côté et la vitesse de traitement CPU d'un autre côté. Pour combler l'écart de performances et répondre aux nouvelles exigences des applications, les chercheurs ont proposé d'intégrer les périphériques de stockage à base de mémoire flash (SSD pour Solid-State Drive) dans les systèmes de stockage [31]. Actuellement, ces périphérique sont utilisés par la majorité des centres de données et des entreprises (e.g. les fournisseurs de Cloud [31]).

Contrairement aux disques HDDs qui sont à la fois énergivores, peu performants pour les accès aléatoires et peu résistants aux chocs, les SSDs fournissent un débit important [71], une faible latence en lecture et écriture (plusieurs ordres de grandeur inférieure au HDD [77]) et une faible consommation d'énergie due à l'absence d'éléments mécaniques [72, 80]. En revanche, ces périphériques souffrent de quelques limites. Ils sont en moyene dix fois plus chers que les HDDs en termes de Go/$ [95]. De plus, ils ont une durée de vie limitée comparée aux HDDs [97]. Le nombre de cycles d'écriture/effacement toléré par une cellule de mémoire flash est limité [31]. Ces caractéristiques rendent ces mémoires très sensibles aux applications nécessitant des écritures intensives. De ce fait, les chercheurs ont suggéré l'utilisation de systèmes de stockage hybride (HSS pour *Hybrid Storage System*) qui combinent deux classes de stockage à savoir HDD et SSD afin de tirer profit des avantages de chacun d'entre eux [27, 72, 80, 118]. L'idée

est *d'essayer d'atteindre les performances des SSDs au prix des HDDs*.

Actuellement, les systèmes de stockage hybride sont de plus en plus utilisés. Cependant, l'adoption d'une telle technologie sans l'utilisation d'une stratégie de gestion adaptée pour les HSS pourra seulement augmenter les coûts sans forcément aboutir à améliorer les performances [95]. Il est donc nécessaire de concevoir des stratégies qui gèrent le placement des données et décident où et quand doit-on placer une donnée dans une classe de stockage. Ces stratégies doivent être capables de gérer le mouvement des données entre les différentes classes de stockage en cas de fluctuation de la charge de travail [117]. Nous définissons une charge de travail d'un client par l'ensemble des requêtes d'E/S transmises à ses données.

Plusieurs stratégies de placement ont été proposées dans la littérature. Leur objectif est de trouver un placement qui maximise les performances. Ces stratégies s'appuient généralement, dans leurs décisions, sur le profil d'accès des données (type et nature d'opérations d'E/S) et les caractéristiques des périphériques [33]. Le profil d'accès est extrait durant une phase préparatoire nommée, *phase de monitoring*. Cette phase permet d'avoir une idée préalable sur la nature des opérations appliquées sur les données. Selon la nature des opérations et les caractéristiques des périphériques, une décision de placement est prise dans une deuxième phase nommée *phase de décision* [33].

Toutefois, ces travaux ne sont pas adaptés pour le Cloud car les objectifs de ces derniers et ceux que pourrait avoir un fournisseur de Cloud ne sont pas les mêmes. Les travaux existants s'intéressent à l'optimisation des performances [33]. Ils sont dédiés aux entreprises dont les services sont consommés en interne et non celles qui offrent des services destinés pour la location à l'image des fournisseurs Cloud. L'objectif du fournisseur Cloud est de satisfaire les contraintes de ses clients et d'optimiser les coûts liés à l'exploitation et à l'infrastructure [117]. Il faut noter que nous nous intéressons dans ce travail à l'optimisation des performances de requêtes d'E/S. Cette optimisation peut permettre d'améliorer le temps d'exécution des requêtes de base de données car le temps passé à traiter les opérations d'E/S liées à une requête de base de données représente en moyenne 90% du temps d'exécution de celle-ci [111].

1.2 CONTEXTE DES TRAVAUX DE THÈSE :

Cette thèse s'inscrit en partie dans le cadre du projet CMEP tassili *A Green and Highly Efficient and Evolutionary data and storage Management System for Cloud Data*

Base as a Service (GHEEMaS) initié en 2016 entre le laboratoire LSI de l'Université des Sciences et de la Technologie Houari Boumediene (USTHB), le Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance (lab-STICC) de l'Université de Bretagne Occidentale (UBO) et le Centre de Recherche sur l'Information Scientifique et Technique (CERIST). Ce projet à pour objectif principal d'étudier l'intégration des mémoires flash dans le Cloud afin d'augmenter les performances et d'optimiser le coût du stockage et la consommation énergétique à travers un placement intelligent des données.

1.3 PROBLÉMATIQUES

Dans cette thèse, nous nous intéressons au problème de placement des données dans un service de type DBaaS sur un système de stockage hybride HDD-SSD, tout en s'inscrivant dans la stratégie globale du Cloud qui est la satisfaction des SLAs et la minimisation des coûts. Dans les sections suivantes, nous donnons plus de détails concernant les motivations et les problématiques visées par ce travail. Toutefois, avant de détailler ces problématiques, nous rappelons rapidement l'architecture générale du système considéré dans notre étude.

1.3.1 *Architecture générale*

Comme le montre la figure 1.2, nous considérons un centre de données doté d'un système de stockage hybride. Ce système regroupe deux classes de stockage HDD et SSD. Les classes de stockage présentent des performances différentes et des coûts différents en Go/$. Le système de stockage est partagé entre plusieurs clients. Chaque client détient sa propre base de données représentée par un ensemble d'objets (e.g. table, index, fragmentent), sa propre charge de travail et son propre SLA.

Dans cette partie, nous décrivons les problématiques traitées dans ce travail de thèse.

1.3.2 *Évaluation du coût de placement des objets*

Rappelons que l'objectif principal du fournisseur de Cloud est de trouver un placement optimal des objets qui minimise au mieux les coûts pour les fournisseurs Cloud en garantissant le SLA. Cependant, pour trouver un placement optimal des objets parmi tous les placements possibles, il faut savoir d'abord les

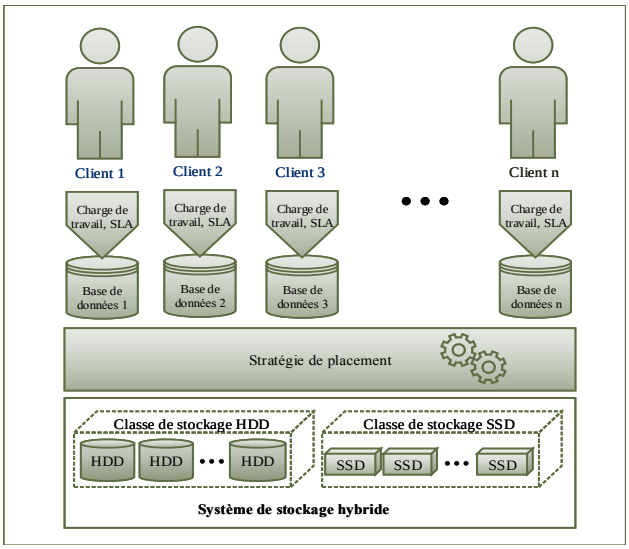

FIGURE 1.2 : Le placement des objets dans le Cloud.

évaluer. De ce fait, nous avons adressé une première problématique qui consiste à investiguer un modèle permettant d'évaluer le coût d'un placement d'objets. En effet, le placement des objets et l'exécution des requêtes d'E/S transmises à ces derniers engendrent différents coûts pouvant découler de plusieurs facteurs (énergies, pénalités, migration, etc). La problématique posée est : *quels sont les coûts induits par un placement et comment modéliser et valider l'ensemble de ces coûts sous forme d'une seule fonction ?*.

1.3.3 *Problème de placement des objets*

La deuxième problématique traitée dans cette thèse est celle du placement des objets dans un système de stockage hybride. Les fournisseurs de Cloud actuels utilisent un système de stockage hybride qui combine plusieurs classes de stockage, généralement HDD-SSD, afin de satisfaire les différentes exigences des applications. Ce système de stockage est partagé entre plusieurs clients (architecture multi-locataires) dont chacun dispose d'un ensemble d'objets et de son propre SLA. *La problématique traitée consiste à trouver un placement optimal des objets de plusieurs clients qui minimise au mieux le coût de placement tout en respectant les contraintes en termes de SLA de ces Clients.*

1.4 CONTRIBUTIONS

Pour chaque problématique soulevée dans la partie précédente, nous avons proposé une approche. Nos approches dans cette thèse sont résumées en deux points décrits ci-après. Pour chaque contribution, nous introduisons une brève description de l'idée, notre raisonnement, les techniques utilisées, la méthode de validation et quelques résultats obtenus.

Après avoir étudié les différentes solutions proposées dans la littérature pour résoudre le problème de placement des données dans un système de stockage hybride, nous avons constaté que la majorité des solutions ne sont pas adaptées pour le Cloud. Les stratégies proposées cherchent à optimiser les performances et ne prennent pas en considération les SLAs des clients. Pour le Cloud, la satisfaction des contraintes des SLAs de ses clients est primordiale [95]. La violation de celles-ci engendre des pénalités et donc, des pertes financières considérables [127]. En outre, elle nuit à la réputation du fournisseur de Cloud [127]. C'est pourquoi le fournisseur de Cloud considère très sérieusement la satisfaction des SLAs. Il n'existe, au moment de la rédaction du manuscrit et à notre connaissance, aucune solution adaptée pour le contexte du Cloud.

L'idée principale est de proposer un système autonome de gestion de placement des objets de DBaaS en s'inspirant du modèle MAPE-K (*Monitor, Analyze, Plan, Execute, Knowledge*) introduit par IBM au début des années 2000 [69]. Ce modèle définit la gestion autonomique d'un système informatique en se basant sur la mise en œuvre d'une boucle de contrôle. Le modèle MAPE-K comporte quatre étapes essentielles. L'étape "*Monitor*" consiste à surveiller le système considéré, en utilisant un ensemble de capteurs préalablement installés. La deuxième étape, "*Analyze*", récupère la sortie de l'étape précédente (*Monitor*) afin de les analyser et de déterminer l'état du système. L'étape "*Plan*" permet de déterminer une nouvelle configuration qui répond aux objectifs visés. La dernière étape "*Execute*" applique la nouvelle configuration à l'aide d'un ensemble d'actionneurs. La base de connaissance "*Knowledge*" détient toute la connaissance sur les modèles et l'historique des précédentes étapes du modèle MAPE-K, et donc chaque étape peut exploiter ces données.

Pour l'étape "*Monitor*", nous avons développé un outil pour tracer les requêtes d'E/S transmises aux objets. Ce dernier agit au-dessous du tampon (*buffer*) de Système de Gestion de Base de Données (SGBD) et du cache système. La caractérisation des charges d'E/S est basée sur l'analyse des traces récoltées. Cet outil est développé sous la forme d'un module noyau Linux.

1.4.1 *Proposition d'un modèle de coût*

Pour l'étape "*Analyze*", nous avons proposé un modèle de coût pour évaluer le placement des objets dans un Cloud. Nous avons considéré que le système de stockage combine deux classes ; HDD et SDD. Notre méthode pour la définition du coût de placement des objets est la suivante :

1. Énumérer et définir les coûts qui peuvent être engendrés par un placement. Ces coûts peuvent provenir de plusieurs sources (énergies, pénalités, migration, etc).
2. Pour chaque coût, nous définissons une méthode d'évaluation. Cette méthode est mise en œuvre par une fonction de calcul.
3. La dernière étape consiste à évaluer toutes les fonctions de calcul proposées.

Le modèle de coût proposé prend en considération le contexte du Cloud (e.g. pénalité, SLA, virtualisation, multi-locataires, etc). Il regroupe plusieurs coûts à savoir : 1) des coûts liés au système de stockage lui-même comme le coût énergétique et l'endurance, 2) des coûts liés à l'investissement comme le coût d'occupation, 3) des coûts liés au Cloud comme le coût de pénalité, 4) des coûts liés à l'exécution de la solution du placement comme le coût de migration et 5) des coûts liés à la gestion, comme les coûts associés à la ressource humaine et le coût de refroidissement. Le modèle de coût a été validé en utilisant deux benchmarks TPC-C et TPC-H. Il a été aussi comparé avec l'état de l'art. Les expériences ont montré que le modèle de coût proposé permet d'avoir une évaluation pertinente et précise en comparant avec les travaux connexes proposés dans la littérature.

Les résultats de ces travaux ont fait l'objet d'une publication dans la conférence DEXA 2016 (The International Conference on Database and Expert Systems Applications) [28].

1.4.2 *Stratégies de placement*

Après avoir traité l'étape "*Analyse*" du modèle MAPE-K, nous nous sommes appuyé sur celle-ci pour construire la phase d'optimisation "*Plan*". La phase "*Plan*" vise à proposer une stratégie de placement des objets de DBaaS dans un contexte de Cloud. Nous avons modélisé le problème de placement sous la forme d'un problème d'optimisation. L'objectif est de trouver un placement parmi d'autres ayant un coût réduit et qui préserve les contraintes des SLAs des clients. Nous avons utilisé le modèle de coût proposé dans la première contribution pour évaluer le coût de placement. Ce dernier représente la fonction objective du pro-

blème d'optimisation. Nous avons considéré trois contraintes pour le problème de placement de données :

1. Les SLAs des clients doivent être satisfaits.
2. Les contraintes de stockage et de performance doivent être vérifiées pour toutes les classes de stockage.
3. Tous les objets doivent être placés (stockés)

Deux stratégies ont été proposées dans ce sens : La première, appelée *Genetic Cost Based Object Placement Strategie* (G-COPS), est basée sur une méta-heuristique à savoir un algorithme génétique (GA), la deuxième approche est basée sur un algorithme heuristique incrémental ad-hoc appelée *Heuristic Cost Based Object Placement Strategie* (H-COPS).

Les stratégies proposées ont été validées sur une plateforme expérimentale réelle par les deux benchmarks TPC-C et TPC-H. Elles ont été comparées avec les travaux connexes [33, 132]. Les tests ont donné des résultats satisfaisants avec une réduction du coût de placement pouvant atteindre 40% par rapport à des algorithmes de état de l'art [29].

Les résultats de ces travaux ont fait l'objet d'une publication dans la conférence internationale *International Symposium on Cluster, Cloud and Grid Computing* (CCGrid 2017) [27] et d'un article dans la revue elsevier *Future Generation Computer Systems* (FGCS) [29].

1.5 PLAN DU MÉMOIRE

Ce manuscrit de thèse est composé de six chapitres organisés en deux parties en plus d'une introduction et une conclusion. Le chapitre 1 présente une introduction de cette thèse. La première partie regroupe le contexte de notre étude et un état de l'art concernant nos contributions. La deuxième partie décrit chacune de nos contributions. Le dernier chapitre conclut ce document.

Introduction

Le chapitre 1 présente un résumé du contexte de travail, les problématiques étudiées et les contributions proposées.

Première partie : contexte et état de l'art

Le chapitre 2 définit le contexte et les concepts généraux concernant les systèmes de stockage et le *Cloud*.

Le chapitre 3 présente un état de l'art sur le placement de données dans un système de stockage hybride. Nous avons étudié les travaux évaluant le coût de placement de données et ceux qui traitent de la problématique de placement de données.

Deuxième partie : contributions

Le chapitre 4 est consacré à la modélisation des coûts de placement des objets et le coût d'exécution des requêtes d'E/S transmises à ces derniers. Ce chapitre introduit les différents coûts constituant le modèle de coût global et présente la pertinence de chacun d'entre eux. Nous y décrivons ensuite la méthode et les résultats d'évaluation de ce modèle.

Le chapitre 5 décrit nos approches d'optimisation de placement des objets. Ces approches prennent en considération les aspects clés du Cloud (e.g. multi-locataires, SLA, pénalités). Nous commençons par introduire brièvement les principes d'optimisation de placement. Nous détaillons par la suite nos approches d'optimisation pour finir par une évaluation et une discussion des résultats obtenus.

Conclusion

Le chapitre 6 conclut le manuscrit en résumant les différentes contributions, et en proposant quelques perspectives.

Première partie

CONTEXTE ET ÉTAT DE L'ART

Chapitre 2

CONTEXTE

Avant de présenter nos contributions et nos approches, nous décrivons d'abord un certain nombre de concepts et de notions utilisées dans ce document. Cette partie constitue un support théorique à notre thèse et et aide le lecteur à mieux comprendre : 1) la problématique de placement des données dans un Cloud DBaaS sur un système de stockage hybride, 2) nos contributions et 3) les outils utilisés tout au long de notre étude.

Nous nous focalisons essentiellement sur le *Cloud Computing*, *DBaaS*, et *les systèmes de stockage*.

Sommaire

2.1 CLOUD COMPUTING

Le *Cloud Computing* ou « informatique dans les nuages », est un modèle informatique qui consiste à proposer les services informatiques à la demande. Ces services sont accessibles de n'importe où, n'importe quand et par n'importe qui [25].

Cette philosophie n'est pas tout à fait nouvelle. Elle remonte à une conférence publiée en 1961 par *John McCarthy* au MIT (Massachusetts Institute of Technology) [112]. Il a suggéré que la technologie informatique partagée (temps partagé) pouvait construire un bel avenir dans lequel la puissance de calcul et même les applications pouvaient être vendues comme un service.

Cette idée, très populaire dans les années 60, disparait au milieu des années 70 car les technologies matérielles, logicielles et réseaux n'étaient tout simplement pas prêtes et non matures. La technologie principale qui a rendu possible le Cloud Computing est la virtualisation [51].

La virtualisation permet d'optimiser l'usage les ressources matérielles en les partageant entre plusieurs environnements [64]. D'un point de vue matériel, la virtualisation est le fait d'exécuter plusieurs systèmes d'exploitation (SE) sur une même machine physique PM (PM pour Physical Machine) [95]. Cela permet de minimiser les coûts et d'améliorer : la performance, le passage à l'échelle, la fiabilité/disponibilité et la flexibilité [74].

2.1.1 *Définitions*

Définition 2.1. *Selon Peter Mell Timothy Grance du NIST (National Institute of Standards and Technology), le Cloud est un paradigme permettant un accès pratique, omniprésent, et à la demande à un ensemble de ressources de calcul partagées et configurables (e.g. réseaux, serveurs, stockage, applications, et services). Ces ressources peuvent être acquises et libérées avec un minimum d'effort de gestion ou d'interaction avec le fournisseur de service [88].*

Définition 2.2. *Selon le Syntec [9], le Cloud est une coopération de ressources informatiques, situées au sein d'une même entité ou dans diverses structures internes, externes ou mixtes, et dont le mode d'accès est basé sur les protocoles et les standards Internet.*

Après avoir définit le Cloud, nous introduisons l'un de ses concepts clés qui est la virtualisation

2.1.2 *Virtualisation*

La virtualisation est l'une des principales technologies qui ont contribué au lancement du *Cloud Computing*. En effet, cette technologie permet une gestion optimisée des ressources matérielles. Le but est de pouvoir exécuter plusieurs systèmes d'exploitation sur une seule machine physique et de fournir une couche supplémentaire d'abstraction entre le matériel et le système d'exploitation (SE pour *operating system*).

Les premiers travaux peuvent être attribués à IBM qui, dans les années 60, travaillait déjà sur les mécanismes de virtualisation en développant dans les centres de recherche de Cambridge un CMS (Conversation Monitor System), le tout premier hyperviseur [50]. C'est donc depuis presque 50 ans que l'idée d'une informatique à la demande est présente dans les esprits même si les technologies n'étaient pas en mesure de pouvoir la concrétiser [64].

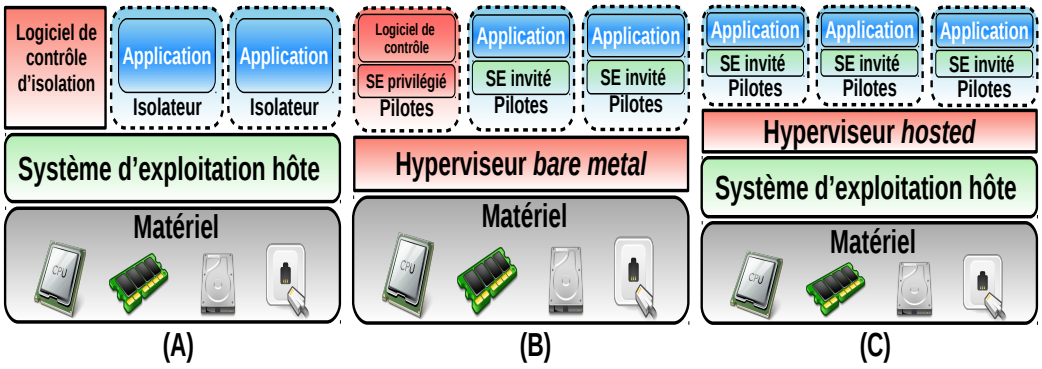

FIGURE 2.1 : Types de virtualisation des ressources matérielles [95]

Comme expliqué auparavant, la virtualisation des serveurs est un ensemble de techniques et d'outils permettant d'exécuter plusieurs systèmes d'exploitation sur un même serveur physique (voir figure 2.1). Le principe de la virtualisation est donc un principe de partage : les différents systèmes d'exploitation partagent entre eux les différentes ressources du serveur. Pour être utile de manière opérationnelle, la virtualisation doit respecter deux principes fondamentaux :

- l'***abstraction*** des ressources matérielles pour toutes les applications utilisant ces dernières.
- l'***isolation*** de l'exécution entre les différentes applications partageant les mêmes ressources.

L'outil de virtualisation des ressources matérielles permettant d'assurer l'abstraction et l'isolation de ces dernières est appelé ***hyperviseur*** ou plus générale-

ment ***gestionnaire de VM*** (VM pour *Virtual Machine*). Il existe trois types d'hyperviseur à savoir : 1) l'hyperviseur *bare metal*, 2) l'hyperviseur *hosted* et 3) l'isolateur.

2.1.2.1 *Hyperviseur bare metal*

Un hyperviseur *bare metal* (ou de type 1), est un logiciel qui s'exécute directement sur une plateforme matérielle (machine physique) comme illustré dans la figure 2.1 (B). Il est considéré comme un noyau hôte allégé et optimisé pour exécuter des noyaux de systèmes d'exploitation invités adaptés et optimisés à cette architecture spécifique. Cette plateforme est alors considérée comme un outil de gestion et d'allocation de ressources matérielles. Elle gère l'abstraction entre ces ressources et les machines virtuelles (les systèmes d'exploitation invités). Les machines virtuelles sont exécutées sur l'hyperviseur. Il faut noter que sur des processeurs ayant des instructions de virtualisation matérielle dédiées (AMD-V et Intel VT), le système d'exploitation invité n'a plus besoin d'être modifié pour pouvoir être exécuté dans un hyperviseur de type 1 [74].

Parmi les hyperviseurs les plus utilisés au moment de la rédaction du manuscrit, nous pouvons citer Xen [12], VMware ESX [10], ou Hyper-V [14].

2.1.2.2 *Hyperviseur hosted*

Un hyperviseur *hosted* (ou de type 2) est un logiciel qui s'exécute au sein d'un autre système d'exploitation. Un système d'exploitation invité s'exécute donc au troisième niveau au-dessus du matériel comme illustré dans la figure 2.1 (C). De la même façon, les systèmes d'exploitation invités n'ayant pas conscience d'être virtualisés, ils n'ont plus besoin d'être modifiés pour pouvoir être exécutés dans un hyperviseur de type 2.

Parmi les hyperviseurs les plus utilisés actuellement, nous citons VMware Workstation [21], VMware Fusion [11], l'hyperviseur open source QEMU [17], les produits Microsoft Virtual PC [21] et Virtual Server, VirtualBox d'Oracle [16], de même que Parallels Workstation de SWsoft et Parallels Desktop.

2.1.2.3 *Isolateur*

Appelé aussi virtualisation système (*OS-level virtualization*) [53], l'isolation est un principe utilisé principalement par les systèmes de type UNIX. L'isolation consiste à utiliser des fonctionnalités pour créer des zones d'exécution pour chaque application/utilisateur (voir figure 2.1 (A)) [95]. Parmi les outils d'isolation existants, nous pouvons citer *OpenVZ*, *Linux-Vserver,* et *LXC* (Linux) [95].

Les expérimentations menées dans le cadre de cette thèse utilisent un hyperviseur *bare metal* (ou de type 1). Toutefois, d'autres types de virtualisation peuvent être utilisés pour la validation de nos approches. La section suivante présente les services offerts par le Cloud.

2.1.3 *Les services du Cloud*

Les services offerts par le Cloud peuvent être divisés en trois catégories principales décrites ci-après :

2.1.3.1 *Software as a Service (SaaS)*

Ce mode désigne l'utilisation d'une solution logicielle à distance qui est hébergée par le fournisseur de Cloud. Le service est consommé, le plus souvent, à partir d'un simple navigateur Internet. Cela permet à une entreprise utilisant ce service de se débarrasser de toute contrainte d'installation, de mise à jour et de maintenance technique. La solution est facturée soit par abonnement, soit proportionnellement à l'usage. Parfois, des frais supplémentaires liés à la personnalisation et à la mise à disposition du service peuvent être imputés au client. Dans le domaine du webmarketing, les plateformes de gestion des campagnes email, les outils de web analytique et des serveurs publicitaires sont généralement proposés en mode SaaS [25].

2.1.3.2 *Plateforme as a Service (PaaS)*

Ce type de service regroupe principalement des serveurs mutualisés et leurs systèmes d'exploitation. En plus de pouvoir délivrer des logiciels en mode SaaS, le PaaS dispose d'environnements spécialisés au développement comprenant des langages, des outils et des modules nécessaires [112].

Ces environnements sont hébergés par un prestataire basé en dehors de l'entreprise ce qui permet d'éviter tous les coûts associes à l'infrastructure et au personnel de maintenance et donc de pouvoir se consacrer au développement de leur cœur de métier. Toutefois, ce service est limité à une ou à deux technologies (e.g. Python ou Java pour Google AppEngine, .NET pour Microsoft Azure). En outre, il n'y a pas de contrôle sur les machines virtuelles exécutant ces plateformes [109].

2.1.3.3 *Infrastructure as a Service (IaaS)*

Il s'agit de la mise à disposition, à la demande, de ressources d'infrastructures à distance localisées dans des centres de données. Le service IaaS permet l'accès aux serveurs et à leur configuration pour les administrateurs de l'entreprise. Le

client a la possibilité de louer des *clusters*, de la mémoire ou du stockage. Le coût est directement lié au taux d'occupation et à l'usage.

Ci-après nous présentons les principaux fournisseurs du Cloud.

2.1.4 *Les principaux fournisseurs de Cloud*

Différents acteurs du monde de l'IT comme Amazon, Google et Microsoft ont commencé à proposer des services informatiques utilisant le modèle du Cloud Computing. Ces services, sont hébergés dans des centres de données immenses et reposent principalement sur des technologies de virtualisation et d'automatisation de la location et de la gestion des ressources. Ci-après quelques exemples de fournisseurs :

2.1.4.1 *Amazon*

En 2002, Amazon le site marchand de renommée internationale, lançait "Amazon Web Services". La crise économique de 2002 a poussé le site-marchant à louer ses ressources non-utilisées [3]. C'est de là qu'est venue l'idée à la société de Seattle de mettre en place un ensemble de services web destiné à plusieurs types de clients. Le catalogue de services s'est enrichi avec le temps. Actuellement, pas moins d'une quinzaine d'offres de services sont proposées par Amazon. L'offre la plus connue est certainement Amazon Elastic Compute Cloud (EC2) qui permet un déploiement de machines virtuelles directement par le client, de manière automatisée et en fonction du besoin, d'où le terme élastique [2]. Les coûts sont calculés en fonction du temps d'utilisation (occupation). La localisation géographique et la personnalisation jouent aussi un rôle dans la variation du prix.

2.1.4.2 *Google*

Lancé en 2008, Google Apps Engine offre un ensemble d'outils qui permettent à ses clients de facilement s'adapter et développer des applications et ensuite les déployer rapidement sur le Cloud [6]. Google Apps Engine s'occupe de la configuration des machines virtuelles, de l'évolutivité des ressources de stockage et de calcul, ce qui permet aux développeurs de se concentrer seulement sur la saisie et l'optimisation du code source. Google Apps Engine offre également un ensemble d'APIs (*Application Programming Interface*), bibliothèques et composants réutilisables comme l'authentification et les services de mailing ce qui permet aux développeurs de réduire le temps de développement de leurs applications. Enfin, le déploiement d'application à grande échelle est l'une des principales tâches de Google Apps Engine, grâce à la réplication automatique de données

et l'équilibrage de charges. Cela permet aux applications web de garder un bon niveau de qualité de service [5].

2.1.4.3 *Microsoft*

Office 365 : constitue une transformation de la suite bureautique Microsoft en version en ligne [76]. L'objectif est de concurrencer la suite bureautique en ligne de Google et de reconquérir les clients perdus au profit de ce dernier. Sa stratégie est d'adapter les prix en fonction du client de telle façon que les petites structures paient un prix plus adapté.

Windows Azure : Windows Azure est une plateforme de développement basée sur le modèle Cloud. Elle permet aux développeurs de créer leurs applications sur la base d'un ensemble de standard tel que REST, SOAP ou HTTP. Windows Azure permet également aux développeurs de configurer dynamiquement les serveurs virtuels ainsi que l'espace de stockage selon les besoins de chaque application. Windows azure offre également un ensemble de services pour :

- La gestion de base de données dans le Cloud, avec SQL-based web service, ce qui permet, d'accéder à ces derniers à distance, par d'autres partenaires, ou par des utilisateurs mobiles.
- La réutilisation de composants, avec les services .Net, qui offrent un ensemble d'outils pour accélérer le développement et le déploiement d'application dans le Cloud.
- La fourniture d'un ensemble de outils et de services web qui représente un centre de documentation, d'API et d'échantillonnage pour la mise en route des applications basées Windows azure.

Jusque-là, nous avons introduit un aperçu général du Cloud. La prochaine section présente les bases de données dans le Cloud.

2.2 DBAAS : LES BASES DE DONNÉES DANS LE CLOUD

Avant de donner un aperçu des bases de données dans le Cloud nous définissons d'abord une base de données d'une manière générale.

2.2.1 *Définitions*

Définition 2.3. *Une base de données est vue comme une collection de données décrivant une activité. Cette collection est organisée et elle est représentée de manière à assurer un accès optimisé (rapide et facile) [65].*

Un service DBaaS est définit comme suit.

Définition 2.4. *La base de données en tant que service (ou DBaaS) est un modèle d'accès à la demande au service de bases de données [26]. L'infrastructure et la base de données sont en effet fournis à distance par le fournisseur de cloud, tandis que le client (le consommateur du service de base de données) est chargé de gérer le contenu et les opérations de la base de données. Le fournisseur de Cloud se charge également de la maintenance et il assure la disponibilité [63].*

2.2.2 *Types de base de données dans le Cloud*

Deux types de base de données sont proposés par les fournisseurs de Cloud : relationnelles et non relationnelles que nous présentons ci-après.

2.2.2.1 *Base de données relationnelle*

Dans ce modèle, les données sont représentées sous forme de tables (appelées relations) composées de lignes et de colonnes, et cela indépendamment de la façon dont les informations sont stockées dans la machine. Un ensemble de données est donc modélisé par un ensemble de tables [36]. Chaque colonne de la table représente un attribut, auquel est associé un domaine de valeurs [42]. Les lignes d'une table sont appelées enregistrements (*tuples* en anglais). Le modèle relationnel définit des relations entre les tables qui associent les enregistrements issus de deux tables ou plus. Les lignes d'une table peuvent être marquées avec une clé unique, qu'on appelle clé primaire. Une ligne peut être référencée dans une autre table en utilisant une clé étrangère [42]. La figure 2.2 montre un extrait d'une table voiture.

Les attributs

Les enregistrements

Matricule	Nom de propriétaire	Prénom de propriétaire	Année de circulation	Couleur
123466-189-02	Ali	Mohammed	1989	Blanche
529757-117-05	Boughouba	Rida	2017	Noir
111245-318-16	Hanafi	Imane	2018	Rouge
009879-111-21	Joh	Rachel	1011	Blanche

FIGURE 2.2 : Exemple d'une table Voiture

Les recherches et les mises à jour de la base de données sont effectuées à l'aide d'un langage non procédural standardisé appelé SQL (Structured Query

Language) [110].

Le succès du modèle relationnel auprès des chercheurs, concepteurs et utilisateurs est dû à la puissance et à la simplicité de ses concepts. En outre, contrairement à d'autres modèles, il repose sur des bases théoriques solides, notamment la théorie des ensembles et la logique mathématique (théorie des prédicats d'ordre 1) [36, 110].

Dans les dernières années, le volume des données traitées par les entreprises a considérablement augmenté. Ces données émanent de sources diverses (transactions, comportements, réseaux sociaux, géolocalisation...) et elles sont souvent structurées autour d'un seul point d'entrée, la clé primaire, et susceptibles de croître très rapidement. Elles possèdent autant de caractéristiques qui les rendent très difficiles à traiter et à gérer par les SGBD relationnels classiques. L'analyse de grands volumes de données, ce qu'on appelle *Big Data*, défie les moteurs de bases de données traditionnels [47]. C'est pour répondre à ces différentes problématiques que sont nées les bases de données NoSQL (Not Only SQL) [26, 63]

2.2.2.2 *Base de données non relationnelle (NoSQL)*

Les bases de données NoSQL désignent une catégorie de systèmes de gestion de base de données (SGBD) qui s'écarte du paradigme classique des bases relationnelles. Elles ont une conception particulièrement adaptée pour gérer un ensemble très volumineux de données [108]. Ces données proviennent de plusieurs sources et peuvent être distribuées.

Nous distinguons quatre types de moteurs NoSQL selon le schéma ou la structure des données qu'ils manipulent [108] :

- **Modèle clé-valeur :** Dans ce modèle, les informations sont stockées sous forme d'un couple clé/valeur où la valeur peut être une chaine de caractères, un entier ou un objet. Ce type de base de données offre de très bonnes performances [108]. Nous citons à titre d'exemple les bases Voldemort (Twitter) ou Riak [18].
- **Modèle orienté documents :** La représentation orientée documents est une extension du modèle clé-valeur. A l'instar de celui-ci, elle associe à chaque clé un document contenant des données organisées d'une manière hiérarchique à l'image d'un document XML ou JSON. Nous citons à titre d'exemple la base CouchDB [8].
- **Modèle orienté colonnes :** Inspiré de Google BigTable, plusieurs moteurs NoSQL mettent en œuvre une structure proche de la table relationnelle.

Les données sont ici clairement représentées en lignes et séparées par colonnes. Chaque ligne est identifiée par une clé, ce qu'on appelle dans le modèle relationnel une clé primaire, et les données de la ligne sont réparties dans des colonnes, ce qui représente un niveau de structuration plus fort que dans les deux modèles précédents [100]. Contrairement aux moteurs relationnels, les colonnes sont dynamiques dans les SGBD orientés colonnes. Au sein d'une même table deux lignes peuvent ne pas avoir le même nombre de colonnes car les valeurs nulles ne sont pas stockées ce qui est le cas dans les SGBD relationnels. Cette propriété permet de libérer de la place de stockage et d'améliorer les performances de traitement car la volumétrie de données à traiter est plus faible. Ce modèle est utilisé pour des volumétries importantes. Les SGBD Cassandra et Hbase appartiennent à cette catégorie de moteurs [124].

- **Modèle orienté graphes :** Les trois premières familles NoSQL n'adressent pas le problème de corrélation entre les éléments. Prenons l'exemple d'un réseau social : dans certains cas, il devient très complexe de calculer la distance entre deux personnes non directement connectées [121]. Les bases de données orientées graphes ont été proposées pour remédier à ce type de problèmes. Dans ce type de base de données, les données stockées sont : les nœuds, les liens et des propriétés sur les nœuds et les liens [108]. Les requêtes que l'on peut exprimer sont basées sur la gestion de chemins, de propagations, d'agrégations, voire de recommandations [100]. Nous citons à titre d'exemple la base Neo4J [15].

Il faut noter que nous prenons les bases de données relationnelles comme cas d'étude dans ce travail de thèse. Toutefois, nos approches sont applicables aux bases de données NoSQL. L'application des stratégies proposées dans le cadre de cette thèse sur les bases de données NoSQL fait l'objet d'une autre étude. La section suivante décrit les différentes architectures utilisées dans le Cloud pour offrir un service d'une base de données relationnelle.

2.2.3 *Architecture Multi-locataire*

Cette section décrit les applications basées sur une architecture multi-locataire d'une manière générale et détaille les approches de conception qui ont été proposées pour développer un système de gestion de base de données multi-locataire.

2.2.3.1 *Application Multi-locataire*

La multi-location est un principe d'architecture logicielle relativement nouveau, généralement utilisé lorsqu'une application est destinée à être fournie sous forme

d'un service, on parle alors de mode SaaS. Dans cette architecture, une seule instance d'application est utilisée pour servir plusieurs clients [93]. Chaque client est appelé locataire (*tenant* en Anglais). Cette architecture s'oppose à une architecture mono-locataire (*single-tenant* en anglais) où chaque organisation cliente détient sa propre instance d'installation logicielle et matérielle. Les locataires partagent donc la même application et les mêmes ressources physiques. L'application doit être personnalisable et configurable pour répondre aux différentes exigences métiers des locataires.

Dans un environnement multi-locataire, une seule machine physique est utilisée pour servir plusieurs clients. Les ressources de la machine (CPU, RAM, I/O) sont partagées entre les différents locataires de la machine physique [91]. Ce nouveau principe réduit considérablement les coûts de déploiement et de maintenance, comme il améliore la scalabilité (le passage à l'échelle) de l'application car tous les clients (locataires) partagent la même instance de cette application.

Cependant, la multi-location rend le développement des applications SaaS plus complexe [128]. En effet, les développeurs de l'application doivent prendre en charge la gestion et la mise en œuvre de la variabilité. Le concept de variabilité permet au logiciel de répondre aux besoins spécifiques d'un locataire sans affecter les autres locataires hébergés [105]. Autrement dit, l'application doit être paramétrable afin qu'elle puisse s'adapter aux différentes demandes des locataires.

Pour qu'une application soit conforme à une architecture multi-locataire, elle doit garantir les trois caractéristiques suivantes [92] :

- *Isolation des données :* Les données d'une application ne doivent être visibles que pour celle-ci. Aucune application ne peut lire ou écrire les données d'une autre application.
- *Personnalisation des fonctionnalités :* L'application doit être personnalisable et configurable pour répondre aux exigences métiers variables de chaque locataire.
- *Isolation des environnements d'exécution :* L'application doit garantir une étanchéité entre les environnements d'exécution. Chaque utilisateur a son propre environnement d'exécution (i.e. espace RAM, registres CPU, etc.)

2.2.3.2 *DBaaS Multi-locataire*

Plusieurs approches de conception ont été proposées pour développer un système de gestion de base de données (SGBD) multi-locataire. Nous distinguons 4 principes de partage selon le niveau de partage et l'isolation [103].

Machine physique partagée, machine virtuelle séparée : Dans ce modèle la machine physique est partagée par plusieurs locataires. Chaque locataire a sa propre machine virtuelle dans laquelle la base de données est installée [38, 46]. Cette architecture offre une isolation très élevée (i.e. plus de sécurité) mais consomme plus de ressources. Exécuter plusieurs machines virtuelles, plusieurs systèmes d'exploitation et plusieurs SGBD au sein de la même machine physique augmente la consommation de ressources. Cette représentation augmente aussi les coûts liés à la gestion et à l'exploitation. Ce modèle est utilisé actuellement par plusieurs fournisseurs de Cloud (i.e. Amazon).

Machine physique partagée, Base de données séparée : Afin de réduire la consommation de ressources et d'optimiser le coût, les chercheurs ont proposé d'éliminer la virtualisation d'OS (i.e. une consolidation au niveau du système d'exploitation), et ceci par l'exécution de plusieurs SGBD au sein d'une seule machine physique. Dans ce modèle, les locataires partagent la même machine physique, cependant chaque locataire a son propre SGBD. Le stockage des données des locataires dans des bases de données distinctes constitue l'approche la plus simple pour isoler les données [46]. Cette alternative facilite l'extension du modèle de données de l'application pour des besoins spécifiques à un client et offre un bon niveau de sécurité qui empêche tout locataire d'accéder accidentellement ou malicieusement aux données des autres locataires. Cependant, le coût de maintenance reste relativement plus élevé par rapport aux autres modèles présentés ci-dessous.

Base de données partagée, schéma séparé : En allant toujours vers une vision de minimisation des coûts et de la consommation de ressources, une deuxième consolidation au niveau du SGBD est proposée. Le principe est d'héberger plusieurs locataires dans la même base de données, chaque locataire ayant son propre ensemble de tables regroupées dans un schéma créé spécifiquement pour le locataire [81]. Lorsqu'un client s'abonne au service, le sous-système de provisionnement crée un schéma dédié qui regroupe l'ensemble des tables de locataire. Il est facile de mettre en œuvre ce modèle mais il offre une isolation moyenne (sécurité moyenne) par rapport aux modèles précédents.

Schéma partagé : Cette approche est la plus optimisée en termes de consommation de ressources et de coût. Elle propose le niveau de consolidation le plus élevé (au niveau de schéma relationnel). L'approche consiste à utiliser la même base de données et le même schéma pour servir plusieurs locataires. Le principe est qu'une table dans la base peut contenir des enregistrements de plusieurs locataires stockés dans n'importe quel ordre [131]. Chaque table possède une colonne (nommée généralement *tenant*) qui spécifie l'identitée du locataire ayant

cet enregistrement. Ce modèle est très facile à mettre en œuvre mais le niveau de sécurité est faible car il n'existe qu'une séparation logique des objets de la base de données.

Il faut noter que nos stratégies supportent les quatre principes de partage. Cependant, les expériences sont menées en utilisant le mode *machine physique partagée, base de données séparée* (i.e. une consolidation au niveau du système d'exploitation).

La suite de ce chapitre est consacrée à la présentation des benchmarks couramment utilisés pour évaluer les performances d'un système de gestion de base de données.

2.2.4 *Benchmarks pour les bases de données*

Plusieurs benchmarks ont été proposés pour simuler une charge de travail réelle d'une application de base de données, nous citons à titre d'exemple TPC-H, TPC-C, TPC-D, TPC-E, ASP3AP et SSB. L'organisation TPC (*Transaction Processing Performance Council*) est l'un des pionniers dans ce domaine. Cette organisation à but non lucratif fondée en 1988 joue un rôle prépondérant en matière de benchmarking des bases de données [20]. Sa mission principale est de : 1) définir des benchmarks standards, 2) vérifier leur application correcte par les usagers qui souhaitent voir leurs produits évalués et 3) publier régulièrement les résultats de ces tests de performance [20]. Au cours de son histoire, TPC a spécifié huit benchmarks, dont quatre seulement sont actuellement en usage. Ces benchmarks partagent toutes les variantes d'une base de données commerciale classique du type client-commande-produit-fournisseur-stock. Le paramètre **facteur d'échelle** détermine la taille de la base de données (e.g.de 1 à 100000 Go). Les principaux benchmarks sont :

- TPC-C : benchmark utilisé pour simuler les transactions d'une base de données transactionnelle.
- TPC-H : benchmark utilisé pour simuler les transactions d'une application décisionnelle.
- TPC-W : benchmark utilisé pour simuler les activités commerciales d'un site de commerce électronique.
- TPC-App : benchmark utilisé pour simuler les transactions et les activités commerciales d'un serveur d'application et d'un web service.

Afin de reproduire le même environnement expérimental des travaux connexes avec les quels nos approches sont comparées, nous utilisons, dans ce travail, les benchmarks TPC-C et TPC-H.

2.2.4.1 *TPC-C*

En usage depuis 1992, TPC-C est spécifiquement dédié aux applications transactionnelles [20]. Plus complexe que son prédécesseur TPC-A, il propose une base de données élaborée qui simule parfaitement une base de données réelle. Cette base de données décrit l'activité d'une entreprise de grande distribution. La base de données comporte neuf tables de taille et de structure variées, ainsi qu'une charge de transactions d'une complexité différente (allant d'une simple requête de sélection jusqu'aux jointures complexes). Les transactions sont exécutées d'une manière concurrente. Cette charge de travail est constituée de cinq types de requêtes [42] et elle s'exerce sur l'ensemble des dimensions du système (i.e. une charge de travail qui sollicitent l'ensemble de données stockées dans la base de données). TPC-C respecte les propriétés ACID (Atomicité, Consistance, Isolation et Durabilité) [37]. Les métriques de performance adoptées par TPC-C sont les suivantes :

- Tpm-C : représente le nombre de transactions par minute. C'est la métrique qui mesure la performance du système.
- $/tpm-C : représente le coût de possession sur 3 ans rapporté à la transaction par minute. C'est une métrique pour mesurer le ratio Prix/Performance.

2.2.4.2 *TPC-H*

TPC-H est utilisé principalement pour simuler des applications décisionnelles [20]. Ce benchmark est conçu à partir d'une base de données comportant huit tables. Cette base de données décrit l'activité d'une entreprise de grande distribution [19]. La charge de travail est constituée de vingt-deux (22) requêtes décisionnelles paramétrées écrites en SQL, numérotées de Q1 à Q22 et de deux fonctions de rafraichissement RF1 et RF2 qui ajoutent et suppriment des n-uplets dans les tables ORDER et LINEITEM [19]. Les paramètres des requêtes sont instanciés aléatoirement en suivant une loi uniforme. Le protocole d'exécution de TPC-H est le suivant :

- Chargement : Cette étape consiste à générer et à charger les données au niveau de la base de données.
- Test de performance : C'est l'étape de mesure des performances. Elle est subdivisée en un test de puissance et un test de débit.

Jusque-là, nous avons donné un aperçu général des concepts de Cloud et des bases de données dans le Cloud. La prochaine section donne un aperçu sur les systèmes de stockages et l'intégration des mémoires flash dans les systèmes de stockage.

2.3 SYSTÈME DE STOCKAGE

Dans cette section on présente les concepts basiques des systèmes de stockage hybride à base de mémoires flash et de disques magnétiques présents dans les systèmes informatiques actuels. Ce chapitre introduit des notions nécessaires à la compréhension du travail de thèse. De plus, il constitue un état de l'art sur les disques magnétiques et les mémoires flash de type NAND considérés dans le cadre de ce travail de thèse.

2.3.1 *Définitions*

Définition 2.5. *Un système de stockage est l'ensemble des équipements informatiques et de logiciels appropriés, responsables du stockage à long terme de données et de leur accès [22, 54, 82]. Aujourd'hui, le stockage de données est assuré principalement par deux types de support de stockage à savoir :*

- *Les supports de stockage à base de mémoire magnétique tel que les disques magnétiques et les bandes magnétiques.*
- *Les supports de stockage à base de mémoire flash.*

La performance du système de stockage est généralement mesurée par une métrique de débit nommée IOPS.

Définition 2.6. *IOPS : c'est l'acronyme de Input/Output Operations Per Second. Cette métrique permet d'évaluer la performance d'un système de stockage exécutant des accès aléatoires [95]. Par convention, la taille d'une opération d'E/S est de 4Ko [94].*

Définition 2.7. *débit : le débit d'un système de stockage est le nombre d'octets transférés par unité de temps mesuré en mégaoctet par seconde (Mo/s) [75].*

Définition 2.8. *latence : la latence est le temps écoulé entre l'envoi d'une requête et la réception d'une réponse ou d'un bloc de données. Elle est considérée également comme l'une des métriques utilisées pour mesurer les performances d'un système de stockage [94].*

2.3.2 *Système de stockage à base de disque magnétique*

Les ingénieurs d'IBM n'étaient pas satisfaits des systèmes de stockage sur tambours magnétiques et bandes magnétiques [95].L'efficacité volumétrique était très faible, les tambours occupaient beaucoup d'espace pour peu de capacité (5 Mo) tandis que l'inconvénient majeur des bandes magnétiques était l'accès séquentiel à la donnée [7]. En 1953, un ingénieur eut l'idée de superposer des plateaux le long d'un axe et d'y adjoindre une tête de lecture/écriture mobile,

située sur un axe parallèle à celui des plateaux [85]. Cette tête venait s'insérer entre les plateaux pour lire les informations, mais elle devait se retirer complètement pour passer d'un plateau à un autre. Le premier disque dur concrétisant l'idée a vu le jour en 1956 dans les laboratoires d'IBM. Il s'agit du Ramac 305 (*Random Access Method of Accounting and Control*), un ensemble de 50 disques en aluminium de 61 centimètres de diamètre, tournant à 1200 rotations par minute et recouverts d'une fine couche magnétique. Ce disque avait une capacité de stockage totale de 5 millions de caractères [13]. Il proposait un taux de transfert de 8.8 Ko par seconde et pesait plus d'une tonne. Il révolutionna l'industrie informatique.

La figure 2.3 montre une découpe horizontale d'un disque dur magnétique. Ce dernier est constitué de plusieurs disques rigides (en anglais *hard disk*) en métal, en verre ou en céramique. Ces disques rigides sont empilés à une très faible distance les uns des autres comme le montre la figure 2.4. Ils sont appelés plateaux (en anglais *platters*). Chaque plateau est doté d'une tête de lecture/écriture. Les bits sont regroupés sur une face d'un plateau en cercles concentriques nommés pistes (voir figure 2.5). L'ensemble des pistes se trouvent à la même position sur les différents plateaux constitue un cylindre. Les pistes sont divisées en secteurs dans lesquels les bits sont stockés (voir figure 2.5). Les secteurs ont généralement une taille de 512 octets. La lecture/écriture d'une donnée sur le disque dur magnétique se fait grâce à la combinaison du mouvement rotatif des plateaux avec le mouvement linéaire sur un axe horizontal des têtes de lecture/écriture.

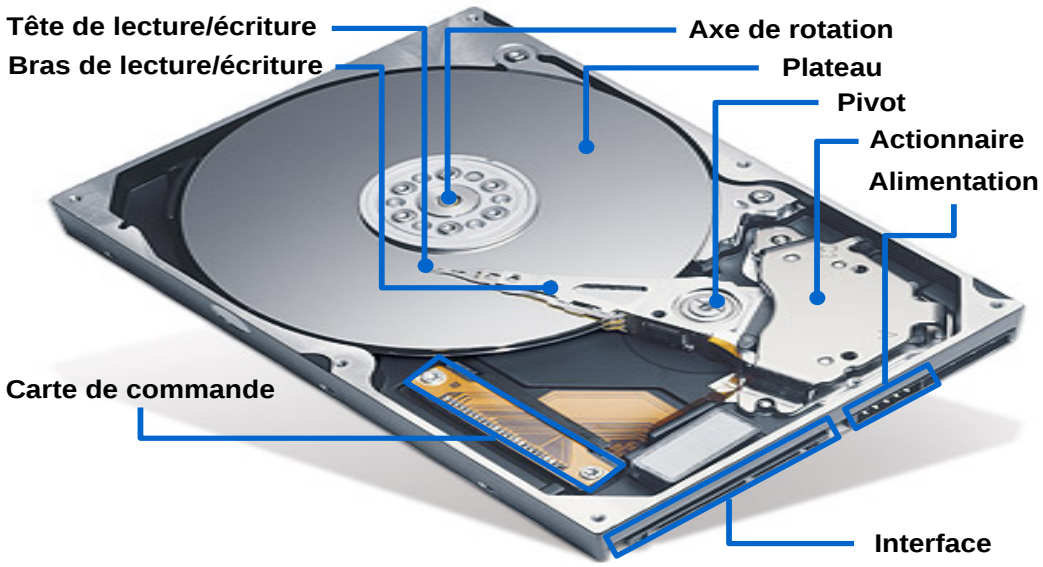

FIGURE 2.3 : Découpe horizontale d'un HDD [95].

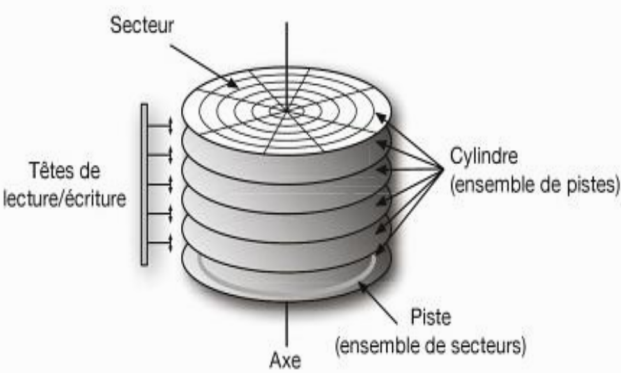

FIGURE 2.4 : Plateaux [95].

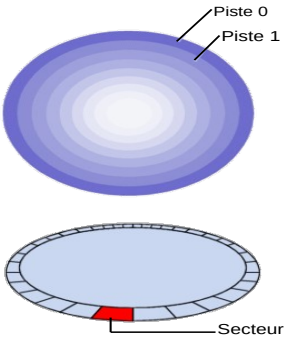

FIGURE 2.5 : Pistes.

Les disques tournent très rapidement autour d'un axe (à plusieurs milliers de tours par minute actuellement). La lecture et l'écriture se font grâce à des têtes de lecture situées de part et d'autre de chacun des plateaux. Ces têtes se baissent et se soulèvent pour pouvoir lire l'information ou l'écrire [129]. Les têtes de lecture/écriture sont dites « inductives », c'est-à-dire qu'elles sont capables de générer un champ magnétique [129].

Les systèmes d'exploitation regroupent les secteurs dans leur système de fichiers en unités logiques de stockage appelées bloc. Le bloc est la granularité sur laquelle s'effectuent les opérations de lectures et d'écritures. L'accès est fondé sur une adresse donnée à chaque bloc au moment de l'initialisation du disque par le système d'exploitation.

L'accès à un bloc, étant donné son adresse, se fait en trois étapes :

- Positionnement de la tête de lecture sur la piste contenant le bloc.
- Rotation du disque pour attendre que le bloc passe sous la tête de lecture.
- Transfert de données du bloc.

La durée d'une opération de lecture/écriture est égale à la somme des temps consacrés à chacune des trois opérations listées précédemment. Le temps de réponse des accès aléatoires est plus important que celui des accès séquentiels car, plus la rotation des disques, un déplacement des têtes de lecture est nécessaire [30] pour les accès aléatoires. Un accès est dit séquentiel si les blocs sollicités sont arrangés dans des secteurs adjacents. Les secteurs sont considérés adjacents s'ils appartiennent à la même piste. Par conséquent, il est de très loin préférable de lire/écrire sur un disque en séquentiel qu'en aléatoire. Les disques durs magnétiques présentent une asymétrie de performance entre les accès aléatoires et les accès séquentiels. Il faut noter que le mécanisme d'écriture est semblable à celui de la lecture, mais il peut prendre dans certains cas un peu plus de temps

si le contrôleur vérifie que l'écriture s'est faite correctement [71].

Nous distinguons trois modes d'opération d'un disque HDD à savoir :

- Mode actif (*active mode*) : il décrit l'état du disque durant l'exécution des opérations d'E/S. Ce mode consomme plus d'énergie que les modes présentés ci-dessous à cause du mouvement rotatif des plateaux ainsi que le déplacement de la tête de lecture/écriture
- Mode repos (*idle mode*) : quand le disque ne reçoit pas de requêtes d'E/S pendant une période définie (appelée timeout), il passe du mode actif au mode repos en diminuant la vitesse de rotation des plateaux. Le disque consomme moins d'énergie dans ce mode par rapport au mode actif car il n'y a que les plateaux qui tournent, les têtes restent fixes.
- Mode sommeil (*sleep mode*) : Ce mode consomme le moins d'énergie car il n'implique aucune activité mécanique.

Estimer la durée de vie d'un disque dur magnétique est une tâche ambigüe qu'hasardeuse [97]. Selon les constructeurs, la durée de vie d'un disque dur magnétique dépend de plusieurs paramètres : le MTBF (*Mean Time Between Failures*), le MTTF (*Mean Time To Failure*), l'AFR (*Annualized Failure Rate*), le nombre de cycles start-stop, la température, etc. [85, 95]. Cependant, la majorité des entreprises et les chercheurs affirment que la durée de vie d'un HDD dépend principalement du temps de service [55, 61, 113].

2.3.3 *Système de stockage à base de mémoire flash*

La mémoire flash est une mémoire de masse à semi-conducteurs réinscriptible, c'est-à-dire une mémoire effaçable et qui peut être reprogrammée de multiples fois. Elle est non-volatile, ce qui lui permet de maintenir de l'information sans source d'alimentation externe (pour une durée de temps définie). La mémoire flash est un sous-type de mémoires EEPROM (*Electrically Erasable Programmable Read-Only Memory*) qui sont des mémoires effaçables via courant électrique.

La mémoire flash utilise comme cellule de base un transistor MOS possédant une grille flottante enfouie au milieu de l'oxyde de grille, entre le canal et la grille (voir figure 2.6). L'information est stockée grâce au piégeage d'électrons dans cette grille flottante [22]. Il existe principalement deux types de mémoire flash NOR, NAND, selon le type de porte logique utilisée pour chaque cellule de stockage [41].

L'architecture des cellules de mémoire flash de type NOR permet d'adresser les données avec la granularité d'un octet [75]. Ce type de mémoire offre des performances élevées en lecture, soit une latence de 20 ns pour la lecture d'une page

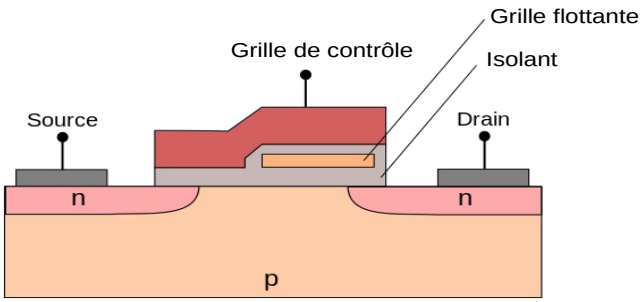

FIGURE 2.6 : Schéma d'une cellule de base de mémoire flash, composée d'un transistor à grille flottante

[104]. Les opérations d'écritures sont moins performantes que celles de lectures, soit une latence de 100 ns pour l'écriture d'une page [104]. Les mémoires flash de type NOR sont utilisées dans les mémoires centrales de quelques systèmes embarqués, PC de bureau et serveurs. Elles ont une faible densité et elles ne sont pas utilisées pour le stockage de masse [41].

Les mémoires flash de type NAND adressent les données avec la granularité d'une page (un groupe de cellules mémoires ayant une taille fixe). Ce type de mémoire flash présente des latences en lecture plus élevées que celles des mémoires flash NOR, soit 25 us pour la lecture d'une page. Ceci est principalement dû au fait que la densité des cellules NAND dans une mémoire flash est plus importante que celle des mémoires NOR. Ces mémoires sont principalement utilisées pour le stockage de données. Elles sont très répandues actuellement et sont utilisées comme mémoire secondaire dans des systèmes embarqués, les ordinateurs de bureau (PC) et les serveurs [75]. Elles équipent, par exemple, les périphériques de stockage SSD (Solid State Drive), les clés de stockage USB et les cartes de stockage SD. Selon MarketResearch, le marché de mémoire flash a une croissance annuelle moyenne de 69% jusqu'à 2015 grâce à l'apparition des nouvelles technologies de NVM comme FeRAM, PCRAM, MRAM et l'utilisation extensive des mémoires flash dans les systèmes embarqués.

Dans cette thèse, nous nous intéressons aux mémoires flash de type NAND utilisées dans le stockage de masse. Ces dernières peuvent être classées en trois catégories à savoir : 1) SLC (*Single Level Cell*), 2) MLC (*Multi Level Cell*) et 3) TLC (*Triple Level Cell*). Une mémoire flash SLC permet de stocker un seul bit par cellule, une mémoire flash MLC peut stocker deux bits par cellule et un mémoire flash TLC peut aller jusqu'à trois bits par cellule [99].

En terme de densité et de coût, les mémoires flash TLC sont volumineuses et moins chères en les comparant avec les mémoires flash MLC. Les mémoires flash MLC sont plus volumineuses et moins couteuses par rapport au flash SLC. En termes de performance, les mémoires flash SLC sont plus performantes que les mémoires flash MLC, tandis que les mémoires flash MLC sont plus performantes que les mémoires flash TLC. Les mémoires flash SLC sont utilisées dans les applications dont la fréquence de mise à jour est forte. En revanche, les mémoires flash MLC et TLC sont destinées à stocker un volume important de données.

La figure 2.7 montre une architecture simplifiée d'une mémoire flash NAND, ainsi que les opérations principales d'E/S. Une mémoire flash est composée d'une ou de plusieurs puces (chips), chaque puce est composée de plusieurs zones (plans), chaque zone est composée d'un nombre fixe de blocs [95]. Chaque bloc contient un nombre fixe de pages qui est un multiple de 32, typiquement 64. Chaque page est composée de deux parties, une partie pour stocker les métadonnées et une partie pour stocker les données de l'utilisateur.

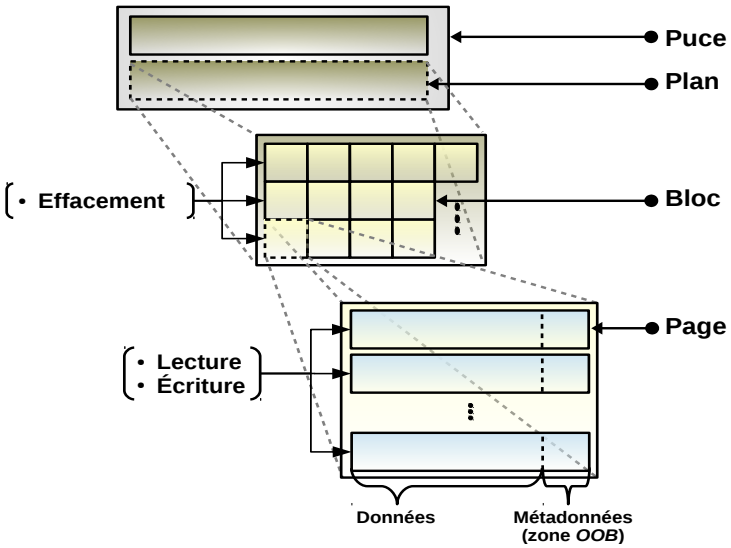

FIGURE 2.7 : Architecture simplifiée d'une puce flash NAND [94]

2.3.3.1 *Caractéristiques des mémoires flash*

Nous présentons ci-dessous les caractéristiques principales des mémoires flash.

Lecture aléatoire : les mémoires flash sont plus performantes que les disques durs magnétiques pour les opérations de lecture aléatoire. Cela revient aux caractéristiques électriques des mémoires flash et à l'absence d'un mouvement mécanique.

Asymétrie entre l'opération de lecture et d'écriture : les mémoires flash présentent une asymétrie de performance par rapport au type d'opérations (lecture/écriture). Les mémoires flash sont plus performantes en lecture qu'en écriture, moins performantes en écriture aléatoire. En effet, l'exécution d'une opération de lecture nécessite l'obtention du statut de la cellule. Cependant, les opérations d'écriture s'effectuent page par page. En outre, une opération d'écriture est précédée par une opération d'effacement d'un bloc si la page contenait des données.

Ecriture séquentielle : l'écriture s'effectue d'une manière séquentielle au sein d'un bloc flash et à une granularité de page. La taille des pages varie en fonction du type de la puce NAND (SLC, MLC, TLC). Les pages ayant des cellules SLC présentent des tailles variables (à partir de 2048 octets) [75]. Pour les pages ayant des cellules MLC, elles peuvent présenter des tailles de 2048, 4096 ou 8192 octets [75].

Faible consommation d'énergie : la consommation d'énergie d'une mémoire flash dépend de l'architecture interne des cellules et de la charge de travail transmise à celle-ci. Cependant, les mémoires flash consomment moins d'énergie par rapport aux disques durs magnétiques HDD.
A l'image d'un disque HDD, un disque à base de mémoire flash a trois modes à savoir : 1) actif (*active*), 2) repos (*idle*) et 3) sommeil (*sleep*). Ces modes correspondent exactement à ceux d'un disque HDD, sauf qu'il n'y a pas un mécanisme mécanique.

Interface E/S : l'une des propriétés principales qui permet l'utilisation des disques à base de mémoire flash SSD comme support alternatif des disques durs magnétiques HDD est l'utilisation d'une interface (connecteur) identique à celui du disque HDD. Aucune modification matérielle n'est requise pour basculer vers un stockage à base de mémoire flash.

Résistance aux chocs : Contrairement aux disques durs magnétiques, les disques à base de mémoire flash résistent mieux aux chocs grâce à l'absence des composants mécaniques.

2.3.3.2 *Contraintes et limitations relatives aux mémoires flash*

Si la mémoire flash possède de nombreuses caractéristiques qui la rendent très intéressante, un certain nombre de règles sont à respecter lors de l'utilisation de ce type de mémoire au sein d'un système de stockage. Ces limitations sont imposées, d'une part, par les principes physiques de fonctionnement des cellules mémoires de base, et d'autre part, par la micro-architecture spécifique des puces de mémoire flash. On peut classer ces contraintes en trois grandes catégories à

savoir : A) l'impossibilité de mettre à jour des données sur place, B) l'asymétrie entre la granularité de l'opération d'écriture et celle de l'effacement et, C) l'usure qui touche la mémoire au fur et à mesure à cause des opérations d'écriture/effacement. Ces contraintes sont décrites ci-après.

Effacement avant l'écriture(C1) : cette contrainte est l'une des plus importantes des mémoires flash. Elle est liée au fait qu'une écriture sur une page contenant des données doit être précédée d'un effacement [41, 41, 99]. Les mises à jour de données en place sont donc impossibles. Avant toute écriture dans une page contenant des données, il est au préalable nécessaire d'effacer cette page. Les limitations imposées par cette contrainte sont renforcées d'une part par l'asymétrie entre la granularité de l'opération d'écriture et celle de l'effacement (voir **C2**), et d'autre part par la différence des performances (latences) relative à chacune de ces opérations.

Écriture au niveau d'une page, effacement au niveau d'un bloc (C2) : la deuxième contrainte des mémoires flash est l'asymétrie entre la granularité de l'opération d'écriture et d'effacement. En effet, l'opération d'écriture s'effectue à une granularité d'une page et l'opération d'effacement s'effectue à une granularité d'un bloc complet [41]. Il est impossible d'effacer une page particulière dans un bloc.

Cycle de vie limité (C3) : le nombre de cycles d'écriture/effacement toléré par une cellule est limité et dépend du type de cellule. Cette limite est d'environ 100 000 cycles pour la NAND SLC, 10 000 pour la MLC, et 5 000 pour la TLC [119].

Une cellule de mémoire flash ne peut supporter qu'un nombre limité d'effacements. Au-delà du seuil toléré, la cellule devient inexploitable et elle est considérée comme défectueuse. Cette contrainte rend ces mémoires sensibles aux applications nécessitant des écritures intensives. L'usure de la mémoire flash est une contrainte à prendre en compte lors de la conception d'un système de stockage.

2.3.4 *Intégration des mémoires flash dans les serveurs*

Les périphériques de stockage ont évolué d'une façon drastique au cours des dernières années en termes de performance et de densité de stockage. Néanmoins, les disques durs magnétiques ne suivent malheureusement pas les progrès des microprocesseurs (CPU) et des mémoires principales (RAM) en termes de performance. Un écart de performance considérable est constaté entre les composants

de traitement (CPU et RAM) et les disques durs magnétiques [22, 41, 71].

Cette situation a poussé les chercheurs à étudier la possibilité d'intégrer les mémoires flash dans les systèmes de stockage. L'intégration des mémoires flash comme un support de stockage alternatif (ou complémentaire) des disques magnétiques permet de réduire l'écart de performance entre les composants de calcul (CPU et RAM) et le système de stockage. Elle permet aussi d'augmenter les performances de celui-ci et de minimiser la consommation d'énergie.

Toutefois, le coût élevé des mémoires flash, qui est dix fois plus important que celui des disques durs magnétiques et la durée de vie limitée de celles-ci, restent un sérieux handicap pour fonder un système de stockage basé entièrement sur des mémoires flash. Quatre modèles d'intégration (architectures) sont alors possibles [30].

- **Extension de la RAM (voir figure 2.8 (A))** : dans ce modèle, la mémoire flash est utilisée au même niveau que la RAM. Elle est considérée comme un buffer de la RAM [30]. Cela permet de réduire la consommation d'énergie, de minimiser les coûts et de restaurer rapidement les données.
- **Accélérateur d'un système de stockage (voir figure 2.8 (B))** : une deuxième architecture est proposée et elle consiste à utiliser la mémoire flash comme un cache du disque magnétique [94]. Dans cette architecture, la mémoire flash est placée entre la RAM et le système de stockage. Cela permet de réduire l'écart de performance existant entre la RAM et les disques durs magnétiques. Cette architecture multi-niveau augmente les performances et économise les coûts.
- **Support de stockage complémentaire pour les disques durs magnétiques (voir figure 2.8 (C))** : afin d'avoir les performances des mémoires flash au prix des disques durs magnétiques, les chercheurs ont recommandé l'utilisation d'un système de stockage hybride [72]. Ce dernier consiste à monter les disques durs magnétiques et les disques flash au sein du même système de stockage. Dans cette architecture les mémoires flash sont au même niveau que les disques magnétiques (un stockage complémentaire).

- **Support de stockage alternatif (voir figure 2.8 (D))** : ce modèle propose de remplacer les HDD par des disques à base de mémoire flash afin de diminuer la latence des opérations E/S, d'augmenter les performances et de réduire la consommation d'énergie [30]. Cependant, le coût important d'acquisition d'un 1 Go de stockage en mémoire flash reste dix fois important qu'un 1 Go de stockage en mémoire magnétique. Cette dernière

est généralement improbable vus les coûts importants liés au basculement vers un stockage entièrement basé sur des mémoires flash.

Nous soulignons que les recherches menées dans le cadre de cette thèse sont applicables pour les systèmes de stockage hybrides où les disques à base de mémoire flash sont utilisés comme un support de stockage complémentaire pour les disques durs magnétiques (voir figure 2.8 (C)).

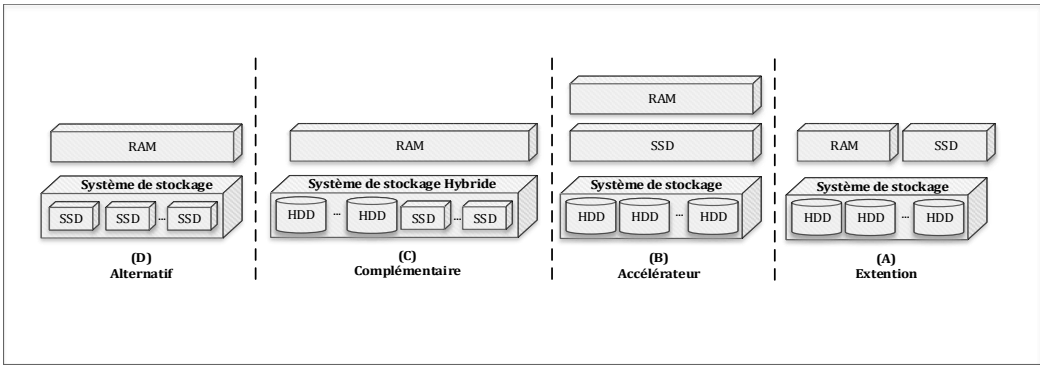

FIGURE 2.8 : Intégration des mémoires flash dans les serveurs

2.4 CONCLUSION

Dans ce chapitre, nous avons présenté les principaux concepts du *Cloud*, les différents services proposés par ce dernier et la virtualisation des ressources matérielles. Nous avons expliqué les différents types de virtualisation utilisés dans le Cloud Nous avons également présenté les deux SGBD (relationnel et NoSQL) utilisés par les fournisseurs du Cloud pour offrir un service DBaaS. Une section a été réservée pour présenter les *benchmarks* fréquemment utilisés pour simuler les transactions d'une base de données, qu'elle soit transactionnelle ou décisionnelle. Ensuite, nous avons présenté les principes de base des systèmes de stockage, notamment le stockage à base de mémoire flash et à base de mémoire magnétique. Pour chaque système de stockage nous avons décrit la technologie utilisée dans la fabrication des cellules, ces caractéristiques techniques et son mode de fonctionnement. Nous avons conclu le chapitre par présenter les différents modes d'intégration des mémoires flash dans les systèmes de stockage.

Les notions, les connaissances et outils introduits dans ce chapitre permettent une meilleure compréhension de nos travaux. Dans le chapitre suivant, nous présentons les travaux connexes à nos contributions.

Chapitre 3

ÉTAT DE L'ART

Dans ce chapitre, nous présentons les travaux qui ont une relation avec notre travail de recherche. Ce chapitre est divisé en deux grandes parties. La première partie est consacrée aux travaux relatifs à l'évaluation du coût de stockage, tandis que la seconde, est consacrée aux travaux s'intéressant au placement de données dans un système de stockage hybride. Dans chaque partie, nous commençons d'abord par une synthèse des travaux réalisés dans le domaine et ensuite nous réalisons une étude approfondie de ces travaux.

Sommaire

3.1 ÉVALUATION DU COÛT DE PLACEMENT DE DONNÉES

De nombreux travaux ont proposé des modèles pour évaluer le coût de placement de données dans un système de stockage. Le tableau 3.1 présente une synthèse de ces travaux. Les lignes représentent les travaux connexes tandis que les colonnes, exceptées la dernière, représentent l'ensemble des coûts pris en compte par ces travaux dans l'évaluation du coût de placement de données dans un contexte du Cloud. La dernière colonne est rajoutée pour préciser si l'approche en ligne propose un modèle de coût adapté aux bases de données. Nous considérons que le Cloud utilise un système de stockage hybride qui combine plusieurs classes de stockage. Le coût de placement de données dans un contexte de Cloud doit inclure selon l'état de l'art les coûts suivants :

- **Le coût de stockage :** Il inclut tous les coûts fixes et variables induits directement par le stockage des données dans le système de stockage. Ce coût regroupe le coût d'occupation, le coût d'endurance (ou de l'usure) et le coût d'énergie. Les approches prenant en compte ces coûts sont marquées par le symbole ✓dans les colonnes correspondantes.
- **Le coût de pénalité :** Un mauvais placement de données dans un contexte du Cloud peut engendrer une violation de contraintes de qualité de service des clients. En cas de violation, le fournisseur doit dédommager le client en appliquant, généralement, une remise sur la facture globale du client. Le montant de la remise, référé ici par la pénalité, dépend de plusieurs paramètres comme le degré de violation et sa durée [127]. Cette perte causée par la violation des contraires de qualité de service doit être pris en compte lors de l'évaluation du placement. Les travaux prenant en compte le coût de pénalité sont marqués par le symbole ✓dans la colonne correspondante.
- **Le Coût de migration :** Ce coût représente le coût de la mise en œuvre d'un placement. Un placement est achevé en migrant les données d'un d'un support à un autre. Cette migration engendre une charge de travail supplémentaire. Le coût de migration représente le coût d'exécution de cette charge de travail nécessaire pour la migration de données. Les travaux prenant en compte le coût de migration sont marqués par le symbole ✓dans la colonne correspondante.

TABLE 3.1 : Classification des travaux relatifs à l'évaluation du coût de placement de données.

Travaux	**Coût de stockage**			**Coût de la pénalité**	**Coût de la migration**		**base de données**
	occupation	énergie	endurance		énergie	endurance	
[43, 54, 71, 72]	✓	✓	✗	✗	✗	✗	✗
[132]	✓	✓	✗	✗	✗	✗	✓
[35]	✗	✗	✓	✗	✗	✗	✗
[78]	✓	✓	✓	✗	✗	✗	✗
[90]	✓	✗	✗	✗	✗	✗	✗
[97]	✗	✓	✓	✓	✓	✗	✗
[79, 82, 107, 115, 130]	✗	✗	✗	✗	✓	✗	✗
Notre modèle de coût	✓	✓	✓	✓	✓	✓	✓

'✓ ' signifie que les travaux référencés prennent en compte le coût, et '✗ 'signifie que les travaux référencés ne prennent pas en compte le coût en question.

3.1.1 *Le coût de stockage*

Plusieurs travaux ont proposé des modèles de coût permettant d'évaluer le coût de stockage des données. Dans [97], les auteurs proposent un modèle de coût pour évaluer le coût de placement d'une machine virtuelle dans un système de stockage hybride. Ce modèle de coût considère la diversité des périphériques de stockage, et calcule le coût de stockage des données en fonction de la consommation énergétique et de l'usure des périphériques de stockage. Cependant, le coût d'occupation qui représente le coût d'amortissement des périphériques de stockage (en comptabilité connu sous le terme dépréciation des actifs) est ignoré. Le coût d'amortissement désigne le coût de dégradation de l'état du périphérique du fait de son utilisation.

Dans [132], Zhang et *al.* tentent d'évaluer le coût de placement de données dans un système de stockage hybride pour une période donnée "T". Ils prennent en considération le cas d'un système de stockage hybride qui combine plusieurs classes de stockage. Ces classes ont des coûts différents en termes de Go/$ et aussi des performances différentes. Les auteurs ont modélisé le coût de placement en se basant seulement sur le coût d'amortissement. Comme annoncé auparavant, l'amortissement correspond à une constatation de perte de valeur d'un périphérique de stockage du fait de son usage. Ce modèle est très basique. Il ne prend pas en considération le coût de l'énergie consommée par les périphériques de stockage pour exécuter la charge de travail d'E/S transmise aux données stockées dans ces derniers. En outre, il ignore le coût d'usure du périphérique causé par l'exécution d'une charge de travail.

Le travail qui aborde une problématique proche de la nôtre est celui présenté dans [78]. Les auteurs proposent un modèle de coût regroupant le coût d'achat initial des périphériques de stockage et le coût total de possession (TCO pour *Total Cost of Ownership*). Le TCO inclut le coût d'énergie et le coût d'endurance. Ce modèle évalue le coût total du système de stockage pendant sa durée de vie, tandis que nous cherchons à évaluer le coût de placement de données pendant une période prédéfinie "T". En outre, le modèle n'est pas applicable dans un contexte de Cloud car il ne prend pas en considération le coût de pénalité engendré par un mauvais placement de données. Il faut noter que le modèle proposé ne prend pas en compte aussi le coût de migration des données.

Dans [43, 54, 71, 72], les auteurs ont proposé des modèles de coût évaluant le coût de la totalité du système de stockage et non un modèle évaluant le coût de stockage/placement de données. Ces modèles sont utilisés par les administrateurs de stockage avant de s'approvisionner. Ils déterminent les types et les quantités de périphériques de stockage devant être achetés. Ces modèles sont

employés pour effectuer un approvisionnement optimisé et adapté à l'entreprise. Leur décision est basée sur une caractérisation préalable de la charge de travail. Ils évaluent le coût du système de stockage par le coût d'achat et le coût d'énergie.

D'autres approches ont été proposées dans [79, 82, 107, 115, 130] pour modéliser le coût de placement. Elles considèrent seulement le coût de migration ou le coût d'usure [35]. Toutefois, ces propositions ne peuvent pas être appliquées pour évaluer précisément le coût de placement de données qui dépend de plusieurs facteurs tels que le coût d'occupation, d'endurance, d'énergie, de pénalité et de migration.

3.1.2 *Le coût de pénalité*

Nous avons présenté précédemment les coûts qui peuvent être induits par le stockage des données sur un périphérique de stockage (le coût d'occupation, le coût énergétique et le coût d'usure). Cependant, il peut y avoir des coûts indirects engendrés par un mauvais placement de données. Rappelons que dans un Cloud la qualité de service est garantie par un (SLA) qui regroupe les métriques nécessaires décrivant les performances souhaitées par le client (e.g. temps de réponse, latence, etc.) et celles quantifiant les ressources nécessaires pour exécuter le service telles que CPU, RAM, I/O, etc. La violation des métriques engendre des pénalités pour le fournisseur de Cloud. Le montant de la pénalité dépend généralement de deux paramètres : le degré de violation et la durée de violation.

Comme nous nous intéressons au système de stockage, nous ne concentrons que sur les métriques évaluant sa performance. Nous utilisons le nombre d'opération d'E/S par seconde (en anglais *IOPS* pour *input output per second*) comme métrique dans ce travail. L'IOPS est un paramètre communément utilisé pour évaluer les performances des applications interagissant avec les systèmes de stockage. Ce paramètre représente le débit de requêtes E/S traitées par le système de stockage. Cette étude pourra être élargie à d'autres métriques à l'image du taux de transfert de données (bande passante en Mo/s).

Il faut noter qu'un mauvais placement de données peut causer une violation de contraintes des applications interagissant avec le système de stockage. Cette violation engendre des pénalités, et donc un coût supplémentaire pour le Cloud. Peu de travaux dans la littérature prennent en considération le coût de la pénalité dans leurs modèles. Cependant, dans [47], les auteurs ont proposé un modèle de coût prenant en compte le coût de pénalité. Malheureusement, le modèle proposé présente certaines limites. En plus des inconvénients cités dans la section

3.1.1, ce modèle n'est pas adapté pour le service DBaaS. Dans ce travail, la pénalité est liée directement au placement de VM, cependant, dans un contexte du DBaaS, la pénalité est liée au client ayant un ensemble d'objets. Un travail supplémentaire est donc nécessaire pour dériver la pénalité au niveau d'un objet.

3.1.3 *Le coût de migration*

Un deuxième coût indirect lié au placement de données sera présenté dans cette section. Ce coût est lié à la mise en œuvre du placement. Il faut noter qu'afin d'atteindre un nouveau placement à partir du placement donné, un ensemble d'objet doivent être migrés d'un périphérique (*source*) à un autre (*destination*) [67]. Cette migration engendre une charge de travail supplémentaire qui consomme de l'énergie, et contribue à l'usure des périphériques. Le coût de migration représente le coût d'exécution de la charge de travail liée à l'opération de migration.

Dans [97], Ouarnoughi et *al.* ont proposé un modèle de coût qui prend en considération le coût de migration. Cependant, ils ignorent l'impact de l'ordre de migration des objets qui peut engendrer un taux d'erreur jusqu'à 8,25% du coût global de placement (voir chapitre 4 section 4.5). L'ordre de migration est l'un des paramètres les plus importants à prendre en considération dans un contexte de Cloud [95] (voir 4.5.3.4.

Il faut noter que l'opération de migration s'exécute de façon sérialisée (i.e. objet par objet). Plusieurs scénarios (séquences de migration) se présentent devant l'administrateur du Cloud. Il doit sélectionner la séquence optimale en terme du coût, à titre d'exemple *choisir la séquence qui favorise la migration des objets des clients ayant des pénalités élevées pourrait être intéressant en termes du coût car cette séquence permet d'éviter les pénalités les plus importantes en premier*. Par conséquent, le modèle de coût utilisé par le Cloud doit être en mesure d'évaluer l'ordre de migration.

3.1.4 *Discussion sur les modèles de coût proposés dans la littérature*

Après avoir décrit les différents modèles de coût de l'état de l'art, nous avons constaté les limites suivantes :

- Les modèles de coût ciblant l'évaluation du coût de stockage négligent le coût d'occupation.
- Les modèles de coût ignorent l'impact de l'environnement du *Cloud* (i.e. SLA, pénalité, migration), qui peut engendrer des coûts supplémentaires. Le contexte de *Cloud* implique différentes contraintes qui rendent le proces-

sus de modélisation de coûts plus complexe. Nous citons à titre d'exemple le coût de pénalité causée par la violation des contraintes de SLA. Ce coût devient de plus en plus contraignant pour les fournisseurs de Cloud actuels [127].

- Aucun modèle de coût, au moment de la rédaction du manuscrit et à notre connaissance, ne prend en considération l'impact de l'ordre de migration. Il faut noter que plusieurs séquences de migration sont possibles. L'administrateur du Cloud doit sélectionner la séquence de migration la plus optimale en termes de coût.
- Aucun modèle de coût, au moment de la rédaction du manuscrit et à notre connaissance, n'a été proposé dans la littérature pour évaluer le coût de placement des objets de DBaaS dans un système de stockage hybride.

3.2 PLACEMENT DE DONNÉES DANS UN SYSTÈME DE STOCKAGE HYBRIDE

Dans cette partie nous allons présenter les travaux de l'état de l'art traitant le problème de placement de données dans un système de stockage hybride. De nombreux travaux ont été proposés. Ils s'appuyant généralement dans leur décision sur le motif d'accès aux données et les caractéristiques des périphériques de stockage. La décision doit être prise dans un temps raisonnable (quelques secondes [132]) pour permettre à l'administrateur de réagir rapidement devant les situations de fluctuation .

Le tableau 3.2 présente une synthèse des travaux de la littérature. Les lignes représentent les travaux connexes tandis que les colonnes représentent les critères de classification des approches proposées. Ces critères peuvent être divisés en deux grandes catégories à savoir :

- Les critères qui ont une relation avec l'optimisation (i.e. objectif, contraintes).
- Les critères qui sont liés au domaine d'application et au contexte du Cloud.

Notre analyse de l'état de l'art est basée sur cinq dimensions à savoir :

- *L'objectif du placement :* Une stratégie est exécutée pour trouver un placement qui répond à un ou à plusieurs objectifs. Nous constatons que les stratégies de placement proposées dans la littérature diffèrent selon l'objectif. Une classification selon ce dernier est présentée dans le tableau 3.2. La première colonne du tableau illustre les différents objectifs. Trois objectifs différents ont été constatés : 1) la minimisation des coûts, 2) la maximisation des performances et 3) la satisfaction des SLA. Rappelons que dans

un contexte du Cloud, l'optimisation répond à des objectifs multiples. Le fournisseur de Cloud cherche à satisfaire les SLA de ses clients à moindre coût où l'optimisation des performances fait partie. Le symbole ✓ dans le tableau 3.2 signifie que les travaux référencés prennent en compte l'objectif précisé en colonne.

- *Les contraintes considérées :* Une stratégie de placement est menée généralement en s'assumant de la satisfaction de plusieurs contraintes. Trois contraintes ont été soulevées dans la littérature à savoir : 1) une contrainte de stockage pour garantir que la taille de l'ensemble d'objets hébergés dans un disque ne dépasse pas la capacité totale de stockage de celui-ci. Cette contrainte n'est pas mentionnée dans le tableau 3.2 car elle est considérée par toutes les stratégies proposées, 2) une contrainte de performance qui signifie que les performances nécessaires pour traiter les requêtes transmises aux objets hébergés dans un disque, ne doivent pas dépasser les performances de ce dernier et 3) une dernière contrainte concernant les SLA des clients dont l'objectif est d'assurer que les exigences des clients en termes de qualité de service soient satisfaites.

- *L'intégration de la pénalité :* Comme expliqué précédemment, la pénalité est l'un des paramètres les plus importants pour les fournisseurs de Cloud [127]. Le tableau 3.2 synthétise les travaux qui intègrent ce critère.

- Le domaine d'application : Une information est rajoutée pour préciser si la stratégie proposée est adaptée aux applications de base de données

- La prise en compte de la multi-location : La dernière colonne mentionne si les travaux référencés en ligne prennent en compte l'architecture multi-locataire ou pas.

TABLE 3.2 : Classification des travaux relatifs au placement de données dans un système de stockage hybride selon leurs objectifs, les contraintes prises en compte, le contexte (base de données, multi-locataire) et l'intégration de la pénalité.

	Objectifs			**Contraintes**		**base de données**	**Multi-locataire**	**Pénalité**
	SLA	Coût	Performance	SLA	Performance			
[132]	✗	✓	✗	✓	✗	✓	✗	✗
[33, 116]	✗	✗	✓	✗	✗	✓	✗	✗
[24, 34, 66, 106, 123, 125, 126]	✗	✗	✓	✗	✗	✗	✗	✗
[68, 80, 84]	✗	✗	✓	✗	✗	✓	✗	✗
[71]	✗	✓	✓	✗	✓	✗	✗	✗
[54]	✗	✓	✓	✗	✗	✓	✗	✗
Nos stratégies	✓	✓	-	✓	✓	✓	✓	✓

✓ signifie que les travaux référencés prennent en compte les critères en question, et ✗ signifie que les travaux référencés en ligne ne prennent pas en compte les critères en question.

3.2.1 *L'objectif du placement*

De nombreux travaux de recherche traitant le problème du placement de données dans un système de stockage hybride ont été proposés dans la littérature (voir le tableau 3.2). La plupart d'entre eux sont orientés performance, autrement dit, leur objectif principal est d'optimiser les performances [24, 33, 34, 66, 68, 80, 84, 106, 116, 123, 125, 126]. Malheureusement, ces approches ne sont pas applicables dans un contexte de Cloud car l'objectif n'est pas le même. Dans un contexte Cloud, la solution doit être orientée coût, avec prise en compte de l'optimisation des performances. L'objectif d'un administrateur de Cloud est de trouver un placement qui satisfait les contraintes de ses clients et minimise le coût global pour le fournisseur de Cloud.

Dans [54, 71], les auteurs ont proposé des solutions d'optimisation du coût et des performances. Cependant, le coût optimisé n'est pas celui du placement (le coût d'exploitation) mais celui d'achat (le coût d'acquisition des périphériques de stockage). Ces travaux cherchent à trouver une combinaison adéquate des périphériques de stockage à acheter. Ces approches sont destinées à optimiser le coût d'approvisionnement. La décision est basée sur : 1) une caractérisation de la charge de travail du centre de données et 2) les caractéristiques des périphériques de stockage.

Dans [132], les auteurs ont proposé une approche de placement orientée coût pour les applications de base de données. Leur modèle de coût considère la diversité des périphériques de stockage. Ils supposent que le système de stockage combine plusieurs périphériques de stockage qui ont différents coûts en termes de Go/$ et différentes performances. Les auteurs ont proposé une approche de placement d'objets de base de données nommée « DOT », dont le but est d'optimiser le coût d'exploitation total TOC (TOC, pour *total operating cost*) et de satisfaire les SLA de différentes charges de travail. Ce travail est le plus proche du nôtre, cependant, l'approche proposée n'est pas adaptée pour un contexte multi-locataire où plusieurs clients ayant des charges de travail d'E/S différentes, des exigences différentes en termes d'IOPS et des pénalités différentes, partagent le même système de stockage. De plus, leur modèle de coût ne prend pas en considération la pénalité qui résulte de la violation de contraintes.

Les auteurs dans [33] ont proposé un outil appelé "Object Placement Advisor" (OPA) pour aider l'administrateur à trouver un bon placement d'objets dans un système de stockage hybride. La solution est destinée aux applications de base de données. Elle est intégrée sous forme d'un module dans le SGBD DB2 d'IBM. La décision est basée sur les statistiques du SGBD et les caractéristiques

des périphériques de stockage. Cependant, cette stratégie met l'accent sur la maximisation des performances et ne prend pas en compte certains aspects clés du contexte Cloud (e.g. SLA, la pénalité, multi-locataire, etc).

3.2.2 *Les contraintes*

Le problème de placement est généralement formulé par un problème d'optimisation sous un certain nombre de contraintes. Une solution est considérée comme faisable (acceptable) si et seulement si toutes les contraintes d'optimisation sont vérifiées. Dans un contexte de Cloud, généralement chaque client a une contrainte *soft* nommée *soft SLA* désignant la qualité de service souhaitée et une contrainte *hard* nommée *hard SLA* définissant la dégradation de service tolérable par ce dernier [89]. Ces contraintes *soft et hard* sont définies dans le SLA du client. Trois contraintes sont donc à respecter à savoir :

- *Les contraintes de stockage :* Afin d'assurer que la taille de l'ensemble d'objets stockés dans un périphérique donné ne dépasse pas sa capacité totale de stockage.
- l*Les contraintes de performance :* De la même manière, la deuxième contrainte garantit que la capacité de performance (IOPS) demandée par l'ensemble d'objets stockés dans un périphérique donné ne dépasse pas sa capacité de performance totale.
- *Les contraintes de SLA :* Afin d'assurer que les contraintes de qualité de service demandées par le client sont satisfaites.

Le tableau 3.2 synthétise les approches proposées, nous observons qu'aucune d'entre elles n'a pris en considération l'ensemble des contraintes d'optimisation.

3.2.3 *L'intégration de pénalité*

Comme décrit précédemment, une approche destinée au Cloud doit être orientée coût. L'optimisation de performances est considérée comme un second objectif. La pénalité est l'un des aspects les plus importants du Cloud et elle représente un réel problématique pour les fournisseurs actuels [127]. L'impact de celle-ci sur l'évaluation de coût est consédérable [97]. Très peu d'approches ont modélisé le coût de pénalité engendrée par la violation des contraintes. Dans [97], les auteurs proposent un modèle de coût modélisant le coût de la pénalité. Cependant, ce travail n'est pas adapté pour un contexte de DBaaS et présente certaines limites expliquées dans la section 3.1.1.

3.2.4 *Discussion sur les modèles de coût proposés dans la littérature*

Après avoir décrit les différentes techniques de placement proposés dans la littérature, nous avons constaté que très peu de travaux étaient orientés coût et que la problématique de placement est vue sous le même angle qui est l'optimisation de performance du système de stockage. Deux études seulement [95, 132] ciblent l'optimisation de coût. Cependant, les auteurs dans [132] ne s'intéressent qu'au coût de stockage et ne prennent pas en compte le contexte du Cloud. En outre, la solution s'exécute dans un temps non raisonnable pour les grandes instances du problème où le nombre d'objets est important. Cela le disqualifie pour être utilisée en temps réel (runtime). Par ailleurs, le modèle proposé dans [95] n'est pas adapté pour un contexte DBaaS. Les auteurs utilisent une granularité très large et potentiellement imprécise (c.à.d. différents motifs d'accès sont appliqués dans différents endroits ce qui rend difficile de sélectionner une classe de stockage pour l'objet).

Dans ce travail de thèse nous proposons des stratégies de placement orientées coût où l'optimisation des performances fait partie. Ces stratégies prennent en compte simultanément les contraintes liées à l'environnement Cloud (i.e. SLA et pénalité, migration), et aux périphériques de stockage (i.e. consommation énergétique, usure du périphérique). Nous avons utilisé une granularité plus fine, au niveau d'un objet, afin d'atteindre ces objectifs avec précision. Les solutions proposées s'exécutent dans un temps raisonnable (quelques secondes [132]) afin d'être utilisées dans des systèmes en temps réel.

3.3 CONCLUSION

Bien que de nombreuses approches ont été proposées dans la littérature que ce soit pour évaluer le coût de placement, ou pour optimiser le placement dans un système de stockage hybride, ces deux problématiques sont loin d'être résolues et restent un sujet actif. En effet, la prise en compte des concepts de Cloud rend les problématiques plus complexe.

Pour ce qui concerne l'évaluation du coût de placement, nous avons présenté les travaux traitant du problème de la modélisation des coûts dans les centres de données. Ces modèles ont été divisés selon des paramètres pris en compte en trois classes à savoir : les modèles de coût qui incluent 1) le coût de stockage, 2) le coût de pénalité et 3) le coût de migration. Nous avons constaté que les modèles de coût proposés ne prennent pas en compte le coût de migration et l'impact de l'environnement Cloud.

Pour la problématique de placement, nous avons classé les approches de placement des données selon 1) les objectifs visés, 2) les contraintes d'optimisation prises par ces derniers, 3) la prise ou non des contraintes liées à l'environnement Cloud (i.e. SLA et pénalité) et 4) le domaine d'application. Nous avons constaté que les stratégies de placement proposées sont orientées performance alors que dans le Cloud la stratégie de placement devrait être orientée coût.

La partie suivante de ce manuscrit présente l'ensemble de nos contributions visant à évaluer et à placer les données dans un système de stockage hybride.

Deuxième partie

CONTRIBUTIONS

Chapitre 4

MODÈLE DE COÛT POUR LE PLACEMENT DE DONNÉES DANS UN SERVICE DBAAS

Une bonne stratégie de placement doit répondre aux contraintes des clients à un coût minimum. Il peut y avoir plusieurs placements qui satisfassent les SLAs des clients, cependant, le but est de les évaluer afin de sélectionner le placement optimal en termes de coût monétaire ($). Pour ce faire, l'administrateur du Cloud doit faire appel à une méthode de calcul ou à un modèle de coût permettant d'évaluer un placement donné. Dans ce chapitre, nous proposons un modèle de coût pour évaluer le coût de placement des objets de DBaaS dans un système de stockage hybride.

Notre méthodologie pour la définition du modèle de coût est structurée en trois phases essentiels à savoir :

1. *Énumération de tous les coûts liés au placement de données* : lister tous les coûts (fixes ou variables, directs ou indirects) qui peuvent être engendrés par un placement donné. Ces coûts peuvent être induits par : 1) le stockage de données sur les périphériques de stockage et 2) l'exécution du placement et 3) les conséquences d'un mauvais placement (e.g. les pénalités) .

2. *Définition d'une méthode de calcul* : pour chaque coût (composant) énuméré dans l'étape précédente, une méthode de calcul est définie. Dans notre contexte, nous avons : 1) modélisé les aspects liés au Cloud en tenant compte les besoins des clients en termes de qualité de service spécifiée dans le SLA, et 2) déterminé tous les paramètres à prendre en compte et qui peuvent avoir un impact sur les coûts

3. *Évaluation du modèle de coût proposé* : pour chaque composant dans le modèle de coût, nous avons évalué sa pertinence et l'exactitude de la méthode de calcul proposée. Nous avons aussi comparé notre modèle de coût avec l'état de l'art pour montrer sa précision.

De nombreux modèles de coûts ont été proposés dans les travaux de l'état de l'art. Nous avons présenté dans le chapitre 3 une étude approfondie des travaux de la littérature avec une classification des approches (voir tableau 3.1 39). Ces modèles ne sont pas adaptés pour le Cloud, et cela pour les différentes raisons énumérées dans la section 3.1.4.

Après exploration des coûts liés au stockage des objets, nous avons constaté qu'ils dépendent de trois facteurs principaux :

1. *L'objet :* le profil d'accès aux objets (type et nature des opérations E/S).
2. *Les périphériques de stockage :* les types et les caractéristiques du périphérique de stockage contenant l'objet.
3. *L'environnement Cloud :* le prix des services Cloud et les contraintes de qualité de service.

Notons qu'il existe certains coûts qui ne dépendent pas du stockage des objets ni de l'environnement cloud (e.g. construction ou location des locaux, ressource humaine, refroidissement,etc). Dans notre modèle, nous considérons que le coût de ces derniers est constant pour l'ensemble de objets présent dans le centre de données.

Nous avons évalué notre modèle de coût sur une plateforme réelle avec un système de stockage hybride qui combine deux classes de stockage HDD et SSD.

Le modèle de coût que nous avons proposé a été publié dans la conférence internationale DEXA *27th The International Conference on Database and Expert Systems Applications (DEXA 2016)* [28].

Ce chapitre est organisé comme suit. Premièrement nous donnons un aperçu global du modèle de coût et de ses principaux composants. La deuxième, la troisièmes, et la quatrième section détaillent les différents composants du modèle de coût. Nous présentons l'évaluation de ce modèle dans la cinquième section avant de conclure ce chapitre.

Sommaire

4.1 COÛT DU PLACEMENT DANS LE CLOUD

Avant de détailler notre modèle de coût, nous définissons ci-après les termes utilisés et ensuite, nous présentons les hypothèses sur lesquelles se base notre proposition.

4.1.1 *Notations*

Le tableau 4.1 présente les notations utilisées dans ce chapitre pour la modélisation du coût de placement. A noter que l'unité de calcul des coût est le $.

Notation	**Description**
$Cost_{pl,T}$	Coût total du placement
$Cost_{stg,T}$	Coût du stockage
$Cost_{pnl,T}$	Coût de la pénalité
$Cost_{mgr,T}$	Coût de la migration
$Cost_{occ,T}$	Coût d'occupation
$Cost_{w,T}$	Coût de la charge de travail
$Cost_{mng,T}$	Coût du management
p_{d_j}	Prix du périphérique de stockage d_j

TABLE 4.1 : Notations

4.1.2 *Hypothèses*

Nous présentons ci-dessous les hypothèses ayant servies de base pour notre travail.

1. Chaque client détient sa propre base de données, sa propre charge de travail, son propre SLA et sa propre fonction de pénalité.
2. Une seule instance d'application est utilisée pour servir les clients (architecture multi-locataire). Autrement dit, les clients partagent la même instance SGBD et les mêmes ressources physiques (CPU, RAM, système de stockage).
3. Le système de stockage combine deux classes de stockage HDD et SSD. Ces classes de stockage sont visible par le SGBD.

4. Le processus de migration s'exécute en arrière-plan et ne consomme que la ressource d'E/S disponible afin de ne pas impacter les charges de travail des clients.

5. Les requêtes d'E/S transmises à l'objet o_{i,u_k} durant sa migration sont traitées par l'un des périphériques de stockage source ou destination afin d'éviter l'indisponibilité du service.

6. La migration est sérialisée, c'est à dire objet par objet, un seul objet est migré à la fois.

7. Les opérations de lecture et d'écriture, générées par le processus de migration, se font d'une manière séquentielles. Cependant, l'opération d'écriture se fait en parallèle avec l'opération de lecture.

4.1.3 *Définitions*

Objet : nous désignons par objet toute entité logique de la base de données (exemple : table, index, fragment,etc.) [33].
Migration : l'opération de migration consiste à transférer un ensemble d'objets d'un disque *source* à un autre disque *destination*, c'est à dire l'opération de changement du placement [54].
Placement : le placement est vu comme une fonction de correspondance (en anglais *mapping*) entre les objets et les périphériques de stockage constituant le système de stockage [132].

4.1.4 *Problème d'évaluation du coût de placement*

Pour illustrer le problème d'évaluation du coût de placement d'objets dans un contexte Cloud, nous considérons le scénario de motivation suivant : un administrateur de Cloud dispose d'un système de stockage hybride qui combine deux classes de stockage HDD et SSD. Ces classes de stockage présentent des performances différentes et des coûts différents en termes de Go/$. Le système de stockage est partagé entre plusieurs clients (utilisateurs). Les clients ont différentes bases de données qui regroupent un ensemble d'objets, différentes charges de travail et différents SLA.
Supposons que l'administrateur Cloud cherche un placement d'objets optimisé qui s'inscrit dans la stratégie globale du Cloud qui est la satisfaction de la clientèle à moindre coût. Pour ce faire, il devra évaluer tous les placements possibles, puis sélectionner celui qui répond aux contraintes des clients à moindre coût. *La question est comment évaluer le coût de placement des objets.*

Pour répondre à cette question, nous proposons un modèle de coût pour évaluer le coût de placement des objets de DBaaS.

4.1.5 *Spécificité du Cloud*

Dans les entreprises traditionnelles où le système de stockage est utilisé par des applications internes (c'est à dire il n'est pas destiné pour être loué à d'autres entités), le coût de placement est étroitement lié au coût de stockage de données sur les périphériques de stockage. Cependant, dans le cas du Cloud, le système de stockage est utilisé pour servir d'autres clients, qui sont des entités externes, sous un certain nombre de contraintes définies au préalable dans les SLA. La violation de ces contraintes engendre des pénalités, par conséquent des pertes financières pour l'entreprise locatrice (Cloud). In faut noter qu'un mauvais placement de données peut conduire à une violation de contraintes des SLA, et donc à des pénalités. Le modèle de coût destiné au Cloud doit prendre en considération les aspects de celui-ci (i.e. SLA, les pénalités)

4.1.6 *Évaluation du coût de placement*

Nous définissons ci-après un modèle permettant d'évaluer le coût d'un placement donné (nommé placement cible) en supposant que ce dernier est obtenu en déplaçant un ensemble de M objets à partir du placement actuel (initial). Nous considérons que l'opération de migration est sérialisée. Cette sérialisation génère une séquence de placements temporaires comme illustré dans la figure 4.2. Soit $overall_{cost}$ le coût de placement pour la période T (la période d'évaluation). $overall_{cost}$ est égal à la somme des coûts de placement temporaire ($Cost_{pl,t_m}$) comme l'indique l'équation 1. $Cost_{pl,t_m}$ désigne le coût de placement temporaire qui dure un temps t_m, sachant que t_m représente le coût de migration du m-ième objet. Il faut noter que l'ordre de migration a un impact sur le coût global de placement.

$$overall_{cost} = \sum_{m=0}^{M} \left(Cost_{pl,t_m} \right) \tag{1}$$

Dans les sections suivantes, nous allons détailler le modèle de coût évaluant le coût de placement.

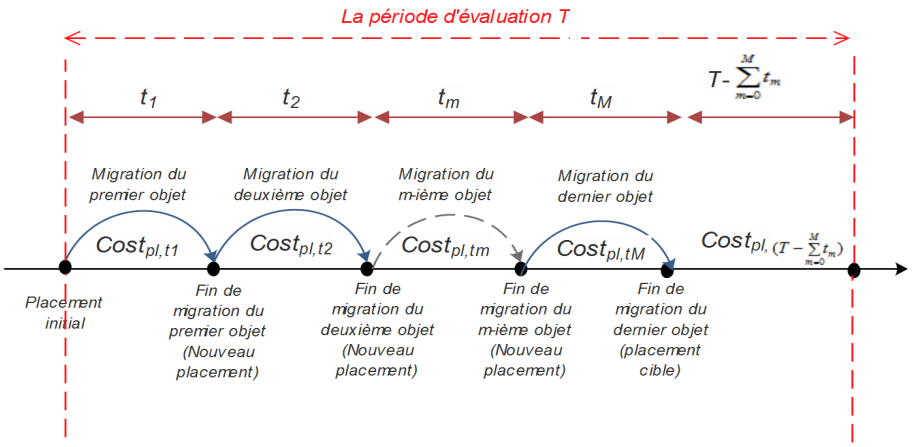

FIGURE 4.1 : Séquence de placements temporaires générés par un placement

4.1.7 *Aperçu du modèle de coût*

La figure 4.2 donne un aperçu de notre modèle de coût sous une forme hiérarchique. Nous avons classé les coûts liés au placement d'objets en trois grandes catégories : (1) le coût du stockage d'objets dans les périphériques de stockage, (2) le coût lié à la pénalité découlant de la violation des SLA (3) le coût lié à l'exécution du placement qu'on représente par le coût de migration des données.

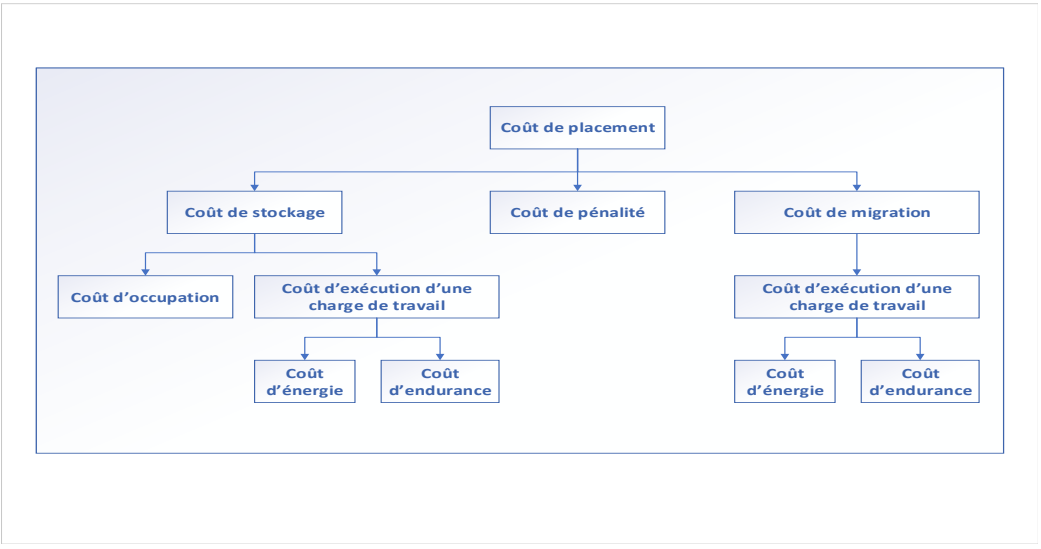

FIGURE 4.2 : Composants du modèle de coût

L'équation 2 présente la fonction principale de calcul du coût de placement pour une période de temps T donnée. T désigne la période d'évaluation. Soit $Cost_{pl,T}$ le coût de placement d'objets pour la période T. $Cost_{pl,T}$ est égal à la

somme du : 1) coût de stockage ($Cost_{stg,T}$), 2) coût de pénalité ($Cost_{pnl,T}$) et 3) coût de migration ($Cost_{mgr,T}$) comme le montre l'équation 2. Nous soulignons que l'équation 2 reste valide dans le cas d'un placement initial (un premier placement) et cela en mettant le coût de migration à zero.

$$Cost_{pl,T} = Cost_{stg,T} + Cost_{pnl,T} + Cost_{mgr,T} \tag{2}$$

4.2 COÛT DE STOCKAGE

Le coût de stockage comprend trois composants à savoir : (1) le coût d'occupation, (2) le coût d'exécution de la charge de travail et (3) le coût de gestion. Le coût d'occupation ($Cost_{occ,T}$) représente le coût de stockage des objets dans le système de stockage pour une période T. Le coût d'exécution de la charge de travail ($Cost_{w,T}$) est lié au coût des performances requises pour traiter les requêtes d'E/S transmises aux objets pour la même période T.

Le coût de gestion ($Cost_{mng,T}$) correspond à tous les coûts fixes qui ne dépendent ni du stockage d'objets ni de l'environnement Cloud. Cette partie peut inclure les coûts liés à l'installation des ressources matérielles, la location des locaux des centres de données, la ressource humaine, etc.

L'équation 3 ci-dessous présente le coût de stockage de données dans le système de stockage. Nous détaillons les sous-coûts liés à celui-ci ($Cost_{occ,T}$, $Cost_{w,T}$ et $Cost_{mng,T}$) dans les sections suivantes.

$$Cost_{stg,T} = Cost_{occ,T} + Cost_{w,T} + Cost_{mng,T} \tag{3}$$

4.2.1 *Coût d'occupation*

L'équation 4 illustre le coût d'occupation des objets stockés dans le système de stockage pour une période T ($Cost_{occ,T}$). $Cost_{occ,T}$ représente la somme des coûts d'occupation des périphériques de stockage qui le constituent, comme le montre l'équation 4.

$$Cost_{occ,T} = \sum_{j=1}^{J} \left(Cost_{occ,d_j,T} \right) \tag{4}$$

Nous calculons le coût d'occupation d'un périphérique de stockage d_j pour une période T ($Cost_{occ,d_j,T}$) par la multiplication de son coût d'amortissement

($Cost_{amz,T}(d_j)$), et la proportion du volume occupée par les objets qu'il contient comme l'indique l'équation 5.

$$Cost_{occ,d_j,T} = \left(\frac{\sum_{o_{i,u_k} \in O_{d_j}} (S_{o_{i,u_k}})}{C_{d_j}} \right) * \left(Cost_{amz,T}(d_j) \right) \tag{5}$$

Le ratio $\frac{\sum_{o_{i,u_k} \in O_{d_j}} (S_{o_{i,u_k}})}{C_{d_j}}$ représente la proportion du volume occupée par l'ensemble d'objets placés dans le périphérique d_j. $Cost_{amz,T}(d_j)$ désigne le coût d'amortissement du périphérique d_j pour la période T.

Nous obtenons $Cost_{amz,T}(d_j)$ en multipliant le coût unitaire d'amortissement ($Cost_{amz,1}$) par le temps T comme exprimé par l'équation 6. $Cost_{amz,1}$ est obtenu en répartissant le coût d'achat du périphérique de stockage $d_j(p_{d_j})$ sur la période d'amortissement. La période d'amortissement représente la période de garantie du périphérique de stockage d_j (e.g. 5 ans) [43].

$$Cost_{amz,T}(d_j) = T * Cost_{amz,1}(d_j) \tag{6}$$

4.2.2 *Coût d'exécution d'une charge de travail*

Le coût d'exécution de la charge de travail désigne tous les coûts induits par l'exécution des requêtes d'E/S transmises aux objets stockés dans le système de stockage. Ce coût exclut le coût d'achat des disques qui est pris en compte dans le coût d'occupation. L'exécution des requêtes d'E/S consomme de l'énergie et affecte l'endurance des périphériques. Par conséquent, nous calculons le coût d'exécution de la charge de travail transmise au système de stockage à partir du coût d'énergie et du coût d'usure comme le montre l'équation 7.

$$Cost_{w,T}(d_j) = Cost_{erg,T} + Cost_{edr,T} \tag{7}$$

Le coût énergétique $Cost_{erg,T}$ représente le coût de l'énergie consommée par le système de stockage afin d'exécuter la charge de travail d'E/S transmise à ce dernier. Le coût d'usure (endurance) $Cost_{edr,T}$ désigne le coût de dégradation de l'état de santé du système de stockage causé par l'exécution de cette charge de travail. Nous donnons plus de détails dans les sections suivantes.

4.2.2.1 *Coût énergétique*

Le coût énergétique est obtenu en multipliant le prix unitaire de l'énergie (E_{up}), par la quantité d'énergie consommée par le système de stockage pour la période T ($E_{ss,T}$) comme l'indique l'équation 8.

$$Cost_{w,T}(d_j) = E_{ss,T} * E_{up,T} \tag{8}$$

La quantité d'énergie consommée par le système de stockage pour la période T est égale à la somme des quantités d'énergie consommées par l'ensemble des périphériques de stockage qui le constituent. L'équation 9 illustre la quantité d'énergie consommée par le système de stockage.

$$E_{ss,T} = \sum_{j=1}^{J} \left(E_{d,T}(d_j) \right) \tag{9}$$

L'énergie consommée par un périphérique de stockage d_j pour une période de temps T est l'intégrale de sa puissance sur cette période comme le montre l'équation 10. La modélisation de la consommation énergétique du CPU et de la RAM n'est pas prise en compte par notre modèle de coût.

$$E_{d,T}(d_j) = \int_T P(d_j) \tag{10}$$

Par conséquent, nous obtenons l'énergie consommée par un périphérique de stockage d_j à partir de la puissance électrique appliquée sur ce dernier et du temps T comme présenté par l'équation 11.

$$E_{d,T}(d_j) = P(d_j) * T \tag{11}$$

Un périphérique de stockage d_j a plusieurs états de puissance. Ces états changent en fonction de son activité. Nous distinguons trois états pour un disque dur HDD à savoir : 1) le mode actif, 2) le mode inactif (en anglais *idle*) et 3) le mode en veille (en anglais *standby*), avec les puissances électriques P_{atv}, P_{idl}, P_{sdb} respectivement où P_{atv}>P_{idl}>P_{sdb} [61]. Il faut noter que le disque consomme plus d'énergie dans l'état actif à cause des mouvements mécaniques (rotation des plateaux et déplacement de la tête), un peu moins dans l'état inactif (arrêt du mouvement de la tête et ralentissement de la vitesse de rotation). Il passe en mode veille, aprés une certaine durée d'inactivité, afin de réduire la consommation énergétique et cela en gardant une faible énergie pour alimenter le contrôleur tout en arrêtant les mouvements mécaniques. Un disque SSD peut avoir les mêmes modes de puissance électrique [39, 40] mais avec des valeurs différentes

de celles d'un disque HDD vu l'absence de mouvements mécaniques.

Plusieurs études ont montré que mettre un disque HDD en mode veille n'est pas intéressant en termes de consommation énergétique dans les grands datacenter [55, 61, 113]. Cela revient à la consommation importante d'énergie causée par l'opération de transition d'un mode de veille à un mode actif. Dans ce travail, nous supposons qu'un périphérique de stockage possède deux états : un état actif et un état inactif (voir l'équation 12).

$$P(d_j) = \begin{cases} P_{atv,d_j} \text{ si le périphérique est actif} \\ P_{idl,d_j} \text{ si le périphérique est inactif} \end{cases} \tag{12}$$

Nous calculons la quantité d'énergie consommée par un périphérique de stockage d_j pour la période T à l'aide des puissances dans les différents états (actif, inactif) et le temps passé dans chaque état. Soit $t_{atv}(d_j)$ (respectivement $t_{idl}(d_j)$) le temps passé par le disque d_j dans un état actif (respectivement inactif). L'équation 13 montre comment obtenir la consommation énergétique d'un disque en fonction de ses états.

$$E_{d,T}(d_j) = \left(P_{atv,d_j} * t_{atv}(d_j)\right) + \left(P_{idl,d_j} * t_{idl}(d_j)\right) \tag{13}$$

Dans ce travail, nous avons utilisé un modèle de puissance simple qui considère des valeurs de puissance constantes pour un périphérique de stockage donné. Il faut noter que dans [97], les auteurs ont prouvé que la puissance dépend aussi du type d'opérations.

$t_{atv}(d_j)$ représente le temps nécessaire pour exécuter la charge de travail d'E/S transmise aux objets stockés dans le disque d_j, tandis que $t_{idl}(d_j)$ est égal à la différence entre la période d'évaluation T et $t_{atv}(d_j)$ comme exprimé par l'équation 14.

$$t_{idl}(d_j) = T - t_{atv}(d_j) \tag{14}$$

Nous pouvons à présent estimer le coût énergétique d'un système de stockage, pour la période T, lié à l'exécution de la charge d'E/S en utilisant l'équation 15.

$$E_{d,T}(d_j) = \left(\sum_{j=1}^{J} \left(\left(P_{atv,d_j} * t_{atv}(d_j)\right) + \left(P_{idl,d_j} * t_{idl}(d_j)\right)\right)\right) * E_{up,T} \tag{15}$$

Le temps passé par le disque d_j dans l'état actif représente le temps nécessaire pour exécuter les requêtes d'E/S transmises aux objets qu'il les contient. Soit

$t_{exe,o_{i,u_k}}$ le temps nécessaire pour exécuter les opérations d'E/S transmises à l'objet o_{i,u_k}. $t_{atv}(d_j)$ est calculé comme le montre l'équation 16.

$$t_{atv}(d_j) = \sum_{o_{i,u_k} \in O_{d_j}} (t_{exe,o_{i,u_k}}) \tag{16}$$

Le temps nécessaire pour exécuter les opérations d'E/S transmises à l'objet o_{i,u_k} ($t_{exe,o_{i,u_k}}$) est obtenu à l'aide du nombre des requêtes d'E/S transmises à ce dernier, et le temps de réponse du périphérique de stockage d_j contenant ce dernier. Soit $req_{op,o_{i,u_k}}$ le nombre de requêtes de type op transmis à l'objet o_{i,u_k} et soit t_{op,d_j} le temps de réponse du disque d_j pour le type d'opération op. $t_{exe,o_{i,u_k}}$ est obtenu comme suit :

$$t_{exe,o_{i,u_k}} = \sum_{op \in OP} (req_{op,o_{i,u_k}} * t_{op,d_j}) \tag{17}$$

Nous obtenons $req_{op,o_{i,u_k}}$ à partir des traces enregistrées durant l'étape de monitoring et la caractérisation de la charge de travail d'E/S.
Les performances d'un périphérique de stockage d_j (t_{op,d_j}) peuvent être obtenues de deux façons : (1) à partir des fiches techniques des périphériques de stockage, ou bien (2) à l'aide de mesures dans une phase de calibration. Nous soulignons qu'il y a une asymétrie des performances pour un périphérique de stockage donné. Le temps de réponse d'un disque varie principalement en fonction de deux paramètres de la charge d'E/S :

1. Le type d'accès (lecture/écriture).
2. Le motif d'accès (séquentiel/aléatoire).

Dans notre cas, nous avons conduit une série d'expérimentations durant une phase préparatoire appelée *la phase de calibration* afin de déterminer le débit d'un périphérique de stockage d_j pour chaque type d'opération op, OP $\in$ {RR,SR,RW,SW}. Cette phase sera détaillée dans la partie traitant l'évaluation.

Jusque-là, nous avons décrit le coût de la consommation énergétique lié à l'exécution de la charge de travail d'E/S transmise au système de stockage pour la période T. La prochaine section présente le coût d'endurance du système de stockage, ce qui complète le coût total d'exécution d'une charge de travail tel que présenté dans l'équation 7.

4.2.2.2 *Coût d'endurance*

Le coût d'endurance (d'usure) d'un système de stockage provoqué par une charge de travail d'E/S correspond au coût de dégradation de ses périphériques

de stockage. Le calcul du coût d'usure d'un périphérique donné est souvent basé sur l'estimation de la durée de vie de ce périphérique.

L'usure d'un périphérique de stockage dépend de la technologie utilisée. Elle est considérée comme l'une des contraintes les plus importantes des périphériques de stockage à base de mémoire flash. Les disques SSD supportent un nombre limité d'opérations d'écriture/effacement. Les cellules de stockage deviennent inaccessibles une fois le seuil de tolérance franchi. L'usure des disques SSD est liée à la fois au trafic d'écriture et à l'amplification interne en écriture. Cependant, le temps de service (le cycle de service) et la température sont identifiés comme les principaux paramètres qui affectent la durée de vie d'un disque HDD et son contrôleur.

Le coût d'usure d'un système de stockage pour une période donnée T ($Cost_{edr,T}$) représente la somme des coûts d'endurance des périphériques de stockage qui le constituent comme le montre l'équation 18.

$$Cost_{edr,T} = \sum_{j=1}^{J} Cost_{edr,d,T}(d_j) \tag{18}$$

Le coût d'usure d'un périphérique de stockage d_j ($Cost_{edr,d,T}(d_j)$) correspond au coût de dégradation causée par l'exécution de la charge de travail d'E/S transmise à ce dernier. $Cost_{edr,d,T}(d_j)$ est obtenu à partir du degré d'usure causé par l'exécution de la charge de travail d'E/S ($Wo_{dgree}(d_j)$), et le prix d'achat du périphérique de stockage d_j (p_{d_j}). L'usure d'un périphérique de stockage d_j est donnée par l'équation 19.

$$Cost_{edr,d,T}(d_j) = Wo_{dgree}(d_j) * p_{d_j} \tag{19}$$

Le degré d'usure est calculé à partir de la durée de vie consommée par l'exécution de la charge de travail d'E/S (wo_{d_j}) et la durée de vie totale du périphérique de stockage d_j (wo_{d_j}) comme l'indique l'équation 20.

$$Wo_{dgree}(d_j) = \frac{wo_w(d_j)}{wo_{d_j}} \tag{20}$$

Comme annoncé ci-dessus, wo_{d_j} dépend de la technologie utilisée dans le périphérique de stockage d_j. Nous mesurons la durée de vie des disques SSD par le nombre total d'opérations d'écriture qu'il peut supporter. Cette information est inscrite dans la fiche technique de chaque périphérique de stockage d_j de type SSD. Nous supposons que le SSD est complètement usé et qu'il doit être changé dès que nous atteignons la quantité de données à écrire pendant la durée de

garantie (e.g. 600TB à écrire pendant 5 ans de garantie pour un SSD de 128GB [39, 40]). De ce fait, la quantité de données écrite par la charge de travail d'E/S doit être quantifiée afin de calculer l'usure du SSD.

Cependant, il est plus complexe de définir l'usure d'un disque dur HDD. Historiquement, plusieurs études utilisaient le nombre de cycles de travail (démarrage-arrêt) pour représenter l'usure d'un disque dur magnétique [78, 101]. Ce paramètre reflète le caractère mécanique du HDD. Le nombre de cycles démarrage-arrêt est un compteur incrémenté à chaque passage vers/depuis le mode en veille. Chaque HDD a un seuil limité de cycles démarrage-arrêt. Une fois ce seuil atteint, il est préférable de changer le HDD afin d'éviter une éventuelle panne. L'obtention de la valeur du compteur fera appel à des outils spéciaux pour chaque constructeur ou à des méthodes complexes basées sur des probabilités statistiques. Par conséquent, un calcul qui s'appuie sur ce paramètre ne sera pas complètement représentatif de l'état d'usure d'un HDD.

De nos jours, les fabricants de disques durs HDD ont remplacé le concept de cycle de travail (démarrage-arrêt) par la quantité totale de données traitées par un disque HDD (c'est à dire la quantité totale de données lues à partir de/écrites sur un disque HDD). Cette information est inscrite dans la fiche technique de chaque périphérique de stockage de type HDD. Nous supposons que le HDD est complètement usé et qu'il doit être changé dès que nous atteignons la charge de travail définie dans sa fiche technique et cela pendant la durée de garantie (e.g. 550TB par an pendant les cinq ans de garantie pour un HDD de 1 TB).

Pour récapituler, la durée de vie d'un périphérique de stockage d_j (wo_{d_j}) dépend de la technologie utilisée pour fabriquer ce dernier. Elle est obtenue en utilisant l'équation 22.

$$wo_{d_j} = \begin{cases} \text{la quantité de données écrite si } d_j = \text{SSD} \\ \text{la quantité de données traitée si } d_j = \text{HDD} \end{cases} \tag{21}$$

Par conséquent, l'usure d'un périphérique de stockage d_j ($wo_w(d_j)$) est obtenue comme le montre l'équation 22, sachant que $total_{write}(d_j)$ représente la quantité de donnée écrites par la charge de travail d'E/S sur le périphérique de stockage d_j et $total_{workload}(d_j)$ représente la quantité de données traitée par

le périphérique de stockage d_j afin d'exécuter la charge de travail qui lui a été transmise.

$$wo_w(d_j) = \begin{cases} total_{write}(d_j) \text{ si } d_j = SSD \\ total_{workload}(d_j) \text{ si } d_j = HDD \end{cases} \tag{22}$$

$total_{write}(d_j)$ représente la quantité de données écrites sur l'ensemble des objets placés dans le disque d_j. Soit $o_{i,u_k,write}$ la quantité de données écrites sur l'objet o_{i,u_k}, avec $o_{i,u_k} \in O_{d_j}$, $total_{write}(d_j)$ est calculé comme le montre l'équation 23.

$$total_{write}(d_j) = \sum_{o_{i,u_k} \in O_{d_j}} (o_{i,u_k,write}) \tag{23}$$

$o_{i,u_k,write}$ est obtenu à partir du nombre de requêtes d'écriture transmises à l'objet o_{i,u_k}, et la taille de la requête d'E/S. Soit io_{size} la taille d'une requête d'E/S. $o_{i,u_k,write}$ est obtenu comme l'indique l'équation 24.

$$o_{i,u_k,write} = \sum_{op \in rw,sw} (io_{size} * req_{op,o_{i,u_k}}) \tag{24}$$

Par ailleurs, la quantité totale de données générées par la charge de travail d'E/S et traitées par le disque d_j ($total_{workload}(d_j)$) correspond à la somme des quantités de données écrites sur/lues à partir de l'ensemble d'objets placés dans ce dernier. $total_{workload}(d_j)$ est décrit par l'équation 25.

$$total_{workload}(d_j) = \sum_{o_{i,u_k} \in O_{d_j}} (o_{i,u_k,workload}) \tag{25}$$

$o_{i,u_k,workload}$ est obtenu à partir du nombre de requêtes transmises à l'objet o_{i,u_k}, et la taille de la requête d'E/S comme exprimé par l'équation 26.

$$o_{i,u_k,workload} = \sum_{op \in OP} (io_{size} * req_{op,o_{i,u_k}}) \tag{26}$$

$req_{op,o_{i,u_k}}$ est obtenu durant la phase de monitoring. C'est une phase préparatoire de caractérisation de la charge de travail.

Pour résumer, nous calculons le coût d'usure d'un système de stockage causé par l'exécution de la charge de travail d'E/S comme le montre l'équation 35.

$$wo_w(d_j) = \begin{cases} \sum\limits_{o_{i,u_k} \in O_{d_j}} \left(\sum\limits_{op \in rw,sw} (io_{size} * req_{op,o_{i,u_k}}) \right) \text{ si } d_j = SSD \\ \sum\limits_{o_{i,u_k} \in O_{d_j}} \left(\sum\limits_{op \in OP} (io_{size} * req_{op,o_{i,u_k}}) \right) \text{ si } d_j = HDD \end{cases} \quad (27)$$

A ce stade, nous avons exprimé en détails tout les composants liés à la modélisation du coût d'exécution de la charge de travail d'E/S transmise au système de stockage. Dans la prochaine section, nous décrivons le coût de pénalité.

4.3 COÛT DE PÉNALITÉ

Pour que le SLA soit un engagement contraignant pour le prestataire de service, il convient de prévoir des pénalités applicables en cas de non-respect du SLA. Ces pénalités peuvent prendre la forme de sanctions financières, de rabais ou d'avoir sur l'utilisation des services du prestataire au profit du client. Les fournisseurs de Cloud sont bien conscients que les pénalités financières pourraient mettre en péril la survie de leur entreprise. L'utilisation des outils et des stratégies modélisant l'aspect de pénalité est primordiale. Dans ce travail, nous proposons un modèle de coût qui prend en considération ce paramètre important pour le Cloud.

Il est nécessaire que le fournisseur Cloud et le client s'accordent sur les métriques déterminant la performance et la qualité de service livrée par le Cloud. Effectivement, les métriques utilisées par les fournisseurs actuels de Cloud restent encore incomplètes et ad-hoc. Les fournisseurs s'intéressent seulement à l'indisponibilité du service due à une panne, ce qui est insuffisant pour mesurer la qualité de service [89]. Dans ce travail, nous utilisons une métrique qui représente réellement la performance du système de stockage, à savoir le débit (IOPS) qui correspond au nombre d'opérations traités par le système de stockage en une seconde. Il faut noter que notre étude peut facilement porter sur d'autres métriques comme le taux de transfert de données (bande passante en Mo/s).

Pour chaque métrique inscrite dans le SLA, les deux parties (client, fournisseur Cloud) doivent définir les exigences et les pénalités associées à celle-ci. Dans notre modèle, nous considérons qu'une remise est appliquée sur la facture globale du client. Le montant déduit (la pénalité) est calculé en utilisant une fonction de calcul de pénalité définie dans le SLA. Nous considérons que

chaque client u_k a une fonction de pénalité (pn_{u_k}) qui lui est propre.

Le dernier point important qui doit être identifié dans le SLA est celui qui consiste à définir des conditions permettant le déclenchement des pénalités. Généralement, un seuil de performance est défini par client, si le seuil n'est pas atteint, le prestataire est sanctionné. Dans notre modèle, nous considérons que chaque client a un seuil de performance nommé $IOPS_{requested}$, si le fournisseur Cloud offre un seuil, nommé $IOPS_{offered}$, inférieur à celui demandé par le client ($IOPS_{requested}$), le fournisseur sera pénalisé. La pénalité a un coût nommé $Cost_{pnl}$. Le montant de cette pénalité sera déduit de la facture du client ($Bill_{original}(u_k)$). Le montant de la facture ($Bill_{final}(u_k)$) dépend du prix unitaire du service cloud. Il s'exprime comme suit :

$$Bill_{final}(u_k) = Bill_{original}(u_k) - Cost_{pnl}(u_k) \tag{28}$$

Le coût de la pénalité $Cost_{pnl}$ dépend de plusieurs paramètres, il est généralement proportionnel au délai et au degré de violation. Nous distinguons trois types de pénalités proposées dans la littérature [49] :

1. **Pénalité fixe :** dans ce cas de figure, $Cost_{pnl}$ est constant et il doit être retranché de la facture à chaque violation du SLA.
2. **Pénalité dépendante du délai :** le montant $Cost_{pnl}$ varie en fonction de la période durant laquelle le SLA n'a pas été respectée.
3. **Pénalité proportionnelle :** dans ce type de calcul, le montant $Cost_{pnl}$ dépend du ratio entre la qualité de service demandée et celle qui est offerte.

Pour notre modèle, nous considérons que la pénalité est proportionnelle au degré de violation et la durée de violation. Le fournisseur du Cloud sera pénalisé en fonction de la gravité de la violation mesurée par le ratio $\frac{IOPS_{offered}}{IOPS_{requested}}$ et à la durée de cette violation.

Nous calculons le montant de pénalité causé par la violation des contraintes du client u_k ($Cost_{pnl}(u_k)$) pour la période T en utilisant la fonction pn_{u_k}. L'équation 29 illustre le calcul de $Cost_{pnl}(u_k)$

$$Cost_{pnl}(u_k) = pn_{u_k}\left(\frac{IOPS_{offered}(u_k)}{IOPS_{requested}(u_k)}, T\right) \tag{29}$$

La fonction de pénalité pn_{u_k} et le $IOPS_{requested}(u_k)$ sont définis dans le SLA du client. $IOPS_{offered}(u_k)$ est obtenu à partir du temps nécessaire au traitement

des requêtes d'E/S émises par le client u_k, et le nombre total de ses requêtes d'E/S comme l'indique l'équation 30. Soit $t_{exe}(u_k)$ le temps nécessaire au traitement des requêtes d'E/S émises par le client u_k et soit $io_{tot}(u_k)$ le nombre total de ces requêtes. $IOPS_{offered}(u_k)$ est calculé comme suit :

$$IOPS_{offered}(u_k) = \frac{io_{tot}(u_k)}{t_{exe}(u_k)} \tag{30}$$

Le nombre total de requêtes d'E/S émises par le client u_k est égal à la somme des requêtes d'E/S transmises à ses objets comme le montre l'équation 31. Soit $req_{o_{i,u_k}}$ le nombre total de requêtes transmises à l'objet o_{i,u_k}. $io_{tot}(u_k)$ est calculé comme suit :

$$io_{tot}(u_k) = \sum_{o_{i,u_k} \in O_{u_k}} req_{o_{i,u_k}} \tag{31}$$

$req_{o_{i,u_k}}$ est calculé à partir du nombre total de requêtes de type $op, op \in \{RR,SR,RW,SW\}$ transmises à l'objet o_{i,u_k} ($req_{op,o_{i,u_k}}$) comme l'indique l'équation 32.

$$req_{o_{i,u_k}} = \sum_{op \in OP} (req_{op,o_{i,u_k}} * T) \tag{32}$$

Nous obtenons $req_{op,o_{i,u_k}}$ à partir de la consolidation des traces générées durant l'étape de monitoring.

Le temps d'exécution des requêtes d'E/S émises par le client u_k ($t_{exe}(u_k)$) est égal au temps d'exécution des requêtes d'E/S transmises à ses objets. Soit $t_{exe}(o_{i,u_k})$ le temps d'exécution des requêtes transmises à l'objet o_{i,u_k}. $t_{exe}(u_k)$ est calculé par l'équation 33.

$$t_{exe}(u_k) = \sum_{o_{i,u_k} \in O_{u_k}} t_{exe}(o_{i,u_k}) \tag{33}$$

Le temps d'exécution des requêtes transmises à l'objet o_{i,u_k} est obtenu à partir du nombre de requêtes transmises à celui-ci et les performances du périphérique de stockage d_j où il est placé comme l'indique l'équation 34.

$$req_{o_{i,u_k}} = \sum_{j=1}^{j} \sum_{o_{i,u_k} \in O_{u_k}} \sum_{op \in OP} \left(x_{d_j} * req_{op,o_{i,u_k}} * t_{op,d_j} \right) \tag{34}$$

avec,

$$x_{d_j} = \begin{cases} 1 \text{ si } pl(o_{i,u_k}) = d_j \\ 0 \text{ sinon} \end{cases} \tag{35}$$

Comme discuté auparavant, les performances d'un périphérique de stockage d_j sont mesurées dans une phase préparatoire nommée *la phase de calibration*.

Nous avons exprimé le coût de pénalité engendré par un placement pour une période T. Dans la suite du chapitre, nous détaillons le coût de migration.

4.4 COÛT DE MIGRATION

Le coût de migration correspond au coût d'exécution du placement proposé. Il faut noter que pour achever un placement donné à partir d'un placement initial, une opération de migration doit être appliquée sur certains objets. Cette opération consiste à déplacer un ensemble d'objets d'un périphérique *source* à un autre périphérique *destination*. L'opération de migration engendre donc une charge de travail d'E/S supplémentaire. L'exécution de cette charge de travail consomme de l'énergie et affecte l'endurance du système de stockage.
L'équation 7 présentée précédemment illustre la méthode de calcul du coût d'exécution d'une charge de travail. Ainsi, le coût de migration est obtenu à partir du coût énergétique et du coût d'usure causés par l'exécution de la charge de travail de migration. Soit d_{src} (respectivement d_{dst}) le disque source (respectivement destination). Le coût de migration est donné comme suit :

$$Cost_{mgr,t_m} = Cost_{edr,d,T}(d_{dst}) + Cost_{edr,d,T}(d_{src}) + Cost_{erg,T}(d_{dst}) + Cost_{erg,T}(d_{src}) \tag{36}$$

Afin de simplifier la modélisation du coût de migration, nous considérons dans un premier temps que la migration s'exécute en arrière plan et ne consomme que la ressource d'E/S disponible afin de ne pas impacter la charge de travail des clients. Par conséquent, une migration ne peut pas être perçue comme une dégradation des performances, cela se traduit par une pénalité de migration nulle.

Nous supposons aussi que les requêtes d'E/S transmises à l'objet o_{i,u_k} pendant sa migration sont traitées par l'un des disques source (d_{src}) ou destination (d_{dst}) afin d'éviter l'indisponibilité du service. Par conséquent, le coût de pénalité causé par l'indisponibilité de service est nul. Dans ces conditions, le coût de migration est uniquement représenté par le coût d'exécution de la charge de travail d'E/S liée à l'opération de migration.

4.4.1 *Coût d'énergie causé par la migration*

Soit $Cost_{mgr,erg,t_m}$ le coût énergétique consommé par le système de stockage pour exécuter la charge de travail d'E/S liée à l'opération de migration et soit t_m le temps de migration. $Cost_{mgr,erg,t_m}$ obtenu à partir du coût énergétique du périphérique de stockage source (d_{src}) et le coût énergétique du périphérique de stockage destination (d_{dst}) comme le montre l'équation 37.

$$Cost_{mgr,erg,t_m} = Cost_{erg,T}(d_{dst}) + Cost_{erg,T}(d_{src}) \tag{37}$$

Dans la section 4.2.2, nous avons présenté la méthode de calcul du coût énergétique d'un périphérique de stockage d_j. La charge de travail et la période d'évaluation sont considérées comme des paramètres entrés pour la méthode de calcul. Par conséquent, il nous suffit juste de définir la charge de travail et le temps de migration t_m pour calculer le coût d'énergie induit par l'opération de migration. D'une manière générale, la charge de travail d'une opération de migration est définie comme suit :

1. La lecture de données à partir du périphérique source d_{src}.
2. L'écriture de données sur le périphérique destination d_{dst}.

Nous considérons que les deux opérations se font d'une manière séquentielle. La lecture (respectivement écriture) se fait d'une manière séquentielle à partir du disque source d_{src} (respectivement disque destination d_{dst}). Le nombre d'opérations est calculé à partir de la taille d'objet o_{i,u_k} et la taille du bloc d'E/S. Soit io_{size} la taille d'un bloc d'E/S et soit x le nombre d'opérations de lecture/écriture générées par la charge de travail liée à l'opération de migration. x est calculé comme l'indique l'équation 38.

$$x = \lceil \frac{S_{o_{i,u_k}}}{io_{size}} \rceil \tag{38}$$

Le temps de migration est la valeur maximale du temps nécessaire à la lecture de données et du temps nécessaire à l'écriture de ces dernières. Soit $t_{m,read}$ (respectivement $t_{m,write}$) le temps nécessaire pour la lecture (respectivement l'écriture) de données. L'équation 39 présente le temps de migration t_m.

$$t_m = max(t_{m,read}, t_{m,write}) \tag{39}$$

Le temps nécessaire à la lecture des données est obtenu à partir du nombre d'opération d'E/S ,x, et les performances du disque source d_{src} en lecture comme le montre l'équation 40.

$$t_{m,read} = x * t_{sr,d_{src}} \tag{40}$$

Le temps nécessaire à l'écriture de données est obtenu à partir du nombre d'opération d'E/S ,x, et les performances du disque destination d_{dst} en écriture comme le montre l'équation 41.

$$t_{m,write} = x * t_{sw,d_{dst}} \tag{41}$$

La prochaine section présente la seconde partie du coût de migration. Elle décrit la méthode de calcul du coût d'usure du système de stockage causé par l'opération migration.

4.4.2 *Coût d'endurance causé par la migration*

Le coût d'usure du système de stockage induit par l'opération de migration est obtenu à partir du coût d'usure du périphérique de stockage source (d_{src}) et du coût d'usure du périphérique de stockage destination (d_{dst}) comme le montre l'équation 42.

$$Cost_{mgr,edr,t_m} = Cost_{edr,d,T}(d_{dst}) + Cost_{edr,d,T}(d_{src}) \tag{42}$$

Le coût d'usure d'un périphérique de stockage causé par une charge de travail est donné dans l'équation 19. Comme la charge de travail est connue, le coût d'usure est calculé comme expliqué dans la section 4.2.2.2.

4.5 ÉVALUATION

Nous présentons dans cette dernière section les expérimentations que nous avons mené afin d'évaluer l'exactitude de notre modèle de coût. La validation est faite en deux étapes. La première étape consiste à présenter un cas d'utilisation réel de notre modèle de coût. Nous allons monter comment obtenir les paramètres et les valeurs d'entrée. Dans la seconde partie, nous allons montrer la pertinence de chaque composant (sous-coût) dans le modèle proposé et comparer nos résultats avec les travaux connexes. Nous avons utilisé TPC-H [19] et TPC-C [20] comme benchmarks pour la validation de notre modèle de coût.

4.5.1 *Plateforme expérimentale*

L'évaluation du modèle de coût proposé est menée sur une plateforme réelle. La plateforme est virtualisée comme le montre la figure 4.3. Nos machines virtuelles sont installées sur un hyperviseur VMware ESXi 5.10. Nous avons utilisé un système de stockage qui se compose d'un disque dur HDD et d'un autre disque

SSD.
Chaque machine virtuelle est dotée d'un traceur qui intercepte les requêtes d'E/S et qui consolide l'information (les traces collectées) au niveau des objets afin d'extraire le motif d'accès de chaque objet dans le système. Le traceur utilisé est développé sous la forme d'un module de noyau Linux. Il capture les requêtes d'E/S en-dessous du cache SGBD et celui du système d'exploitation afin d'éviter l'impact de la mise en cache (voir la figure 4.3).

Chaque machine virtuelle est configurée avec vCPU de 8 coeurs et 8Go de RAM. L'hyperviseur ESX s'exécute sur un serveur HP G9 DL380 doté d'un processeur Intel Xeon 2,4 GHz et de 12 Go de RAM. Nous avons utilisé la distribution GNU Debian 7.5 (noyau 3.19.5) de Linux comme système d'exploitation et PostgreSQL 9.3.5 comme SGBD. Le tableau 2 décrit les caractéristiques des périphériques de stockage utilisés dans cette évaluation.

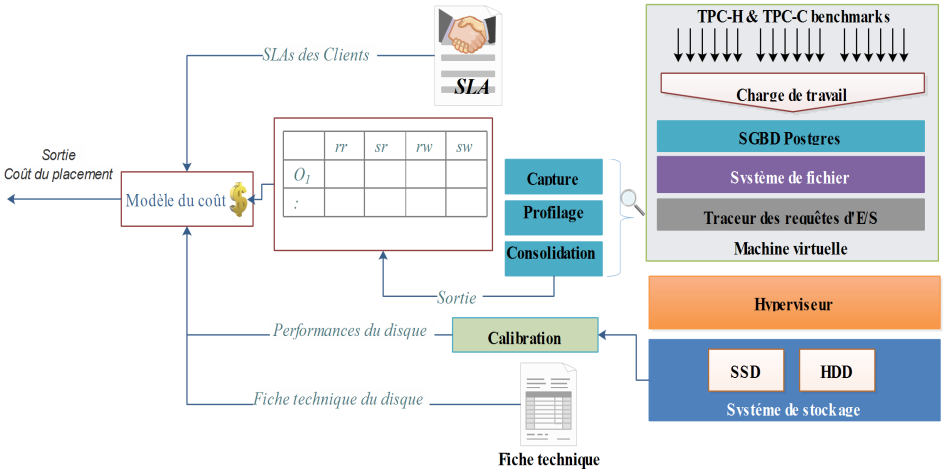

FIGURE 4.3 : Plateforme d'expérimentation

4.5.2 *Cas d'utilisation du modèle de coût*

Cette partie présente un cas d'utilisation du modèle de coût proposé. Nous montrons d'abord comment obtenir les valeurs d'entrée, ensuite nous détaillons la phase de calibration requise pour obtenir les valeurs d'entrée spécifiques.

Plusieurs paramètres d'entrée sont utilisés pour faire appel à notre modèle de coût. Nous pouvons les organiser, selon la source d'obtention, en quatre grandes catégories :

1. La fiche technique du périphérique de stockage : elle contient p_{d_j}, c_{d_j}, c_{wo_j}, $t_{atv_{dj}}$, $P_{idl_{dj}}$ comme l'indique le Tableau 4.2.

2. Le traceur : comme évoqué précédemment, nous avons développé un traceur qui capture les requêtes d'E/S transmises aux objets et compte le nombre de requêtes de type op ∈ OP ($req_{op,o_{i,u_k}}$) transmis à chaque objet o_{i,u_k}.

3. La calibration : nous conduisons une série d'expérimentations afin d'obtenir les performances des périphériques de stockage d_j (t_{op,d_j}, $iops_{op,d_j}$) et cela pour chaque type d'opération op ∈ OP. Cette phase sera présentée ci après.

4. L'administrateur du Cloud : ce dernier est le seul habilité à accéder aux méta-données du système pour obtenir des informations sur les clients et leurs objets tels que (u_k, $iops_{sla,u_k}$, pn_{u_k}, O_{u_k}, o_{i,u_k}, $S_{o_{i,u_k}}$).

Calibration : La calibration consiste à mesurer les performances d'un périphérique de stockage d_j telles que le temps de réponse (t_{op,d_j}) et le débit ($iops_{op,d_j}$). Pour ce faire, nous avons adopté la même méthodologie décrite dans [97] pour la calibration. Nous soulignons que nos expérimentations sont menées en utilisant la taille de bloc d'E/S par défaut de la base de données. Les expérimentations sont menées avec un taux d'occupation de moins de 50% pour les deux périphériques de stockage comme dans [71]. Le tableau 4.3 résume nos résultats.

Nous observons dans le tableau 4.3 que le temps de réponse d'un disque SSD est stable et que les valeurs mesurées sont très proches de la moyenne (écart-type faible). La valeur de t_{op,d_j} inscrite dans le tableau est calculée à partir du temps de réponse moyen des mesures prises.

Cependant, nous observons que le temps de réponse d'un disque dur HDD est stable pour les accès séquentiels en raison de l'absence du mouvement mécanique de la tête du lecture/écriture. Par contre, nous avons noté une forte instabilité pour les accès aléatoires et cela est dû à l'impact de la distance de recherche (en anglais *seek time*).
Afin de prédire les performances d'un disque HDD pour les opérations d'accès aléatoire, nous avons mené une étude dont le but est d'explorer l'impact de la distance de recherche sur le temps de réponse. Nous avons observé que le temps de réponse varie d'une manière linéaire avec la distance de recherche. Cette corrélation est modélisée par une méthode de régression linéaire comme l'indique l'équation 43. L'équation présente un taux d'erreur entre ±3, 11 % et ±11 %.

$$\varphi(seek_{d_j,t}) = t_{r(r/w),HDD} = \begin{cases} 0.98 \text{ ms si } seek_{HDD,t} < 0.5GB \\ 0.058 * seek_{d_j,t} + 5.56 \text{ si } seek_{HDD,t} \in [0.5, 150]GB \\ 15 \text{ ms si } seek_{HDD,t} > 150GB \end{cases}$$

Table 4.2 : Spécifications du périphérique de stockage

	HDD	SSD
modèle	ST91000640SS	850 PRO
prix ($)	230	200
capacité (Go)	1000	120
Garantie (ans)	5	5
Performance	seek :8.5ms - read/write :9.5ms	SR :540 MB/s, SW :520 MB/s RR :10 000 IOPS, RW :40 000 IOPS
Latence (ms)	4.16	-
puissance idle (w)	3.27	3.7
endurance	550TB/ans	75TBW (téraoctets écrits)

Table 4.3 : Temps de réponse d'un périphérique de stockage

	HDD[min - max - avg - atdev]	SSD[min - max - avg - atdev]
SR(ms)	[0.6 - 1 - 0.81 - 0.014]	[0.031 - 0.031 - 0.031 - 0.0002]
RR(ms)	[5.29 - 15.03 - 8.42 - 2.49]	[0.32 - 0.39 - 0.35 - 0.024]
SW(ms)	[0.6 - 1 - 0.8 - 0.012]	[0.22 - 0.26 - 0.23 - 0.01]
RW(ms)	[5.08 - 14.68 - 8.31 - 2.37]	[0.53 - 0.57 - 0.55 - 0.013]

(43)

4.5.3 *Évaluation du modèle de coût*

4.5.3.1 *Benchemark*

Le modèle de coût est évalué avec plusieurs bases de données construites à l'aide des deux benchmarks TPC-H et TPC-C. Au total, six bases de données ont été crées dont trois à l'aide du benchemark TPC-C et trois autres à l'aide du benchemark TPC-H. Les bases de données TPC-H ont été générées à l'aide de DBGen avec un facteur d'échelle (en anglais *scale factor*) respectivement de 30, 100 et 300. Les charges de travail ont été générées de façon aléatoire à partir du 22 modèles de requêtes TPC-H comme dans [132]. Les bases de données TPC-C ont été générées en utilisant un facteur d'échelle de 12, 350 et 800 d'entrepôts respectivement. Nous avons varié le taux d'arrivée des requêtes pour le benchemark TPC-H, le nombre de terminaux/entrepôt pour le benchemark TPC-C et la taille des bases de données pour bien simuler un environnement réel.

Nous avons utilisé notre outil pour tracer les requêtes d'E/S et extraire le profil (motif) d'accès des E/S des objets. Nous avons estimé le temps de réponse du disque dur pour l'accès aléatoire en utilisant l'équation 43. Le tableau 4.4 ci-dessous montre les coûts mensuels et le profil d'accès des différentes bases de données pendant une heure d'exécution.

4.5.3.2 *Paramétrage du modèle de coût*

Dans la présente évaluation, le coût d'amortissement est calculé en répartissant le coût d'achat du périphérique de stockage sur une période de 5 ans comme dans [39, 40, 78]. Cette période représente la durée de garantie. Le coût énergétique est calculé en utilisant un coût de 0,1$ par kWh comme dans [132]. Bien évidemment, le coût de migration dépend de la fréquence de migration et de la quantité de données transférées. Nous supposons que la migration est effectuée à une fréquence de deux fois par jour comme dans [130]. Le montant de pénalité est calculé sur la base de 30% de la facture du client (total des frais payés par le client) comme dans Amazon Cloud. Nous soulignons que les résultats présentés sont basés sur le coût d'achat réel des périphériques de stockage.

4.5.3.3 *Discussion*

Le tableau 4.4 indique que tous les coûts évalués ($Cost_{occ,T}$, $Cost_{edr,T}$, $Cost_{erg,T}$, $Cost_{stg,T}$, $Cost_{pnt,T}$ et $Cost_{mgr,T}$) sont du même ordre de grandeur et que les coûts dominants varient en fonction de la taille de la base de données, de la charge de travail et du périphérique de stockage. Nous observons que le coût d'occupation est dominant pour les bases de données volumineuses rarement consultées (voir DB4, DB6). Nous prenons à titre d'exemple les bases de données OLAP rarement consultées. Cependant, le coût d'endurance devient dominant et le plus important pour les petites bases de données qui ont une charge de travail d'E/S intensive (voir DB2, DB3). Nous citons à titre d'exemple les bases de données OLTP. Nous avons observé que le coût d'énergie représente 5 à 28% du coût global pour les bases de données placées sur un disque HDD (voir DB1 et DB5). Nous remarquons que le coût d'endurance peut atteindre jusqu'à 90% du coût global pour les bases de données stockées dans un disque SSD et qui font beaucoup d'opérations d'écriture. De plus, l'administrateur Cloud doit être attentif au coût de migration qui peut augmenter d'une façon drastique lorsque une grande quantité de données est migrée (voir DB4).

Nos expérimentations montrent que l'impact de l'ordre de déplacement (migration) des objets représente entre 0,5 et 8,25% du coût global. D'un autre côté, nous voyons que le coût des pénalités est très important et qu'il peut atteindre

TABLE 4.4 : Estimation des coûts pour un mois en ($)

	TPC-C			TPC-H		
	DB1	DB2	DB3	DB4	DB5	DB6
taille(Go)	32	1.2	60	147	34	281
disque	HDD	SSD	SSD	HDD	HDD	HDD
RR(op/h)	136800	540000	601200	10800	32400	1080
SR(op/h)	46800	68400	104400	216000	216000	8280
RW(op/h)	108000	468000	543600	3600	7200	360
SW(op/h)	32400	82800	147600	18000	10800	1800
$Cost_{occ,T}$	0.1104	0,0308	1.5411	0.507	0.117	1.314
$Cost_{edr,T}$	0.1454	3.9396	4.9438	0.111	0.120	0.005
$Cost_{erg,T}$	0.3559	0.0349	0.0402	0.242	0.253	0.236
$Cost_{stg,T}$	0.5013	3.9746	4.9841	0.749	0.372	1.550
$Cost_{pnt,T}$	0.96	0.036	1.8	4.410	1.020	0.000
$Cost_{mgr,T}$	0.0854	0.0032	0.1602	0.393	0.091	0.000
le coût Total	1.5468	4.0138	9.2843	5.552	1.483	1.550
Travaux connexes						
[78]	0.5013	3.9746	4.9841	0.3537	0.3723	0.2412
[132]	0.4663	0.0657	1.5813	0.7494	0.3701	1.5504
[97]	1.5468	4.0138	6.9443	5.1562	1.4831	0.2412
[43, 54, 71, 72]	0.3559	0.0349	0.0402	0.242	0.253	0.236

80% du coût global. C'est la raison pour laquelle le coût de pénalité devient l'une des plus grandes préoccupations du CSP [44, 109, 127].

Nous constatons que toutes les parties (sous-coûts) constituant notre modèle de coût peuvent avoir un impact significatif sur le coût global et ceci selon la configuration globale. Par conséquent, Il est nécessaire de prendre en compte toutes ces parties afin de faire une estimation correcte et précise du coût de placement. Il faut noter que l'évaluation mentionnée ci-dessus est conduite à une échelle beaucoup plus petite que celle des vrais centres de données.

4.5.3.4 *Comparaison avec les travaux connexes*

La deuxième partie du tableau 4.4 illustre une comparaison entre le modèle de coût proposé et les travaux connexes proposés dans la littérature. Nous observons que dans [78], les auteurs proposent un modèle de coût qui évalue avec précision le coût d'exploitation en combinant le coût d'occupation, le coût énergétique et l'endurance. Cependant, les auteurs ignorent le coût de pénalité qui représente un paramètre très impactant pour le Cloud. L'utilisation de ce modèle de coût dans un contexte Cloud conduit à une mauvaise estimation et pourra générer un taux d'erreur qui atteint les 80% dans le pire des cas. Dans [97], les auteurs proposent un modèle incluant les coûts liés à la pénalité et à la migration. Cependant, ils ignorent le coût d'occupation et l'impact de l'ordre de la migration. Le modèle de coût proposé entraîne un taux d'erreur (comparé au nôtre) entre 7% et 80%, comme le montre le tableau 4.4. Notons que le taux d'erreur augmente linéairement avec la taille de la base de données. De plus, ce modèle de coût ignore l'impact de l'ordre de migration qui engendre un taux d'erreur non négligeable.

Afin de montrer l'impact de l'ordre de migration sur le coût global, nous avons réalisé quatre expérimentations en utilisant le benchmark TPC-C. Dans chaque expérimentation nous partirons du même placement initial et migrons vers le même placement cible mais avec des séquences de migration différentes. La figure 4.4 représente le coût de placement pour la période d'évaluation. Le coût de placement global pour la période d'évaluation est représenté par la surface délimitée par la représentation graphique. Nous remarquons que la superficie varie d'une expérimentation à l'autre. Nous avons constaté que l'ordre de migration impacte le coût global à hauteur de 8,25%.

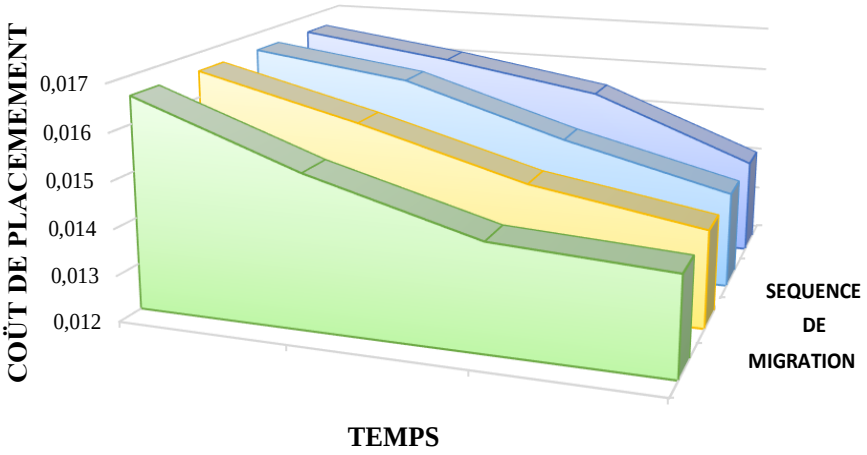

FIGURE 4.4 : Impact de l'ordre de migration

4.6 CONCLUSION

Dans ce chapitre, nous avons introduit notre modèle de coût pour évaluer le coût de placement des objets de DBaaS. Le modèle proposé complète les modèles existants en prenant en compte 1) Les contraintes SLA des clients, 2) Le coût des pénalités, 3) Le coût de migration et 4) L'impact de l'ordre de migration des objets sur l'estimation du coût global.

Le modèle de coût proposé pourra être utilisé dans plusieurs stratégies à savoir :

1. Une stratégie de tarification afin d'évaluer précisément le coût global du système de stockage ;
2. Une stratégie pour trouver le placement optimal des objets ;
3. Une stratégie pour une allocation efficace de la ressource d'E/S ;

Il faut noter que le modèle de coût proposé considère la diversité des périphériques de stockage. Nous avons utilisé un système de stockage hybride HDD-SSD très simple comme un cas d'étude. Cependant, le modèle peut être généralisé pour des cas beaucoup plus complexes. Par exemple, l'utilisation d'autres classes de stockage comme les NVM. Dans nos expérimentations, nous avons démontré la facilité d'utilisation et la pertinence de notre modèle de coût.

Après avoir présenté le modèle de coût, nous décrivons, dans le chapitre suivant, notre approche de placement de données.

Chapitre 5

STRATÉGIE DE PLACEMENT DES OBJETS DE DBAAS

Nous proposons dans ce chapitre deux approches de placement des objets de DBaaS dans un système de stockage hybride. Ces approches emploient le modèle de coût présenté dans le chapitre précédent (chapitre 4). Leur objectif est de trouver un placement optimisé qui satisfait les contraintes des clients (SLAs) et minimise le coût global pour le Cloud.

Nous avons utilisé une modélisation sous forme d'un problème d'optimisation, dont l'objectif principal est de minimiser le coût de placement global pour le fournisseur de service Cloud. Le problème d'optimisation est soulevé sous un ensemble de contraintes. Ces dernières peuvent être divisées en trois catégories principales :

1. Les contraintes de performance liées aux SLA des client.
2. Les contraintes relatives aux objets.
3. Les contraintes liées au système de stockage

Nous proposons deux stratégies pour résoudre ce problème d'optimisation. La première nommé « G-COPS » est basée sur une méta-heuristique, précisément sur un algorithme génétique (AG), tandis que la seconde nommée « H-COPS » est basée sur un algorithme heuristique incrémental.

L'idée de H-COPS consiste à commencer par un plan de placement initial qui sera amélioré par la suite en deux phases. Le plan de placement initial est construit en plaçant chaque objet dans la classe de stockage ayant le coût de stockage le plus bas. La deuxième phase, nommée *étude de faisabilité*, reprend le plan de placement initial pour le rendre faisable. La troisième phase, nommée *optimisation*, améliore le plan résultant de la deuxième phase et cela par élimination les pénalités.

Plusieurs travaux ont proposé des approches de placement de données dans un système de stockage hybride. Cependant, ces approches ne sont pas adaptées pour le contexte Cloud. Les solutions proposées sont orientées performance alors que dans un contexte Cloud la solution doit être orientée coût.

Nous avons évalué l'ensemble de ces approches sur une plateforme réelle avec un système de stockage hybride qui combine deux classes de stockage HDD et SSD. Deux benchmarks ont été utilisés. Le benchmark TPC-C [20] pour simuler une base de données OLTP et le benchmark TPC-H [19] pour simuler un environnement OLAP. Nous avons comparé nos approches avec deux travaux connexes [132] et [33] proposés dans la littérature. Les résultats obtenus sont très prometteurs. Les approches proposées réduisent le coût de placement à plus de 40%.

Nos approches de placement ont été publiées dans *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 2017, pp. 659-664* [27] et dans la revue *Future Generation Computer Systems*, Volume 93, Avril 2019, Pages 176-187.

Ce chapitre est organisé comme suit. Premièrement, nous définissons quelques principes utilisés dans les différentes sections du chapitre. Ensuite, nous présentons un aperçu du placement des objets. La troisième partie détaille nos contributions. La quatrième partie présente l'évaluation de nos contributions. La cinquième partie conclut le chapitre.

Sommaire

5.1 PLACEMENT DES OBJETS DANS LE CLOUD

Avant de décrire nos approches d'optimisation, nous définissons les principaux concepts utilisés et nous présentons les hypothèses sur lesquelles nous nous sommes appuyés dans ce travail.

5.1.1 *Hypothèses*

Nous présentons ci-dessous les hypothèses ayant servies de base pour notre travail.

1. La base de données est relationnelle [26, 45, 63, 122] ;
2. Le service DBaaS est basé sur une architecture multi-locataires où plusieurs locataires partagent le même SGBD et les mêmes ressources physiques ;
3. Le client exprime une contrainte de performance dite *soft* désignant la qualité de service souhaitée par celui-ci et une contrainte de performance dite *hard* définissant la dégradation de qualité de service tolérable par ce dernier ;
4. Le placement et la migration s'effectuent à la granularité d'un objet ;

Remarque : Les hypothèses du chapitre précédent (chapitre 4) restent valides pour ce chapitre.

5.1.2 *Définitions*

Multi-locataire : désigne un principe d'architecture logicielle permettant à un logiciel de servir plusieurs organisations clientes (en anglais *tenant*, en français locataire) à partir d'une seule installation [81, 98, 131].

5.1.3 *Problème de placement des données*

Dans un environnement de Cloud, la charge de travail des clients varie dans le temps. Par conséquent, l'administrateur du service Cloud doit réviser (revoir) le placement des objets pour faire face à cette fluctuation et aux nouvelles exigences en termes de performance. Le nouveau placement doit répondre aux contraintes des clients listées dans les SLAs et doit minimiser le coût de placement global pour le fournisseur Cloud. En outre, l'administrateur Cloud doit trouver un placement adéquat dans un temps raisonnable.

De ce fait, l'administrateur doit faire appel à une stratégie de placement adaptée pour un contexte de Cloud. Cette stratégie doit prendre en considération les contraintes des clients en termes de performance et de sévérité des clients. La problématique peut être défini comme suit : *comment avoir un placement optimal qui puisse répondre aux exigences des clients et qui minimise le coût de placement pour le fournisseur Cloud et cela dans un temps raisonnable.*

5.1.4 *Travaux connexes*

Plusieurs travaux traitent le problème de placement des données dans un système de stockage hybride. Ces travaux s'appuient généralement sur le profil d'accès aux données et sur les caractéristiques des périphériques de stockage afin de prendre une décision de placement rapide pour la période suivante [33, 71, 72, 132].

Ce problème est considéré comme un problème NP-difficile [54]. Il est généralement modélisé sous forme d'un problème d'optimisation [54]. Par conséquent, plusieurs heuristiques ont été proposées afin de trouver une solution (proche à l') optimale. Ces heuristiques doivent s'exécuter dans un temps raisonnable et d'agir dans des délais courts afin d'être utilisées en ligne.

Les stratégies [33, 54, 71, 72, 132] proposées dans la littérature sont orientées performance. Leur but principal est de trouver un placement qui améliore les performances du système de stockage. Cependant, une telle approche dans le Cloud doit être orientée coût. L'administrateur Cloud cherche à trouver un placement qui satisfait les contraintes des clients et qui minimise le coût d'exploitation global pour le Cloud.

En outre, la stratégie de placement doit prendre en considération les caractéristiques du Cloud (i.e. le SLA, la pénalité, l'aspect multi-locataire, la sévérité des clients). Il faut noter que dans un environnement de Cloud, le système de stockage est partagé entre plusieurs clients qui ont différentes sévérités. La sévérité du client est représentée par le montant de la pénalité réclamé par celui-ci en cas de non-respect du SLA.

Dans ce travail de thèse, nous proposons deux stratégies de placement adaptées pour un contexte Cloud. Nos approches sont orientées coût et elles prennent en considération les caractéristiques du Cloud. Nous formulons le problème de placement sous forme d'un problème d'optimisation. L'optimisation est menée sous un ensemble de contraintes qui seront présentées dans les sections sui-

vantes.

5.2 FORMULATION DU PROBLÈME DE PLACEMENT DANS UN CLOUD

Avant de modéliser le problème de placement, nous présentons un aperçu général sur l'architecture du système utilisé.

5.2.1 *Aperçu général du système*

L'architecture du système considérée dans cette thèse est illustrée dans la figure 5.1. Nous supposons que le fournisseur de Cloud utilise un SGBD qui supporte l'architecture multi-locataires pour fournir le service de base de données. Par conséquent, une seule instance SGBD est utilisée pour satisfaire les requêtes des clients.

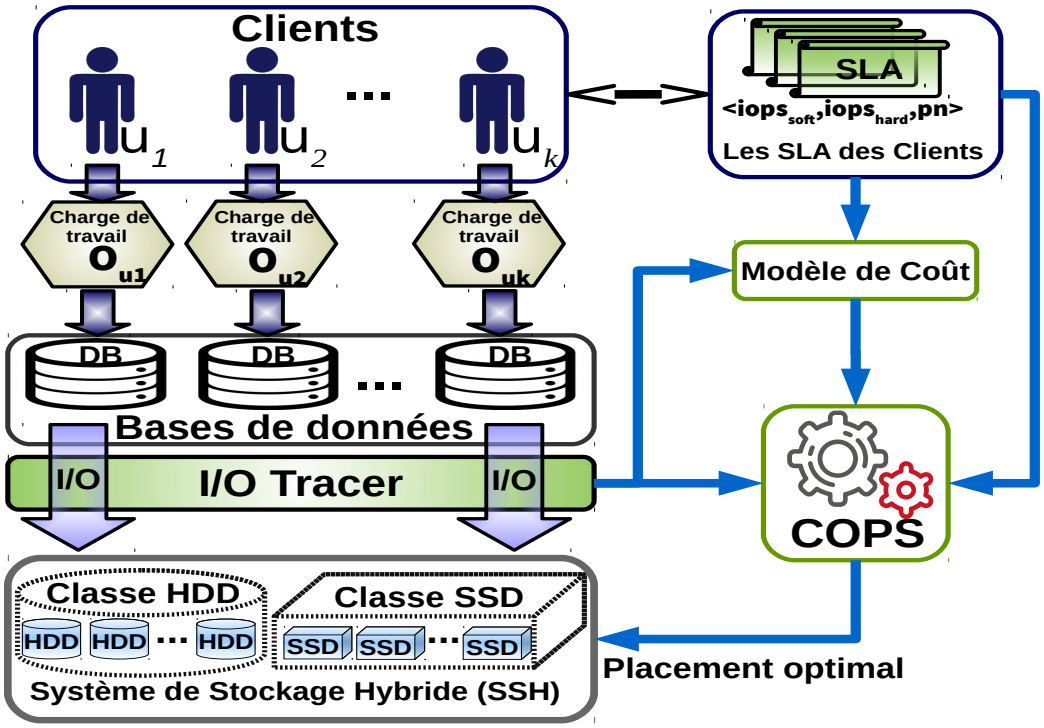

FIGURE 5.1 : Aperçu global du système

Avant de détailler nos approches, nous présentons ci-dessous les différentes notions faisant l'objet de notre étude.

Les stratégies proposées dans cette thèse supportent les deux premières architectures multi-locataires citées ci-dessus. Cependant pour que la troisième architecture soit applicable il faut :

1. Segmenter les tables partagées selon l'ID locataire. C'est la méthode la plus simple.
2. Changer le niveau de granularité de placement en passant à une granularité plus fine au niveau de l'enregistrement.

Il faut noter que nous avons exclu dans cette classification le cas classique de la virtualisation où chaque locataire détient sa propre machine virtuelle, toutefois, nos stratégies restent valides pour cette dernière architecture.

5.2.1.1 *Client*

Rappelons que nous désignons par un client u_k toute entité hébergeant sa base de données dans le Cloud. Chaque client u_k a une base de données constituée d'un ensemble d'objets $O_{u_k} = \{o_{1,u_k}, ..., o_{i,u_k}\}$. Un objet représente toute entité logique d'une base de données (par exemple un index ou une table pour une base de données relationnelle). $s_{o_{i,u_k}}$ est la taille de l'objet o_{i,u_k}.

Nous considérons que le placement s'effectue à la granularité d'un objet de base de données. L'utilisation d'une granularité plus petite (par exemple un bloc) augmenterait le volume de métadonnées à traiter [56–60, 79, 114], le temps de traitement et la complexité. D'autre part, l'utilisation d'une granularité de base de données entière comme dans [97] n'est pas optimale pour un contexte DBaaS car la taille de la base de données devient importante, par conséquent des différents motifs d'accès seront appliqués aux différentes parties de celle-ci. Cela rend la décision de placement non optimale.

5.2.1.2 *SLA*

Dans un contexte DBaaS, chaque client dispose d'un SLA qui définit et quantifie les ressources allouées et la qualité de service souhaitée. Comme nous nous intéressons au système de stockage, nous résumons le SLA dans la métrique de débit d'E/S (IOPS). Il faut noter que notre travail peut être étendu pour d'autres métriques (i.e. le taux de transfert de données).

Le fournisseur de service Cloud doit satisfaire le SLA convenu. Dans le cas contraire, une pénalité lui sera appliquée. Nous considérons que chaque client u_k requiert une valeur d'IOPS appelé $iops_{soft,u_k}$ (SLA soft) et il tolère une dégradation de performance peuvant aller jusqu'à $iops_{hard,u_k}$ (SLA hard) comme

dans [89].

Soit $iops_{offert,u_k}$ les IOPS fournis par le Cloud au client u_k. Dans le cas où le $iops_{offert,u_k}$ se trouve dans l'intervalle $]iops_{soft,u_k}, iops_{hard,u_k}]$, une pénalité p est appliquée au fournisseur de service Cloud. Le montant de la pénalité (p) est calculé en utilisant une fonction de pénalité pn_{u_k} qui dépend de $iops_{offert,u_k}$. Il faut noter que $iops_{soft,u_k}$ est considéré comme étant l'objectif à satisfaire pour le fournisseur de service cloud (*soft contrainte*), tandis que le $iops_{hard,u_k}$ est une *contrainte dure* qui doit être absolument vérifié.

5.2.1.3 *Système de stockage*

Nous supposons que le système de stockage combine deux classes de stockage, SSD et HDD. Pour simplifier, nous considérons que les périphériques de stockage d'une classe donnée sont homogènes en termes de performances. Il faut noter que notre étude peut être facilement étendue pour considérer d'autres classes de stockage.

Chaque classe de stockage d_j avec $j = \{HDD, SSD\}$ a une capacité c_{d_j} et un débit d'E/S $iops_{op,d_j}$ pour les opérations d'E/S de type op. Nous présentons les différents types d'opérations d'E/S ci-dessous. Les objets du client peuvent être répartis entre différentes classes de stockage. Soit O_{d_j} l'ensemble des objets stockés dans la classe de stockage d_j.

Dans ce travail, la décision de placement se limite dans la classe de stockage ou l'objet sera stocké et non pas le disque physique ou l'objet sera placé. Nous supposons que l'administrateur de Cloud utilise une technique de virtualisation qui agit comme une couche d'abstraction des périphériques de stockage.

5.2.1.4 *Le placement*

La charge de travail d'E/S des clients peut varier avec le temps. Par conséquent, l'administrateur du service Cloud doit mettre à jour le placement en cherchant un nouveau placement qui satisfait les nouvelles exigences des clients en termes d'IOPS et minimise le coût d'exploitation global (coût de stockage, coût de la mise en place, etc.) en même temps.

Un placement est vu comme une projection de l'ensemble d'objets sur l'ensemble de classes de stockage. La fonction de projection $pl(o_{i,u_k})$ indique la classe de stockage d_j qui stocke l'objet o_{i,u_k}.

5.2.1.5 *Le monitoring*

Afin d'obtenir un placement pertinent pour la période de temps suivante, l'administrateur du service Cloud doit prédire le motif d'accès d'E/S de chaque objet. Dans ce travail, nous supposons que la charge de travail d'E/S de la période de temps suivante $(T+1)$ est la même que celle de la précédente période T comme dans [54].

Le motif d'accès d'E/S d'un objet est défini par le type d'accès (lecture / écriture) et la nature d'accès (aléatoire / séquentielle). Par conséquent, nous distinguons quatre types d'opérations d'E/S (op) : lecture aléatoire (RR), lecture séquentielle (SR), écriture aléatoire (RW) et écriture séquentielle (SW).

Dans ce travail, nous effectuons une surveillance continue des requêtes d'E/S afin d'extraire le profil d'accès des différents objets. La phase de surveillance trace pour chaque objet o_{i,u_k}, le nombre de requêtes d'E/S de type op transmises à ce dernier dans la période T ($req_{op,o_{i,u_k}}$).

5.2.2 *Modélisation du problème de placement*

Le placement des objets dans un système de stockage hybride est généralement modélisé comme un problème d'optimisation similaire à un problème de bin packing [54]. En faisant une analogie avec un problème de bin packing, les périphériques de stockage représentent les boites avec une capacité limitée en termes de stockage et les objets représentent les outils avec une taille définie en termes de demande de stockage. La fonction coût est représentée par l'ensemble des coûts engendrant le placement d'un objet dans un périphérique de stockage (voir chapitre 4).

Nous formalisons le problème de placement des objets de DBaaS comme suit :

$$
\begin{cases}
Minimize(Cost_{pl,T}) & (44a)\\
\forall\, d_j,\ (\sum_{o_{i,u_k} \in O_{d_j}} s_{o_{i,u_k}}) \leqslant c_{d_j} & (44b)\\
\forall d_j,\ (\sum_{op \in OP} (\frac{\sum_{o_{i,u_k} \in O_{d_j}} req_{op,o_{i,u_k}}}{iops_{op,d_j}})) \leqslant 1 & (44c)\\
\forall u_k \in U,\ iops_{offered,u_k} \geqslant iops_{hard,u_k} & (44d)\\
\forall o_{i,u_k} \in O_{u_k}, \exists!\ d_j, pl(o_{i,u_k}) = d_j & (44e)
\end{cases}
$$

Le but est de minimiser le coût global exprimé par la fonction objectif (44 a). Cette fonction calcule le coût de placement de l'ensemble des objets sur l'ensemble des périphériques de stockage en prenant en compte le coût d'exécution des charges de travail d'E/S des différents clients, le coût de pénalité et le coût de migration. Nous utilisons le modèle de coût présenté dans le chapitre 4 pour calculer la valeur de la fonction objectif pour un placement donné. L'optimisation du coût de placement est soumise à un ensemble de contraintes que nous présentons dans la section suivante.

5.2.3 *Contraintes*

Nous considérons quatre contraintes liés à notre problème de placement :

1. **Contrainte de stockage :** la première contrainte est représentée par l'équation 44 **b**. Elle assure, pour chaque périphérique de stockage d_j que la taille de l'ensemble des objets qu'il héberge ne dépasse pas la capacité de stockage totale de ce dernier.

2. **Contrainte de performance :** la deuxième contrainte exprimée par l'équation 44 **c** assure pour chaque périphérique de stockage d_j que la performance (IOPS) demandée par l'ensemble des objets qu'il héberge ne dépasse pas la performance totale de ce dernier.

3. **Contrainte de SLA :** la troisième contrainte indiqué dans l'équation 44 **d** est liée au SLA. Elle assure pour chaque client u_k que le nombre d'IOPS livré à celui-ci est supérieur ou égal au nombre d'IOPS défini dans le SLA ($iops_{hard,u_k}$). Le paramètre $iops_{hard,u_k}$ représente le seuil de dégradation de service toléré par le client u_k.

4. **Contrainte d'unicité :** elle est exprimée par l'équation 44 **e**. Le but est de garantir que tous les objets dans le système ($\forall o_{i,u_k} \in O_{u_k}$) sont stockés sans réplication. Il faut noter que la réplication des données n'est pas considérée dans ce travail.

Pour résoudre le problème d'optimisation du placement que nous venons de formaliser, nous proposons deux approches heuristiques différentes.

Avant de présenter nos approches, nous illustrons au préalable l'inefficacité des algorithmes exacts en terme de temps d'exécution. Nous désignons par un

algorithme exact tout algorithme explorant toutes les solutions possibles pour trouver la solution optimale.

5.3 ALGORITHME EXACT

Une solution naïve à notre problème d'optimisation consisterait à énumérer et à évaluer tous les placements d'objets possibles puis de sélectionner celui qui a le coût optimal et qui satisfait toutes les contraintes. La complexité d'un tel algorithme est $O(2^{|O|})$, où $|O|$ est le nombre total d'objets dans le Cloud. Le temps d'exécution de cette solution évolue de manière exponentielle, ce qui n'est pas pertinent car la décision doit être effectuée en temps réel (runtime).

Pour confirmer cette hypothèse, nous avons implanté et évalué une approche d'énumération de toutes les solutions possibles de placement des objets. La plateforme expérimentale et la configuration sont détaillées dans la Sections 5.6.1.

Nous avons mis en œuvre l'algorithme exact en utilisant une méthode par séparation et évaluation (en anglais *branch and bound*). Cette méthode est considérée comme le moyen générique le plus utilisé pour la résolution exacte des problèmes d'optimisation combinatoire [95]. Elle effectue une énumération intelligente de l'espace des solutions.

5.3.1 *Algorithme branch and bound*

L'algorithme consiste à séparer de manière récursive le problème en sous problèmes de cardinalité inférieure tant que la résolution de ces problèmes reste difficile. Le cardinal de l'ensemble à explorer est réduit en imposant à cet ensemble des contraintes supplémentaires (réduction du domaine). Une série de tests appliquée à tous les sous problèmes permet de supprimer de l'espace de recherche les sous problèmes qui ne peuvent pas engendrer de la solution optimale. Cette recherche par décomposition de l'ensemble des solutions peut être représentée graphiquement par un *arbre de recherche*.

5.3.1.1 *Arbre de recherche*

Avant d'entamer la résolution du problème d'optimisation, nous devons construire l'arbre de recherche et d'énumération. L'arbre est construit en représentant chaque objet dans le système par un nœud. Chaque nœud dans l'arbre possède exacte-

ment deux descendants le première avec une valeur de 0 ce qui veut dire que l'objet est stocké dans un HDD et le deuxième avec une valeur de 1 ce qui veut dire que l'objet est placé dans un SSD (voir la figure 5.2 ci-dessous).

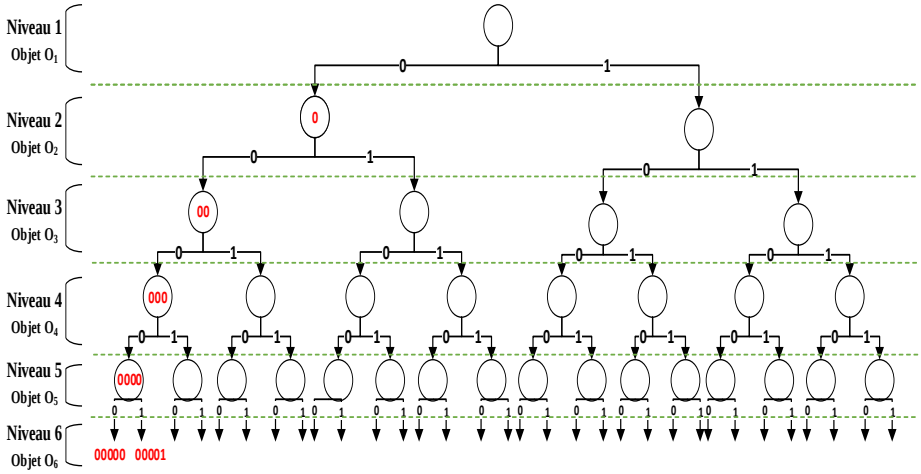

FIGURE 5.2 : Arbre de recherche et d'énumération

Chaque niveau dans l'arbre représente un objet. Nous construisons un placement en arrangeant séquentiellement les poids des nœuds entre la racine et la feuille. L'espace de recherche (les solutions possibles) est représenté par l'ensemble des chemins routes entre la racine et l'ensemble de feuilles. La hauteur de l'arbre est égale au nombre d'objet dans le système. La figure 5.3 illustre un exemple d'un arbre de recherche et d'énumération de trois objets.

5.3.1.2 *Algorithme de parcours*

Comme cité auparavant, la méthode de branch and bound (procédure par évaluation et séparation progressive) consiste à énumérer les solutions d'une manière intelligente en utilisant certaines propriétés du problème en question. Cette technique arrive à éliminer des solutions partielles qui ne mènent pas à la solution recherchée dans un temps réduit.

Notre réflexion du parcours pour la recherche de la solution optimale est basée sur l'élimination explicite de toutes les solutions non pertinentes du problème. Si un placement partiel sature les disques en espace ou en performance, tout

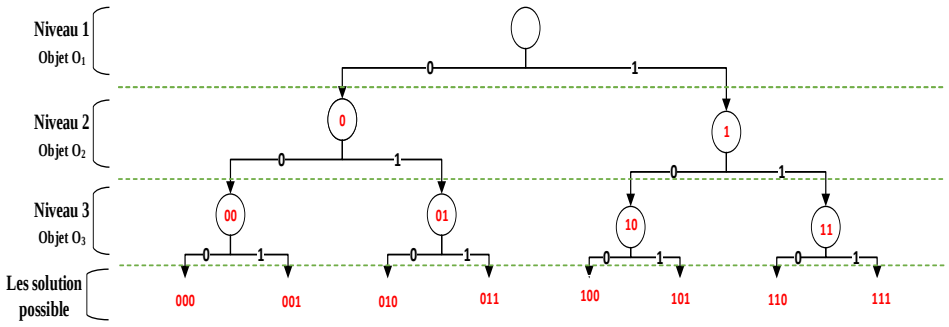

FIGURE 5.3 : Exemple d'un arbre de recherche

placement d'objets supplémentaires ne fait que saturer d'avantage ces disques. Donc, en faisant le parcours, notre algorithme lance une opération de sondage pour chaque nœud et cala en contrôlant si les contraintes d'optimisation sont vérifiées (Stockage, performance). Si l'une des contraintes est violée, notre algorithme exclue toute les solutions dérivées de la solution actuelle (voir la figure 5.4) et fait un retour-arrière pour développer l'autre branche. Sinon, il remonte dans l'arborescence vers un autre nœud situé à un niveau inférieur ou égal dans l'arborescence.

L'opération de sondage consiste à vérifier les trois contraintes suivantes :

1. La contrainte du stockage : la taille des objets déjà placés ne dépasse pas les capacités de stockage des supports disponibles.
2. La contrainte de performance : le placement actuel du sous-ensemble d'objet doit vérifier la contrainte de performance
3. La contrainte du coût : le coût du placement partiel est inférieur au coût minimum déjà rencontré.

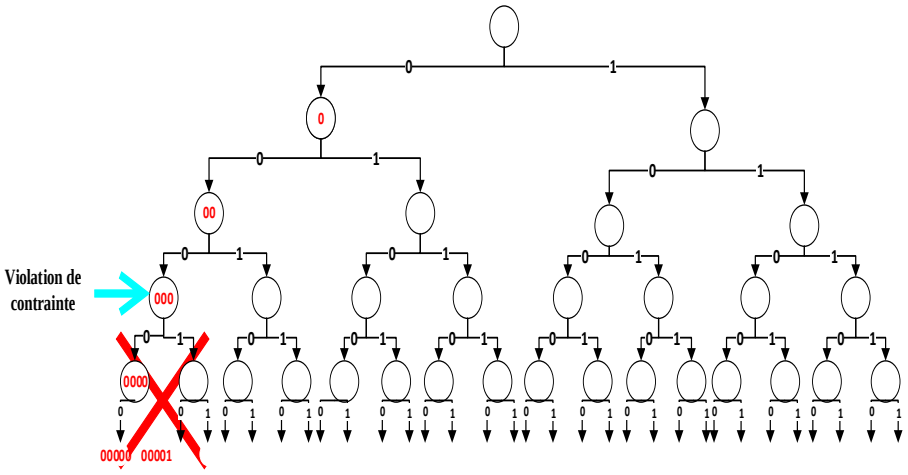

FIGURE 5.4 : Parcours d'un arbre de recherche pour trois objets

5.3.2 *Résultats d'application de l'algorithme exact*

La figure 5.5 et le tableau 5.1 montrent l'impact du nombre d'objets sur le temps d'exécution alors que le tableau 5.1 montre le temps d'exécution par extrapolation.

Nous observons que la résolution d'un problème de 50 objets (qui pourrait être un simple module d'un système ERP) peut prendre de nombreuses années. Par conséquent, les algorithmes exacts pourraient difficilement aboutir à un résultat dans un délai raisonnable. Ces résultats montrent que l'utilisation de l'algorithme exact n'est pas adaptée dans une stratégie de placement dans le Cloud où une décision de placement doit être prise rapidement (quelques secondes). Cela le disqualifie pour être utilisé en ligne.

Pour faire face à la complexité exponentielle de la méthode exacte, nous proposons dans la prochaine partie deux algorithmes d'approximation. Le premier (G-COPS) est basé sur un algorithme génétique tandis que le deuxième (H-COPS) est basée sur un algorithme heuristique incrémental ad-hoc.

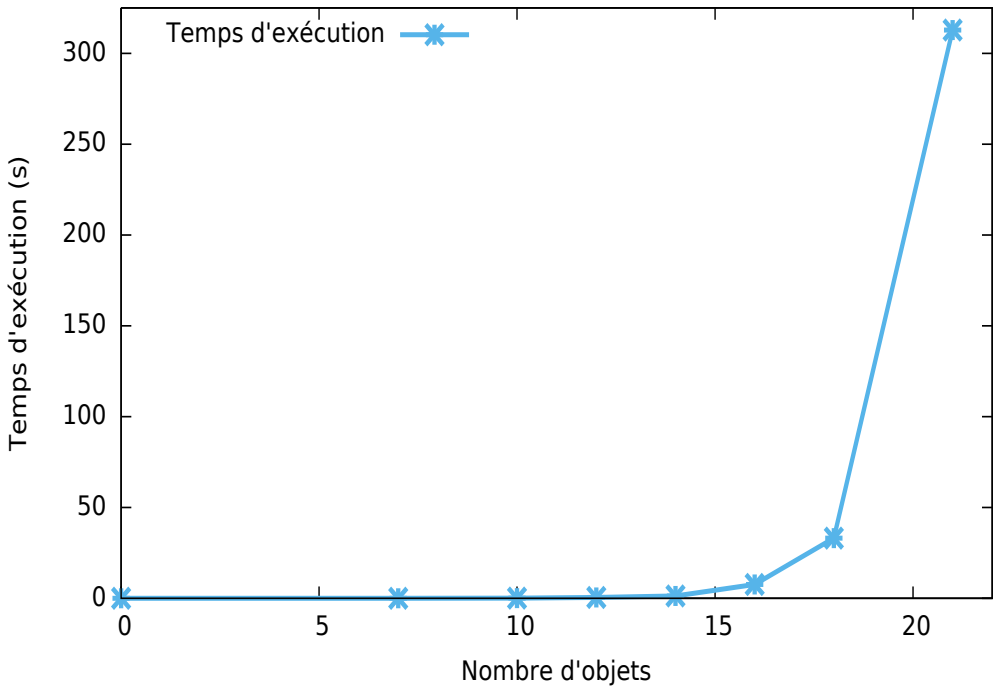

FIGURE 5.5 : Temps d'exécution de l'algorithme exact

TABLE 5.1 : Temps d'exécution par extrapolation

Nombre d'objet	Temps
10	<1 seconde
20	2 minutes
30	1.4 jours
40	4 jours
50	4033 ans

5.4 ALGORITHME G-COPS

5.4.1 *Principe général des algorithmes génétiques*

Les algorithmes génétiques tentent de simuler le processus d'évolution naturelle suivant le modèle darwinien dans un environnement donné. Ils utilisent un vocabulaire similaire à celui de la génétique naturelle [52]. Cependant, les processus naturels auxquels ils font référence sont beaucoup plus complexes. On parlera ainsi d'individu dans une population.

L'individu est représenté par un chromosome constitué de gènes qui contiennent les caractères héréditaires de l'individu. Les principes de sélection, de croise-

ment, de mutation s'inspirent des processus naturels de même nom.

Pour un problème d'optimisation donné, un individu représente un point de l'espace d'états, une solution potentielle. On lui associe la valeur du critère à optimiser, son adaptation. On génère ensuite de façon itérative (voir la figure 5.6) des populations d'individus sur lesquelles on applique des processus de sélection, de croisement et de mutation.

La sélection a pour but de favoriser les meilleurs éléments de la population pour le critère considéré (les mieux adaptés). Le croisement et la mutation assurent l'exploration de l'espace d'états.

On commence par générer une population aléatoire d'individus. Pour passer d'une génération k à la génération $k+1$, les opérations suivantes sont effectuées :

1. Dans un premier temps, la population est reproduite par sélection où les bons individus se reproduisent mieux que les mauvais.
2. Ensuite, on applique un croisement aux paires d'individus (les parents) d'une certaine proportion de la population (probabilité P_c , généralement autour de 0.6 [52]) pour en produire de nouveaux (les enfants).
3. Un opérateur de mutation est également appliqué à une certaine proportion de la population (probabilité P_m, généralement très inférieure à P_c).
4. Enfin, les nouveaux individus sont évalués et intégrés à la population de la génération suivante.

L'algorithme 1 illustre le déroulement d'un algorithme génétique.
Plusieurs critères d'arrêt de l'algorithme sont possibles : le nombre de générations peut être fixé a priori (temps constant) ou l'algorithme peut être arrêté lorsque la population n'évolue plus suffisamment rapidement (i.e aucune amélioration après x itérations).

Pour utiliser un algorithme génétique sur un problème d'optimisation on doit donc disposer d'un principe de codage des individus, d'un mécanisme de génération de la population initiale, d'opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace de recherche.

5.4.2 *Codage*

Dans ce travail, nous utilisons un codage binaire. Chaque individu est représenté par un ensemble de bits. Un individu de l'algorithme génétique est représenté

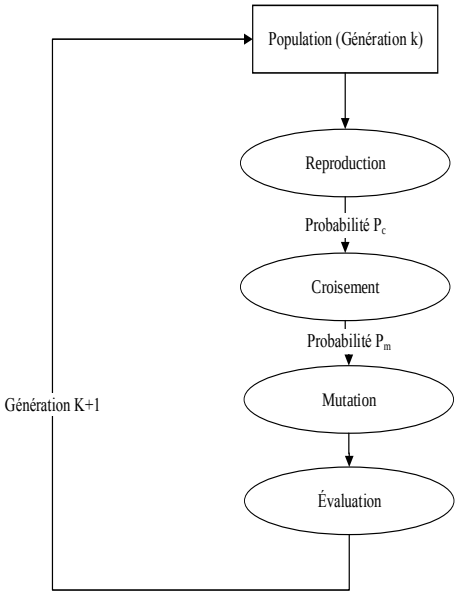

FIGURE 5.6 : Principe général des algorithmes génétiques

par un chromosome constitué d'une chaîne de gènes. Chaque gène (bit) de l'individu est associé à un objet de base de données. Il prend la valeur 0 s'il est placé dans un disque HDD, ou 1 s'il est placé dans un disque SSD. La figure 5.7 illustre un exemple d'un individu (chromosome).

La cardinalité d'un individu (le nombre total des gènes dans un individu) est égale au nombre total d'objets. Nous construisons un individu (un chromosome) en arrangeant séquentiellement les gènes représentant les objets du même client (voir la figure 5.7).

5.4.3 *Initialisation de la population*

Cette opération consiste à produire une population d'individus non homogène qui servira de base pour les générations futures. Le choix de la population initiale est important car il peut rendre plus ou moins rapide la convergence de l'algorithme génétique vers l'optimum global.

D'un point de vue général, on peut dire qu'une hétérogénéité maximale est souhaitable (analogue à la diversité génétique des populations). En effet, plus la population initiale sera hétérogène, plus la proportion de l'espace des solutions explorée sera importante et donc, moins on aura de risque de manquer l'opti-

Algorithme 1 : Aperçu de G-COPS

Données : N : Nombre d'individus dans la population ;
CR : probabilité de croisement ; MR : probabilité de mutation ;
SC : critère d'arrêt (nombre d'itérations sans amélioration) ;
NewChrom : Chromosome ; P : Population ;
Résultat : meilleur placement

```
/* Génération aléatoire de la population initiale.                    */
/* GenerateRandomChromosome: fonction qui génère un chromosome aléatoirement.  */
/* InsertInPopulation(P, NewChrom): fonction qui insert le chromosome NewChrom dans P.
   */
```

1 **pour** *i :=1 to N* **faire**
2 NewChrom := GenerateRandomChromosome() ;
3 InsertInPopulation(P, NewChrom) ;

```
/* Évaluer chaque chromosome dans P en utilisant notre modèle de coût.        */
/* Evaluate(P): fonction qui évalue chaque chromosome dans P.                 */
/* GetBest(P): fonction qui récupère le meilleur chromosome dans P.           */
```

4 Evaluate(P) ;
5 BestChromosome := GetBest(P);
6 NI=0 ;
7 **répéter**

```
    /* Croisement & Mutation.                                                 */
    /* SelectChromosome(P): fonction qui sélectionne aléatoirement un chromosome.  */
    /* Crossover(p1,p2, CR): fonction de croisement des chromosomes p1 et p2 selon la
       probabilité CR.                                                        */
    /* Mutation(offspring, MR) fonction de mutation selon la probabilité MR.  */
```

8 p1 := SelectChromosome(P) ;
9 p2 := SelectChromosome(P) ;
10 Offspring := Crossover(p1,p2, CR);
11 Offspring := Mutation(offspring, MR) ;
12 InsertInPopulation(P, Offspring) ;

```
    /* Réduire la taille de P à N chromosomes                                 */
    /* ReducePopulation(P, N): fonction qui réduit la taille de P à N chromosomes  */
```

13 ReducePopulation(P, N) ;
14 Evaluate(P) ;

```
    /*  Mettre à jour le meilleur chromosome si nécessaire.                   */
    /*  Cost(p1): fonction qui calcule le coût du chromosome p1               */
```

15 **si** *Cost(BestChromosome) > Cost(GetBest(P))* **alors**
16 BestChromosome = GetBest(P);
17 NI :=0 ;
18 **sinon**
19 NI++ ;
20 **jusqu'à** *NI > SC*;
21 **retourner** BestChromosome ;

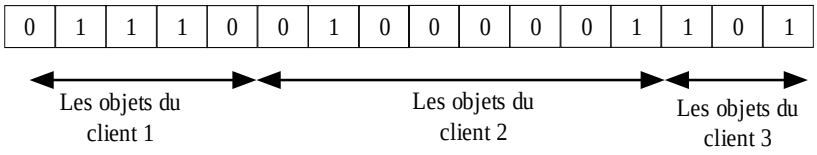

FIGURE 5.7 : Codage d'un individu

mum global. Cette hétérogénéité peut être générée aléatoirement ou de façon plus déterministe.

Dans notre cas, nous n'avons aucune connaissance préalable du problème à résoudre. Par conséquent, nous avons généré les individus initiaux de manière aléatoire. Ces individus serviront de base pour les générations futures. Il faut noter qu'il peut y avoir des solutions infaisable dans la population initiale qui ne vérifient pas toutes les contraintes du problème d'optimisation. Dans ce travail, nous avons expérimenté avec deux variantes : (1) la première élimine les solutions infaisables qui ne satisfont pas les contraintes d'optimisation et la deuxième version (2) garde les solutions infaisables afin de préserver la diversité.

5.4.4 *Croisement*

L'opérateur de croisement prend en entrée un couple d'individus parents P_1 et P_2 et renvoie un couple d'individus enfants C_1 et C_2. Les enfants sont obtenus en choisissant aléatoirement un point de croisement (ou éventuellement plusieurs points de croisement pour éviter certains effets de bord du codage) dans les chromosomes et en recopiant dans le fils C_1 les gènes de P_1 jusqu'au point de croisement puis en complétant avec les gènes de P_2 . On effectue l'opération symétrique pour C_2.

Dans cette thèse, nous avons comparé quatre méthodes de croisement à savoir :

1. Croisement à un seul point (SPC) : un seul point de croisement est utilisé. Le point de croisement est sélectionné aléatoirement
2. Croisement à deux points (TPC) : deux points de croisement sont utilisés. Les points de croisement sont sélectionnés aléatoirement
3. Croisement d'un seul point par client (SPPCC) : pour chaque client un point de croisement est sélectionné. Le nombre de points de croisement est

égal au nombre de clients dans le système. Les points de croisement sont sélectionnés aléatoirement.

4. Croisement aléatoire d'un seul point par client (RSPPCC) : pour chaque client, une décision aléatoire est prise en ce qui concerne la sélection ou non d'un point de croisement. Si la décision est positive, un point de croisement est sélectionné aléatoirement. Sinon, aucun croisement n'est appliqué pour les gènes qui représentent les objets du client en question.

5.4.5 *Mutation*

Par analogie à la mutation biologique, l'opérateur de mutation classique prend en entrée un individu P sélectionné pour la mutation et renvoie un individu mutant P' obtenu par transformation locale de l'un des gènes de P. L'individu mutant P' obtenu en choisissant aléatoirement un/plusieurs points (gènes) de mutation ensuite inverser le codage de 0 à 1 et vice-versa.

Dans ce travail, nous avons expérimenté quatre méthodes de mutation :

1. Mutation à un seul point (SPM) : un seul point de mutation est utilisé pour tous les gènes d'un individu. Le point de mutation est sélectionné aléatoirement.

2. Mutation à deux points (TPM) : deux points de mutation sont sélectionnés. La sélection des points se fait aléatoirement.

3. Mutation à un seul point par client (SPPCM) : pour chaque client un point de mutation est sélectionné. Le nombre de points de mutation est égal au nombre de clients dans le système. La sélection des points se fait aléatoirement.

4. Mutation aléatoire à un seul point par client (RSPPCM) : à l'image du croisement, pour chaque client une décision aléatoire est prise en ce qui concerne la sélection ou non d'un point de mutation. Si la décision est positive, un point de mutation sera sélectionné aléatoirement. Sinon aucune mutation n'est appliquée pour les gènes qui représentent les objets du client en question.

5.5 ALGORITHME H-COPS

L'idée de notre algorithme heuristique est de commencer par la construction d'un plan de placement initial dans **une phase d'initialisation**, ensuite l'améliorer dans deux phases supplémentaires afin d'obtenir le meilleur plan de place-

Algorithme 2 : Apercu de H-COPS

Données : Objets, les classes de stockage, SLA, le motif d'accès
Résultat : meilleure placement
1 **Initialisation :** construire un plan de placement initial. ;
2 **Faisabilité :** rendre le plan de placement faisable. ;
3 **Optimisation** : optimiser le plan de placement précédent. ;

ment des objets. Le pseudo-code de H-COPS est donné dans l'algorithme 2.

Le plan de placement initial est construit en plaçant chaque objet dans la classe de stockage ayant le coût de stockage le plus faible. Il faut noter que cette étape ignore les contraintes du problème d'optimisation (voir équation 44) et peut générer un placement infaisable (i.e. un placement qui ne respecte pas la contrainte du stockage ou de performance).

La deuxième phase met à jour le plan de placement initial pour le rendre faisable. Un plan faisable est un plan de placement qui remplit toutes les contraintes du problème d'optimisation. Pour ce faire, nous modifions le plan du placement initial, si nécessaire, pour satisfaire les contraintes de capacité (stockage et performance) et du SLA. Cette phase est nommée **étude de faisabilité**.

La troisième phase nommée **phase optimisation** améliore le plan de placement issu de la deuxième phase en optimisant le coût de placement. L'optimisation consiste à éliminer les pénalités engendrées par la violation du SLA. Nous détaillons chaque étape ci-dessous. L'algorithme 3 illustre notre approche H-COPS.

5.5.1 *Initialisation*

Le but de cette phase est de fournir un plan de placement initial des objets. Nous le construisons en plaçant chaque objet $o_{i,u_k} \in O$ dans la classe de stockage ayant le coût de stockage le plus faible (voir l'algorithme 3). Soit $d_{cheap}(o_{i,u_k})$ la classe de stockage qui a le coût de stockage le plus faible pour l'objet o_{i,u_k}. Nous calculons $d_{cheap}(o_{i,u_k})$ comme suit :

$$d_{cheap}(o_{i,u_k}) = \min\{Cost(o_{i,u_k}, d_{hdd}), Cost(o_{i,u_k}, d_{ssd})\} \tag{45}$$

$Cost(o_{i,u_k}, d_{hdd})$ (respectivement $Cost(o_{i,u_k}, d_{ssd})$) représente le coût de placement de o_{i,u_k} dans la classe de stockage HDD (respectivement SSD).

Nous calculons le coût de placement d'un objet o_{i,u_k} ($Cost(o_{i,u_k}, d_{hdd})$ ou $Cost(o_{i,u_k}, d_{ssd})$) en utilisant le modèle de coût proposé dans le chapitre précé-

dent. Ce dernier inclut à la fois le coût de stockage et le coût de migration afin de prendre en compte le placement actuel.

De même, nous notons $d_{costly}(o_{i,u_k})$ la classe de stockage qui a le coût de placement le plus élevé pour l'objet o_{i,u_k}.

5.5.2 *Faisabilité*

Cette phase met à jour et améliore le plan de placement initial construit dans la première phase afin de le rendre faisable. Un placement faisable devrait satisfaire toutes les contraintes définies dans l'équation 44. Nous soulignons que la contrainte d'unicité (44 e) est vérifiée, par défaut, dans la première phase, car un objet est stocké soit dans une classe de stockage SSD ou dans une classe de stockage HDD.

L'étude de faisabilité se fait en deux étapes à savoir :

1. Validation de la contrainte de stockage (44 b) : Le but est de garantir que la taille globale des objets placés dans chaque classe de stockage ne dépasse pas la capacité de cette dernière.
2. Validation de contrainte de SLA (44 c) et de performance (44 d) : Le but est d'assurer que les SLA des clients sont vérifiés et que la contrainte de la capacité de performance n'est pas violée.

5.5.2.1 *Contraintes de stockage*

Pour chaque classe de stockage, nous devons assurer que la taille des objets qui y sont hébergées ne dépasse pas la capacité totale de la classe de stockage. Cette étape est ignorée si la contrainte de stockage est déjà vérifiée. Sinon, si la taille de l'ensemble d'objets est supérieure à la capacité totale du stockage des deux classes de stockage, le processus d'optimisation se termine en affichant *"Échec, aucune solution possible"* (voir l'algorithme 3).

Si l'une des classes de stockage est surchargée après la phase d'initialisation (i.e. la taille des objets qui y sont hébergées dépasse la capacité totale du stockage, nous mettons à jour le plan de placement des objets en déplaçant un sous ensemble d'objets vers la classe de stockage qui n'a pas atteint sa limite de capacité.

La sélection des objets à déplacer est basée sur un algorithme glouton comme le montre l'algorithme 3. Par conséquent, un score δ est calculé pour chaque

objet o_{i,u_k} dans la classe surchargée. Soit $\delta(o_{i,u_k})$ le score de l'objet o_{i,u_k}. Nous calculons δ comme suit :

$$\delta(o_{i,u_k}) = \frac{s_{o_{i,u_k}}}{d_{costly}(o_{i,u_k}) - d_{cheap}(o_{i,u_k})} \tag{46}$$

Les objets sont triés en fonction de leur score dans l'ordre décroissant. Ensuite, ils seront déplacés dans l'ordre un par un jusqu'à ce que la contrainte de stockage soit satisfaite.

L'idée dernière le calcul de score est de déplacer les objets qui engendrent moins de perte financière ($) et libère le maximum d'espace. En outre, le score donne la priorité aux objets ayant une grande taille et montrant une faible différence en termes de coût monétaire entre les deux classes de stockage.

5.5.2.2 *Contraintes du SLA et performance*

Pour chaque client u_k, le système doit garantir que l'IOPS offert ($iops_{offert,u_k}$) est supérieur ou égal à l'$iops_{hard,u_k}$ spécifié dans son SLA. Lorsque l'IOPS réellement demandé par le client est inférieur à l'$iops_{hard,u_k}$, l'algorithme doit garantir que le $iops_{offert,u_k}$ est égal à l'IOPS réellement demandé ($iops_{requested,u_k}$).

Par conséquent, pour chaque client $u_k \in U$, nous vérifions sa contrainte SLA. Si la contrainte est satisfaite, aucun objet lui appartenant ne sera déplacé. Dans le cas contraire (SLA violé), nous mettons à jour le plan de placement des objets du client concerné afin de trouver un placement qui satisfait sa contrainte SLA et optimise le coût monétaire pour le fournisseur de service Cloud (voir l'algorithme 3). Notons que seuls les objets de client concerné qui sont affectés.

Le plan est mis à jour en déplaçant un sous ensemble d'objets du client en question (i.e. le client qui a subit une violation) de la classe de stockage la moins performante (qui est la classe de stockage HDD) vers la classe du stockage la plus performante (qui est la classe de stockage SSD). La sélection des objets à déplacer est basée sur un algorithme glouton. Un score λ est calculé pour chaque objet o_{i,u_k} comme indiqué dans l'algorithme 3 (phase de faisabilité). Soit $\lambda(o_{i,u_k})$ le score de l'objet o_{i,u_k}. Nous calculons λ comme suit :

$$\lambda(o_{i,u_k}) = \frac{t_{exec}(o_{i,u_k}, d_{costly}) - t_{exec}(o_{i,u_k}, d_{cheap})}{d_{costly}(o_{i,u_k}) - d_{cheap}(o_{i,u_k})} \tag{47}$$

Une fois le score λ est calculé. Nous trions les objets du client en fonction de leur score dans un ordre décroissant. Nous mettons à jour le plan de placement des objets en déplaçant les objets dans l'ordre (i.e. en fonction de leur score) un

par un jusqu'à ce que la contrainte SLA du client soit satisfaite ou qu'il n'y ait plus d'espace disponible dans la classe SSD. Dans ce dernier cas, s'il n'y plus d'espace disponible dans la classe de stockage SSD, le processus d'optimisation se termine. Il faut noter qu'avant d'apporter un changement sur le plan du placement, le système doit vérifier si la contrainte de stockage est toujours respectée.

L'idée derrière le calcul du score λ est de déplacer les objets engendrant une amélioration des performances et moins de perte financière pour le fournisseur de Cloud (i.e. les objets de petites tailles et qui ont une forte lecture aléatoire par exemple). Le score de priorité (λ) est un rapport entre le gain de performance et la perte financière causée par le déplacement de l'objet comme indiqué dans l'équation 47. Le gain en performance est représenté par le gain en temps d'exécution enregistré à cause du déplacement de l'objet.

Nous calculons le temps d'exécution des requêtes d'E/S transmises à l'objet o_{i,u_k} stocké dans la classe d_j ($t_{exec}(o_{i,u_k}, d_j)$ à partir du nombre de requêtes d'E/S transmises à ce dernier et le temps de réponse de la classe de stockage (t_{op,d_j}) comme indiqué dans l'équation 48 ci-dessous.

$$t_{exec}(o_{i,u_k}, d_j) = \sum_{op=\{rr,sr,rw,sw\}} req_{op,o_{i,u_k}} * t_{op,d_k} \tag{48}$$

La valeur de $req_{op,o_{i,u_j}}$ est obtenue à partir d'une phase de surveillance (*monitoring*), tandis que la valeur t_{op,d_j} est obtenue à partir de la fiche technique des devises constituent la classe de stockage ou dans une phase de calibration.

5.5.3 *Optimisation*

Dans cette dernière étape, nous améliorons le plan de placement résultant de la phase précédente en éliminant les pénalités engendrées par la violation du *soft SLA* des clients ($iops_{soft,u_k}$) comme le montre l'algorithme 3. Cependant, éviter les pénalités augmente le coût monétaire de stockage car nous devons placer plus d'objets dans la classe de stockage SSD qui est plus coûteuse en termes de Go/$ par rapport à la classe de stockage HDD.

L'idée est d'étudier le compromis entre le coût de stockage à supporter pour satisfaire le *soft SLA* et le coût de pénalité dans le cas d'une violation du *soft SLA*. Soit O_{soft,u_k} l'ensemble des objets du client u_k qui doivent être déplacés vers la classe de stockage SSD afin d'éviter la pénalité. Nous obtenons le coût monétaire supplémentaire nécessaire pour satisfaire le *soft SLA* et éviter la pénalité du client u_k ($Cost_{sla}(u_k)$) comme indiqué dans l'équation (49) ci-dessous.

$$Cost_{sla}(u_k) = \sum_{o_{i,u_k} \in O_{soft,u_k}} d_{costly}(o_{i,u_k}) - d_{cheap}(o_{i,u_k}) \tag{49}$$

L'ensemble des objets du client u_k qui doivent être déplacés vers la classe de stockage SSD afin d'éviter la pénalité (O_{soft,u_k}) est obtenu en utilisant l'algorithme glouton indiqué précédemment. Une fois le coût monétaire de satisfaction du *soft SLA* déterminé, l'administrateur Cloud peut choisir entre le coût de satisfaction du SLA et le coût de pénalité.

En revanche, si la violation est présente pour plusieurs clients, l'administrateur Cloud doit traiter ces violations dans l'ordre. Par exemple, il serait intéressant de commencer par les clients ayant une pénalité très élevée et nécessitant moins d'efforts en terme de coût de satisfaction du SLA (i.e. ceux qui ont un $Cost_{sla}(u_k)$ faible).

Par conséquent, pour chaque client u_k, nous calculons un score ($\mu(u_k)$). $\mu(u_k)$ représente la différence entre le coût de pénalité du client (u_k) ($Cost_{pnl}(u_k)$) et son $Cost_{sla}(u_k)$ comme indiqué dans l'équation (50). Nous trions les clients en fonction de leur score $\mu(u_k)$ par ordre décroissant. Les clients sont traités dans l'ordre un par un jusqu'à ce que tous les soft SLA soient satisfaits, ou que les ressources (stockage et performance) de la classe de stockage SSD soient épuisées ou critiques (elle atteint le seuil défini par l'administrateur de Cloud).

$$\mu(u_k) = Cost_{pnl}(u_k) - Cost_{sla}(u_k) \tag{50}$$

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

5.6 ÉVALUATION

Cette section présente l'ensemble des expérimentations menées afin évaluer l'efficacité de G-COPS et de H-COPS. Nous commençons d'abord par monter la pertinence de nos algorithmes pour les petites instances de problème ensuite les larges instances de problème. Nous désignons par large instance de problème, une instance qui ne peut pas être résolue dans un délai raisonnable en utilisant un algorithme exact. Dans notre contexte, le problème est considéré comme large, du point de vue algorithmique, lorsque le nombre total d'objets dépasse 30, car sa résolution en utilisant un algorithme exact prend plus d'une journée (voir Table 5.1), tandis que la décisions doit être prise plusieurs fois par jour (de 1 à 24 fois par jour selon la littérature [79, 132]).

Deux métriques ont été utilisées pour valider la pertinence de nos algorithmes à savoir :

Algorithme 3 : Algorithme H-COPS

Données : Objets, Classe de Stockage, SLA, motif d'accès

Résultat : placement

Initialisation;

pour *Pour chaque objet* $o_{i,u_k} \in O$ **faire**

    calculer $d_{cheap}(o_{i,u_k})$;

    placer o_{i,u_k} dans la classe ayant le coût de stockage le plus faible ;

Faisabilité;

si *(la contrainte de stockage est violée)* **alors**

    pour *pour chaque* $o_{i,u_k} \in O_{d_j}$ **faire**

        Calculer $\delta(o_{i,u_k})$;

    Trier les objets de la classe de stockage surchargée;

    Déplacer les objets jusqu'à ou la contrainte de stockage soit vérifiée;

pour *Pour chaque client* $u_k \in U$ **faire**

    si $(iops_{offered,u_k} < iops_{hard,u_k})$ **alors**

        pour *pour chaque objet* $o_{i,u_k} \in O_{u_k}$ **faire**

            calculer $\lambda(o_{i,u_k})$;

        Trier les objets du client;

        Déplacer les objets jusqu'à ou le hard SLA soit vérifié;

Optimisation;

pour *pour chaque client* $u_k \in U$ **faire**

    calculer $\mu(u_k)$;

Trier les client selon μ ;

Traiter client par client dans l'ordre jusqu'à la fin ou l'épuisement des ressources SSD;

1. Le taux d'erreur en terme de coût de placement : cette métrique mesure l'écart de coût entre le placement optimal et celui trouvé par nos algorithmes.

2. Le temps d'exécution des algorithmes : cette métrique nous permet de vérifier si l'algorithme s'exécute en un temps raisonnable ou non.

Une fois la pertinence validée, nous analysons dans une deuxième phase la qualité de la solution de placement proposée par nos algorithmes en termes de SLA. Nous montrons à quel point nous nous rapprochons du SLA (la QoS demandée). Pour ce faire, nous comparons pour chaque client la qualité de service délivrée après avoir exécuté le placement et la qualité de service demandée. Idéalement, la QoS offerte doit être plus élevée et très proche de la qualité demandée afin d'éviter à la fois le sur-approvisionnement et la pénalité.

Dans l'ensemble des expériences, nous comparons nos algorithmes avec les solutions exactes, DOT [132] et OPA [33]. En effet, DOT cherche à trouver un placement qui optimise le coût d'exploitation total (TOC) et satisfait le SLA. Par contre, OPA est orientée performance. Comme DOT et OPA ne prennent pas en compte la multi-location, nous supposons que tous les objets appartiennent à un seul client. Les algorithmes sont lancés sur la base qu'il n'existe qu'un seul client qui détient l'ensemble des objets. Une fois l'exécution terminée, nous évaluons et analysons chaque client séparément en ce qui concerne la violation du SLA et la pénalité.

Le calcul du coût de placement des objets et l'évaluation des SLA sont appliqués à une granularité du client pour tous les algorithmes testés. Nous utilisons le modèle de coût présenté dans le chapitre précédent pour l'évaluation du coût de placement des objets.

Avant de détailler le processus d'évaluation, nous présentons d'abord la plateforme expérimentale, ensuite nous résumerons brièvement les résultats de la phase calibration effectuée pour G-COPS afin ajuster les paramètres généraux de l'algorithme génétique.

5.6.1 *Plateforme expérimentale*

Nous avons expérimenté en utilisant l'architecture illustré dans la figure 5.1. Toutes les expériences ont été effectuées dans des machines virtuelles exécutées sur un serveur VMware ESXi-5.1.0. Chaque machine virtuelle était configurée avec un vCPU de 8 cores et 4Go de RAM. L'hyperviseur ESX s'exécute sur un serveur HP G9 DL380 doté d'un processeur Intel Xeon 2,4 GHz et 32 Go de

RAM. Nous avons utilisé PostgreSQL (version 9.3.5) comme SGBD. Ce dernier est installé au-dessus du système d'exploitation GNU Linux (distribution Debian version 7.5 avec un noyau Linux 3.19.5).

Une seule instance a été utilisée pour gérer plusieurs bases de données de différents clients (multi-locataire). Les données étaient stockées dans un système stockage hybride composé d'un disque HDD Hitachi 7200 RPM modèle 1SB HDS721010CLA632 et d'un disque SSD Samsung 850 PRO de 128Go.

5.6.2 *Calibration de G-COPS*

Le but de cette phase est de choisir la meilleure configuration parmi toutes les configurations possibles et de répondre aux questions suivantes :

1. Quelle est la taille adéquate de la population initiale ?
2. Quelle est la méthode de mutation la mieux adaptée à ce problème d'optimisation ?
3. Quelle est la méthode de croisement la mieux adaptée à ce problème d'optimisation ?
4. Est-il intéressant d'inclure les solutions irréalisables (infaisables) ?

Méthodologie de validation :
Nous avons conduit plusieurs séries d'expériences en utilisant de petites instances. Dans chaque série d'expériences nous modifions la configuration et nous comparons la précision de G-COPS par rapport à l'algorithme exact en termes de coût. Les résultats sont donnés ci-dessous.

- *Taille de la population initiale :* la figure 5.8 illustre l'impact de la taille de la population sur le temps d'exécution et la qualité de la solution retournée. Bien évidemment, l'augmentation de la taille de la population a un impact négatif sur le temps d'exécution et un impact positif sur la précision. Par conséquent, un compromis entre les deux est nécessaire. Nous jugeons qu'une population de 700 individus est adéquate car l'algorithme converge vers une solution (proche de l') optimale dans un temps raisonnable (quelques secondes).
- *Mutation et croisement :* le tableau 5.2 montre que RSPPC (respectivement RSPPM) donne une meilleure précision par rapport aux autres types de croisement (respectivement mutation). Nous observons également que RSPPC (respectivement RSPPM) donne le plus grand nombre de solutions acceptables, le plus grand nombre de solutions exactes et un écart-type plus

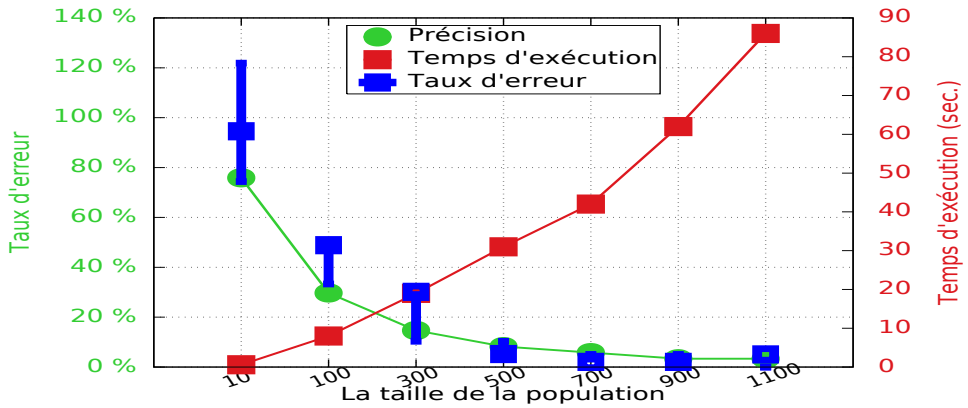

FIGURE 5.8 : Impact de la taille de la population sur le temps d'exécution et la qualité de la solution

TABLE 5.2 : Impact de la mutation et du croisement

		SP	SPPC	RSPPC
Croisement	Taux d'erreur moyen %	12.21	11.93	**10.36**
	Solutions acceptables %	68	74	**82**
	Ecart type	17.02	15.79	**14.14**
	Solutions exactes %	19	20	**22**
Mutation	Taux d'erreur moyen %	9.47	8.28	**7.9**
	Solutions acceptables%	72	82	**88**
	Ecart type	13.45	11.97	**9.40**
	Solutions exactes %	22	22	**22**

faible. Par conséquent, nous utilisons RSPPC (respectivement RSPPM) dans le reste de nos expérimentations.

- *Solution infaisable :* nous observons dans le tableau 5.3 que l'inclusion des solutions infaisables (voir l'algorithme 1) augmente la précision et améliore les résultats de la recherche. Par conséquent, nous gardons les solutions infaisables au sein des différentes générations.

5.6.3 *Benchmarks*

Nous avons mené toutes les expériences avec les benchmarks TPC-H et TPC-C. Pour les problèmes de petites instances, nous avons utilisé différentes configurations (facteur d'échelle, terminal/entrepôt, temps de réflexion, taux d'arrivé des requêtes, etc.) afin de prouver que les résultats restent valides dans différents scénarios et pour différentes configurations. Nous avons observé que la taille de la base de données et la charge de travail n'avaient aucun impact sur la précision

TABLE 5.3 : Gestion des solutions infaisables

	Solutions faisables	**Solutions infaisables**
Taux d'erreur moyen%	9.94	**7.81**
Solutions acceptables%	79	**89**
Ecart type	14.12	**9.4**
Solutions exactes %	22	**22**

et le temps d'exécution de G-COPS et de H-COPS. Par conséquent, une seule configuration a été utilisée dans la validation des algorithmes pour les grandes instances du problèmes, et cela pour les deux benchmarks TPC-C et TPC-H.

Pour simuler un environnement multi-locataire (multi-tenant), nous devons expérimenter au minimum avec deux bases de données de deux clients différents. Comme nous avions l'intention de comparer nos approches avec un algorithme exact pour les petites instances du problème, nous devons réduire le nombre d'objet afin de pouvoir résoudre le problème en utilisant un algorithme exact dans un temps qui reste raisonnable. Pour ce faire, nous avons simplifié le benchmark TPC-H, et cela en utilisant une charge de travail modifiée qui interrogeait quatre tables (lineitem, orders, customer, part) avec leur index. Nous avons utilisé 11 modèles de requêtes provenant des 22 requêtes TPC-H (Q1, Q3, Q4, Q6, Q12, Q13, Q14, Q17, Q18, Q19, Q22).

Pour montrer l'impact des pénalités sur le coût du placement, nous devons provoquer une violation de SLA pour avoir une pénalité. Pour ce faire, nous avons poussé l'utilisation du système de stockage au-delà de sa capacité. Nous avons réduit le temps de réflexion du client et changé le nombre de terminaux/entrepôts afin d'imiter une utilisation élevée des ressources d'E/S.

5.6.4 *Paramétrage du modèle de coût*

Comme mentionné ci-dessus, nous avons utilisé le modèle de coût présenté dans le chapitre précédent pour évaluer le coût de placement. Ce modèle intègre les coûts d'amortissement, d'énergie et de pénalité. Nous calculons le coût amortissement en ventilant le coût d'achat du périphérique de stockage sur une période de 5 ans. Le coût énergétique est calculé en utilisant un coût de 0,1 $ par kWh.

Le montant de la pénalité est calculé en utilisant le modèle adopté par les fournisseurs actuels du Cloud. Actuellement, ces derniers définissent des intervalles de QoS et les pénalités correspondantes [4]. La pénalité est représenté par un pourcentage qui sera déduit de la facture final du client. Nous avons utilisé les prix unitaires du service EBS (*Elastic Block Store*) d'Amazon pour calculer

le montant de la facture [1]. Nous avons modifié les pourcentages de pénalité pour simuler différents modèles de pénalité. Notons que la pénalité s'applique lorsque les IOPS livrés sont en dessous du soft SLA demandé pour un client donné.

5.6.5 *Les problèmes de petite instance*

5.6.5.1 *Méthodologie de validation*

Le tableau 5.4 détaille les différents scénarios avec lesquels on a expérimenté. Nous avons fixé la période d'évaluation à deux heures et nous avons utilisé dix minutes comme période d'adaptation (en anglais *warm-up*). Nous avons exécuté les benchmarks pendant une heure afin de caractériser la charge de travail et d'extraire le profil d'accès des E/S des différents objets. Nous avons utilisé notre outil [96] pour tracer les requêtes d'E/S et extraire le profil d'accès des E/S des objets.

A la fin de la première heure, les traces sont transférées à un processus d'optimisation. Ce dernier résout le problème de placement des objets en utilisant cinq algorithmes : G-COPS, H-COPS, DOT, OPA, et un exact.

Pour chaque algorithme nous exécutons le placement trouvé et nous transmettons à nouveau la même charge de travail (les benchmarks) pour une période d'une heure. À la fin de cette période, nous calculons le coût de placement associé à chaque algorithme.

5.6.5.2 *Résultats des expériences*

Les coûts présentés dans la figure 5.9 sont normalisés au coût de la solution optimale fournie par l'algorithme exact. Nous observons que G-COPS et H-COPS produisent les meilleurs résultats. Le taux d'erreur moyen relatif au coût optimal (le coût de la solution trouvée par l'algorithme exacte) est de 2% pour G-COPS, 6,06% pour H-COPS, 11,09% pour DOT et 50,97% pour OPA.

La non prise en compte du coût de migration est la principale cause d'inefficacité de DOT. Il faut noter que pour atteindre un placement d'objets donné, les objets doivent migrer d'un placement actuel vers le nouveau. La migration des objets engendre un coût supplémentaire qui doit être pris en compte. DOT propose un placement pertinent mais sa réalisation est coûteuse.

Cependant, la cause principale d'échec d'OPA revient à sa politique orienté performance. Les objets sont placés d'une façon à augmenter les performances

TABLE 5.4 : Configuration des scénarios

	TPC-C				TPC-H	
	taille (Go)	terminaux/ entrepôt	connexion simultanée	temps entre deux requêtes	taille (Go)	taux d'arrivé des requêtes
sc1	10	1.5	130	1	30	1.5
sc2	10	1	130	1	30	1
sc3	30	1	350	1	30	1
sc4	30	1	350	0.5	30	0.5
sc5	10	2	260	0	30	0
sc6	30	1	350	0	100	0

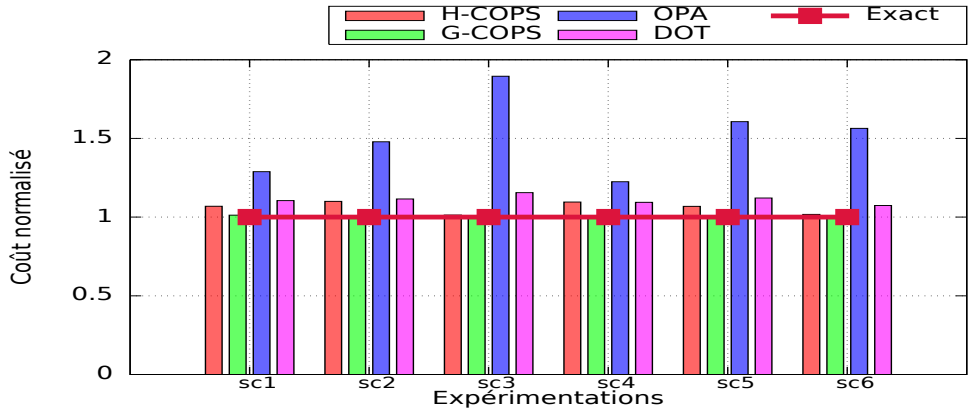

FIGURE 5.9 : Taux d'erreur pour les petites instances du problème

du système de stockage mais sans la prise en compte des SLAs des clients.

Nous observons que G-COPS et H-COPS proposent des placements optimisés en termes de coût par rapport à l'OPA à une moyenne respectivement de 50% et de 40%, alors qu'ils présentent une amélioration respectivement de 11% et 6% par rapport à DOT.

En ce qui concerne les temps d'exécution, nous observons dans la figure 5.10 que H-COPS, DOT et OPA s'exécutent en quelques dizaines de microsecondes, alors que G-COPS s'exécute en environ 40 secondes. Tous les algorithmes peuvent donc être utilisés en ligne sauf l'algorithme exact qui prend plus d'une demi-heure.

Du point de vue complexité, H-COPS et OPA s'exécutent en $O(|O|\log|O|)$ car ils sont basés sur un algorithme glouton. DOT s'exécute en $O(G2^K)$, sachant que G et K sont liés à certains paramètres d'optimisations spécifiques pour DOT [132].

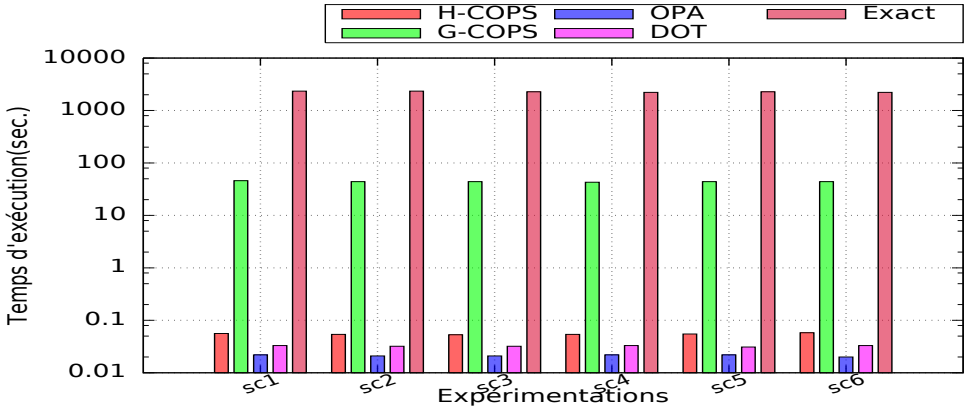

FIGURE 5.10 : Temps d'exécution pour les petites instances du problème

5.6.6 *Les problèmes de large instance*

5.6.6.1 *Méthodologie de validation*

Contrairement aux petites instances de problème dans lesquelles nous pouvons calculer la solution optimale dans un temps raisonnable avec l'algorithme exact, il est impossible de le faire pour de larges instances du problème (voir figure 5.5). Pour pallier à ce problème, nous proposons de générer des instances synthétiques du problème dans lesquelles nous connaissons au préalable la solution optimale par construction (sans calcul).

Nous construirons une large instance du problème en considérant que tous les clients sont identiques. Par conséquent, trouver le placement optimal revient à le trouver pour un seul client. Donc, nous obtenons la solution de placement optimale du problème en appliquant la même solution pour tous les clients, sous la condition que les contraintes (stockage et performance) soient vérifiées.

Dans cette partie d'évaluation, nous avons utilisé le scénario "sc2" illustré dans le tableau 5.4. Nous avons mené trois expériences pour les benchmarks TPC-C et TPC-H, dans lesquelles nous avons varié le nombre de clients. Nous avons expérimenté avec 10, 25 et 50 clients, en suivant la même méthodologie de validation que celle donnée dans la section 5.6.5 de ce chapitre.

Nous avons comparé les placements générés avec les algorithmes G-COPS, H-COPS, DOT et OPA avec le placement optimal. Ce dernier est construit en appliquant le même placement pour tous les clients.

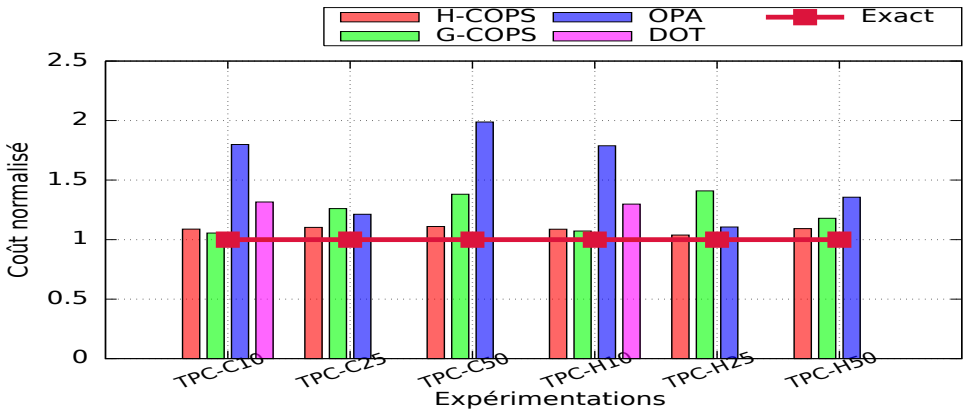

FIGURE 5.11 : Taux d'erreur pour les grandes instances du problème

5.6.6.2 *Résultats d'expérience*

La figure 5.11 montre les résultats de nos expérimentations. Contrairement aux petites instances du problème, nous observons que le H-COPS est plus précis que G-COPS dans la plupart des cas, avec un taux d'erreur de 10 % (23 % pour G-COPS).

L'algorithme H-COPS est le mieux adapté pour cette catégorie de problèmes. OPA a montré un taux d'erreur de 54% en moyenne alors que le DOT s'exécute dans un temps non raisonnable (plus de deux heures) ce qui le disqualifie pour être utilisé en temps réel (runtime).

Concernant le temps d'exécution des algorithmes, nous observons dans la figure 5.12 que H-COPS et OPA présentent les meilleurs résultats en ce qui concerne le temps d'exécution. Les deux ne dépassent pas une seconde (0,5 seconde pour 50 clients), alors que G-COPS atteint 5 minutes dans le pire des cas (pour 50 clients). Néanmoins, G-COPS optimise le coût de 17% en moyenne par rapport à l'OPA . Il faut noter que 5 minutes restent un temps acceptable. Nous avons remarqué que le temps d'exécution de tous les algorithmes augmente linéairement avec le nombre de clients.

En résumant, H-COPS présente les meilleurs résultats par rapport aux autres algorithmes testés. Le coût fourni dépasse le coût optimal à 11% seulement et l'algorithme s'exécute en moins de 5 minutes.

5.7 CONCLUSION

Dans ce chapitre nous avons étudié le problème de placement des objets dans un système de stockage hybride, et cela dans un environnement de Cloud. Notre réflexion principale est que les approches destinées pour le Cloud doivent être

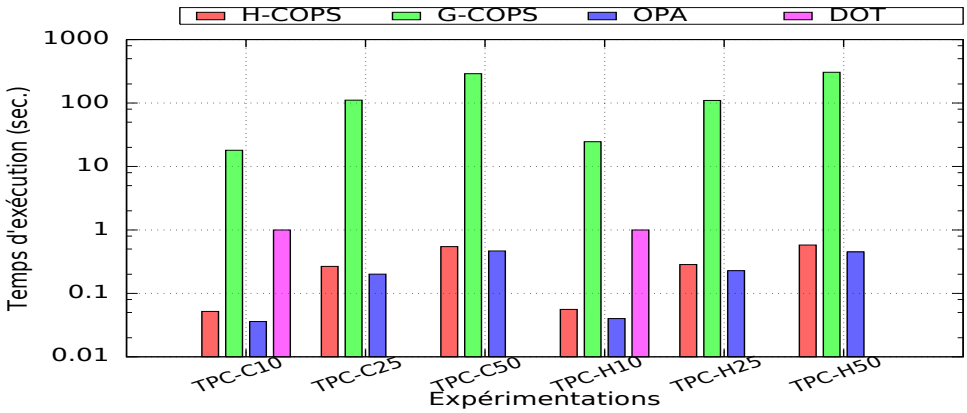

FIGURE 5.12 : Temps d'exécution pour les grandes instances du problème

orientées coût et non pas performance.

Nous avons proposé deux approches de placement : G-COPS basée sur un algorithme génétique et H-COPS basée sur un algorithme heuristique ad-hoc. Contrairement aux travaux connexes, nos approches prennent en compte les concepts clés du Cloud tels que le SLA, la multi-locataire et les pénalités.

Nous avons évalué et testé les deux algorithmes en utilisant les benchmarks TPC-C et TPC-H. Les résultats expérimentaux montrent que G-COPS fournit un placement proche à l'optimal dans un temps raisonnable pour les problèmes de petite taille. Le taux d'erreur en termes de coût par rapport à l'optimum ne dépassant pas 2%. H-COPS est plus efficace pour les problèmes larges. Il présente un taux d'erreur moyen de 10 % et un temps d'exécution de moins de 5 minutes pour les problèmes de grande taille. H-COPS converge vers le SLA des clients avec un sur-approvisionnement qui ne dépasse pas 11 % (8 % en moyenne).

Troisième partie

CONCLUSION

Chapitre 6

CONCLUSION

Dans l'optique de maximiser leurs revenus et de proposer des prix compétitifs pour les clients, les fournisseurs des services cloud cherchent à minimiser le coût total (acquisition et exploitation) de leur infrastructure. L'optimisation des coûts relatifs au système de stockage représente un sérieux problème depuis de nombreuses années [54, 71, 125]. Le problème se complique d'autant lorsqu'il s'agit d'un système de stockage complexe qui combine plusieurs classes de stockage présentant des performances différentes et des coûts différents en termes de Go/$.

Actuellement, la majorité des centres de données utilisent un système de stockage hybride qui combine des disques HDD et des disques SSD [132]. En effet, les disques HDD sont des supports de stockage à la fois énergivores et peu performants comparés aux unités de calcul. Néanmoins, leur prix par gigaoctet et leur longévité peuvent jouer en leur faveur. Contrairement aux HDD, les disques SSD à base de mémoire flash sont plus performants et consomment peu d'énergie. Leur prix élevé par gigaoctet et leur courte durée de vie (comparés aux HDD) représentent leurs contraintes majeures.

Dans cette thèse, nous proposons un ensemble de propositions visant à optimiser le placement de données dans un système de stockage hybride. Malgré la panoplie des travaux de l'état de l'art qui proposent des approches de modélisation et d'optimisation des coûts relatifs au système de stockage [33, 84, 125, 126, 132], ils ne prennent pas en compte simultanément les contraintes liées à l'environnement cloud (i.e. SLA et pénalité, migration), et aux périphériques de stockage (i.e. consommation énergétique, usure du périphérique). Dans ce chapitre, nous commençons par un rappel de nos principales contributions dans le cadre de cette thèse. Nous détaillons ensuite quelques ouvertures de recherche et perspectives liées à nos travaux.

Sommaire

6.1 CONTRIBUTIONS

Nous avons apporté deux contributions majeurs dans le cadre de cette thèse. La première consiste a proposé un modèle pour l'évaluation du coût de placement des objets DBAAS dans un système de stockage hybride. La deuxième est consacrée aux stratégies de placement des données. Les deux contributions prennent en compte les contraintes économiques de l'environnement cloud (i.e. SLA et pénalité, migration).

6.1.1 *Modèle pour l'évaluation du coût de placement dans un environnement de Cloud*

Notre première contribution consiste en la proposition d'un modèle pour évaluer le coût de placement des données dans un système de stockage hybride qui combine deux classes de stockage (disque dur magnétique et disque SSD). Le modèle de coût proposé inclut tous les coûts fixes et variables, directs et indirects induits par le stockage des données (les objets des différentes bases de données dans notre cas d'étude) sur les périphériques de stockage. Il prend en compte différents paramètres modélisant les caractéristiques physiques et fonctionnelles du système de stockage. Nous considérons également les contraintes liées au service cloud, comme l'environnement virtualisé, l'architecture multi-locataires et les besoins des clients en termes de qualité de service spécifiées dans le SLA. Le modèle proposé introduit pour la première fois, à notre connaissance et au moment de la rédaction du manuscrit, l'impact de l'ordre de migration sur le coût de placement global.

L'évaluation du modèle de coût proposé a été menée sur une plateforme réelle. Le système de stockage utilisé se compose d'un disque dur HDD et d'un autre disque SSD. Nous avons utilisé TPC-H [19] et TPC-C [20] comme benchmarks dans notre validation.

Nous avons comparé notre modèle de coût avec les approches proposées dans [78, 97, 132]. Les expérimentations montrent que notre modèle de coût évalue précisément et dans un temps raisonnable le coût de placement en comparant avec les travaux connexes. Cela lui permet d'être utilisé pour une évaluation en temps réel.

6.1.2 *Stratégie de placement des données*

Notre deuxième contribution consiste à proposer une stratégie de placement des données tout en considérant les aspects liés au Cloud qu'ils soient économiques ou techniques. En effet, nous avons formulé le problème de placement par un problème d'optimisation sous un certain nombre de contraintes qui sont liées au stockage, à la performance et aux SLA des clients. La fonction objectif est construite autour de notre modèle de coût. Nous avons conçu et implanté deux d'algorithmes d'optimisation de placement des objets. Ces algorithmes ont comme objectif la minimisation de coût de placement.

La première approche est basée sur un algorithme génétique. Un travail de calibration est mené pour définir les bons paramètres d'optimisation (la taille de la population initiale, la condition d'arrêt, la prise ou non des solutions infaisables, etc.). Comme les méta-heuristiques ne sont pas toujours la bonne solution, nous avons proposé une deuxième approche heuristique visant à exploiter les avantages du système de stockage hybride dans le but de minimiser les coûts (i.e. consommation énergétique, pénalités, et usure des périphériques de stockage). Nous avons mis en œuvre aussi une approche exacte qui énumère tous les plans de placement des objets puis choisit le plan le moins coûteux. La méthode exacte a été utilisée comme une référence afin d'analyser la précision des approches proposées.

Nos approches ont été implantées et évaluées dans un environnement réel en utilisant les benchmarks TPC-H et TPC-C. Nous avons montré la pertinence de nos algorithmes pour les problèmes de petites et de larges instances. Nos approches ont été comparées avec les travaux proposés dans [33, 132].

En termes de précision, nos approches ont donné des solutions proches du plan de placement optimal (10% plus coûteux que la solution optimale dans les pires situations), tandis que les travaux connexes donnent des plans de placement des objets qui sont 40% plus coûteux que la solution optimale.

En termes de temps d'exécution, nos approches s'exécutent dans un temps raisonnable (quelques secondes) dans les pires cas. Ceci est important pour la mise en œuvre en temps réel de ces solutions.

En termes de satisfaction des SLAs, nos approches proposent des placements qui satisfont les exigences des clients mais en restant juste au-dessus du débit (IOPS) demandé afin d'éviter le sur-approvisionnement. Notons que le sur-approvisionnement gaspille les ressources et engendre des coûts supplémen-

taires pour le fournisseur du Cloud.

6.2 PERSPECTIVES

Les contributions abordées dans cette thèse s'appliquent au problème de placement des objets du service DBaaS dans un système de stockage hybride. Dans les sections qui suivent, nous discutons les perspectives ouvertes par ces contributions. Nous classons ces perspectives en trois grands axes à savoir : (1) amélioration des contributions proposées : regroupe les travaux qui visent à améliorer et corriger les limites constatées dans l'évaluation des contributions proposées, (2) extension du périmètre d'application : regroupe les perspectives qui ciblent d'autres domaines d'application au-delà des bases de données relationnelles et (3) mise en application des contributions : regroupe les travaux d'intégration des approches proposés dans des simulateurs/émulateurs de Cloud.

6.2.1 *Amélioration des contributions proposées*

Dans cette thèse, nous avons proposé une approche suivant le modèle MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) [69] pour le placement autonomique des objets de DBaaS sur un système de stockage hybride. Nos perspectives seront présentées suivant ce modèle.

6.2.1.1 *Monitor*

L'outil de surveillance (tracer) proposé pour l'étape *Monitor* ne permet pas la prédiction des charges d'E/S transmises aux objets. En effet, nous considérons actuellement que la charge de travail transmise à un objet pendant la période T est égale à celle transmise pendant la période $T + 1$ et que la charge de travail ne peut changer avant une nouvelle période de *monitoring*.

Afin d'éviter un *monitoring* permanent, nous planifions d'étudier des modèles de prédictions des activités d'E/S des objets. Ces modèles profiteront de l'historique de trace et de caractérisation des charges d'E/S. Pour ce faire, plusieurs techniques peuvent être explorées (e.g. modèles de Markov [70], algorithmes d'apprentissage automatique [102], réseaux de neurones [83]).

6.2.1.2 *Analyze*

Notre modèle de coût actuel a été conçu pour les périphériques des stockage sans prise en compte des caches du système d'exploitation et du SGBD. Nous envisageons d'étudier l'intégration d'autres contraintes dans notre modèle (e.g.

mécanismes de cache, hétérogénéité des périphériques du même classe de stockage, etc), et de modéliser l'impact des E/S sur l'ensemble du système (CPU et RAM).

Le modèle de coût proposé considère la diversité des périphériques de stockage mais nous avons pris comme un cas d'étude un système de stockage hybride simple qui combine deux classes de stockage HDD et SSD. Une des perspectives est de généraliser l'utilisation de ce dernier pour des cas beaucoup plus complexes, par exemple l'utilisation d'autres classes de stockage comme les NVM.

6.2.1.3 *Plan*

Pour le placement des objets, nous avons proposé une méthode exacte qui consiste à énumérer toutes les solutions. Nous aimerions étudier les limites d'autres approches exactes, telles que la programmation linéaire et en nombre entier.

Pour les méthodes approchées, nous avons proposé deux heuristiques. Nous envisageons d'élargir notre étude sur d'autres approches multi-objectifs (e.g. recuit simulé, colonies de fourmis). Nous envisageons aussi de modéliser le problème sous forme d'une optimisation multi-objectifs afin de minimiser les coûts et maximiser les performances.

6.2.1.4 *Execute*

L'étapes *Execute* n'a pas été intégrée dans nos approches de placement autonomique des objets. Afin de compléter notre version du modèle MAPE-K, nous envisageons étudier l'étape *Execute*. Cette étape doit se charger d'appliquer les plans de placement des objets sur l'infrastructure du *cloud*. Pour ce faire, une étude sur l'ordonnancement des migrations des objets doit être menée [120].

6.2.1.5 *Knowledge*

Nous envisageons aussi d'intégrer l'étape *Knowledge* qui consiste à implémenter une base de connaissance qui détient toute la connaissance sur les profils d'accès au objets et l'historique des plans de placement précédents afin de les exploiter par la suite dans les autres étapes. Une des applications de la base de connaissances est d'analyser l'historique des accès afin de prédire la charge de travail de la période $T + 1$.

6.2.2 *Extension du périmètre d'application*

6.2.2.1 *Bases de données NoSQL*

Nous avons pris les bases de données relationnelles comme un cas d'étude. Nous envisageons de profiter des connaissances acquises lors de la réalisation de ce travail afin d'étendre nos travaux pour les bases de données NoSQL. Nous estimons qu'aucune modification (adaptation) ne sera apportée sur les approches proposées car aucune hypothèse n'est faite sur la nature du SGBD (relationnel ou NoSQL). La seule révision réside dans l'adaptation de la granularité de placement. Il faut noter que les objets des bases de données NoSql sont, généralement, volumineux et l'utilisation d'une granularité très large et potentiellement imprécise (c'est à dire différents motifs d'accès sont appliqués dans différents endroits ce qui rend difficile de sélectionner une classe de stockage pour l'objet). L'adaptation de la granularité dépend de type de SGBD. S'il s'agit d'une SGBD document comme Mangodb, le niveau de granularité pourra passer au niveau de la collection, document ou sous document tandis que pour les SGBD orientés colonne, une granularité au niveau d'une colonne ou une famille de colonne pourra être utilisée. Pour un SGBD orienté graphe, la granularité pourra être au niveau d'un graphe, un sous graphe ou un nœud.

6.2.2.2 *Généralisation des approches proposées pour d'autres types d'application hors les SGBD*

Un des travaux envisagés est de généraliser l'utilisation des approches proposées pour d'autres types d'application. Comme annoncé ci-dessus, aucune hypothèse n'est faite sur la nature d'application. La généralisation de l'utilisation de ces méthodes consiste à trouver une granularité générique qui s'adapte avec le type d'application afin d'avoir des résultats précis.

6.2.3 *Mise en application des approches proposées*

Un des travaux envisagé consiste à intégrer nos approches dans le simulateur *CloudSim* [32]. CloudSim est un simulateur de cloud IaaS fréquemment utilisé pour l'implantation et l'expérimentation de méthodes d'optimisation [32]. Il est le plus utilisé dans l'état de l'art [86]. Cette popularité vient de la simplicité de ses fonctions permettant l'implantation et l'évaluation des algorithmes de consolidation. L'inconvénient majeur de CloudSim réside dans sa gestion élémentaire des E/S et des systèmes de stockage. De plus, les coûts et l'utilisation CPU liés à l'exécution des E/S ne sont pas considérés par CloudSim. Nous prévoyons d'intégrer nos contributions sur CloudSim pour la prise en compte des E/S et des

systèmes de stockage

Afin d'intégrer nos approches, nous devons d'abord faire une extension de CloudSim pour qu'il considère la variété des périphériques de stockage (i.e. HDD et SSD), ainsi que le calcul du temps et de la consommation énergétique liée à l'exécution des charges d'E/S.

[1] Amazon Elastic Block Store (Amazon EBS) – Pricing – Amazon Web Services (AWS). [Online]. Available https://aws.amazon.com/ebs/pricing/. Accessed :29-10-2017.

[2] Amazon elastic file system – shared file storage for amazon ec2. [Online]. Available https://aws.amazon.com/fr/blogs/aws/amazon-elastic-file-system-shared-file-storage-for-amazon-ec2/, . Accessed :04-04-2016.

[3] Aws | amazon relational database service (rds). [Online]. Available http://aws.amazon.com/rds/, . Accessed :17-07-2016.

[4] Amazon compute service level agreement. [Online]. Available https://aws.amazon.com/compute/sla/?nc1=h_ls, . Accessed :10-04-2019.

[5] Google : From 112 servers to a $5b-plus quarterly data center bill. [Online]. Available http://www.datacenterknowledge.com/archives/2014/07/23/from-112-servers-to-5b-spent-on-google-data-centers-per-quarter/, . Accessed :13-11-2016.

[6] Compute engine. [Online]. Available https://cloud.google.com/compute/, . Accessed :05-03-2019.

[7] Ibm 350 disk storage unit. [Online]. Available http://www-03.ibm.com/ibm/history/exhibits/storage/storage_350.html. Accessed :27-05-2016.

[8] Apache CouchDB. [Online]. Available http://couchdb.apache.org/, . Accessed :22-03-2019.

[9] Cloud computing : les nouveaux modèles économiques. [Online]. Available https://syntec-numerique.fr/cloud-computing, . Accessed :03-09-2018.

[10] ESXi | bare metal hypervisor | VMware. [Online]. Available https://www.vmware.com/products/esxi-and-esx.html, . Accessed :04-09-2018.

[11] Fusion makes running windows on a mac easy. [Online]. Available https://www.vmware.com/products/fusion.html, . Accessed :04-09-2018.

[12] Hyperviseur citrix XenServer. [Online]. Available http://www.servitel-cm.com/produits/systemes-informations/Hyperviseur-Citrix-XenServer.php, . Accessed :04-09-2018.

[13] IBM archives : IBM 350 disk storage unit. [Online]. Available https://www-03.ibm.com/ibm/history/exhibits/storage/storage_350.html, . Accessed :17-09-2018.

[14] Introduction to hyper-v on windows 10 | microsoft docs. [Online]. Available https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/, . Accessed :04-09-2018.

[15] Neo4j graph platform – the leader in graph databases. [Online]. Available https://neo4j.com/, . Accessed :22-03-2019.

[16] Oracle VM VirtualBox. [Online]. Available https://www.virtualbox.org/, . Accessed :04-09-2018.

[17] Qemu — lea linux. [Online]. Available http://lea-linux.org/documentations/Software-soft_emul-qemu, . Accessed :04-09-2018.

[18] Riak KV. [Online]. Available https://docs.riak.com/riak/kv/2.2.3/index.html, . Accessed :16-08-2019.

[19] TPC-h - homepage. [Online]. Available http://www.tpc.org/tpch/, . Accessed :13-07-2018.

[20] TPC-c - homepage. [Online]. Available http://www.tpc.org/, . Accessed :13-07-2018.

[21] Windows VM | VMware workstation pro. [Online]. Available https://www.vmware.com/products/workstation-pro.html, . Accessed :04-09-2018.

[22] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *USENIX 2008 Annual Technical Conference*, pages 57–70. USENIX Association, Juin 2008.

[23] M. Alouane and H. El Bakkali. Virtualization in cloud computing : NoHype vs HyperWall new approach. In *The International Conference on Electrical and Information Technologies (ICEIT)*, pages 49–54. IEEE, Mai 2016.

[24] R. Appuswamy, D. C. van Moolenbroek, and A. S. Tanenbaum. Integrating flash-based SSDs into the storage stack. In *The Symposium on Mass Storage Systems and Technologies*, pages 1–12. IEEE, Avril 2012.

[25] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4) :50–58, Avril 2010. doi : 10.1145/1721654.1721672.

[26] E. Baralis, A. Dalla Valle, P. Garza, C. Rossi, and F. Scullino. SQL versus NoSQL databases for geospatial applications. In *The International Conference on Big Data (Big Data)*, pages 3388–3397. IEEE, Decembre 2017.

[27] D. Boukhelef, K. Boukhalfa, J. Boukhobza, H. Ouarnoughi, and L. Lemarchand. COPS : Cost Based Object Placement Strategies on Hybrid Storage System for DBaaS Cloud. In *The International Symposium on Cluster, Cloud and Grid Computing*, pages 659–664. IEEE, Mai 2017.

[28] Djillali Boukhelef, Jalil Boukhobza, and Kamel Boukhalfa. A cost model for DBaaS storage. In *The International Conference on Database and Expert Systems Applications*, pages 223–239. Springer International Publishing, septembre 2016.

[29] 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, Avril 2019. doi : 10.1016/j.future.2018.10.030.

[30] Jalil Boukhobza. Flashing in the cloud : Shedding some light on NAND flash memory storage systems. *Data Intensive Storage Services for Cloud Environments*, 1 :241–266, Septembre 2013. doi : 10.4018/978-1-4666-3934-8.ch015.

[31] Jalil Boukhobza and Pierre Olivier. *Flash Memory Integration*. ISTE Press - Elsevier, 2017. ISBN 9780081011584.

[32] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. Cloudsim : A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software—Practice & Experience*, 41(1) :23–50, Janvier 2011. doi : 10.1002/spe.995.

[33] Mustafa Canim, George A. Mihaila, Bishwaranjan Bhattacharjee, Kenneth A. Ross, and Christian A. Lang. An Object Placement Advisor for DB2 Using Solid State Storage. *Proceedings of the VLDB Endowment*, 2(2) : 1318–1329, Août 2009. doi : 10.14778/1687553.1687557.

[34] Feng Chen, David A. Koufaty, and Xiaodong Zhang. Hystor : Making the Best Use of Solid State Drives in High Performance Storage Systems. In *The International Conference on Supercomputing*, pages 22–32. ACM, Mai 2011.

[35] Yue Cheng, M. Safdar Iqbal, Aayush Gupta, and Ali R. Butt. Pricing games for hybrid object stores in the cloud : Provider vs. tenant. In *USENIX Conference on Hot Topics in Cloud Computing*, pages 20–20. USENIX Association, Juillet 2015.

[36] Carlo Curino, Evan PC Jones, Raluca Ada Popa, Nirmesh Malviya, Eugene Wu, Sam Madden, Hari Balakrishnan, and Nickolai Zeldovich. Relational cloud : A database-as-a-service for the cloud. In *Innovative Data Systems Research, Conference Proceedings*, pages 235–240. Mit, Janvier 2011.

[37] Carlo A. Curino, Djellel E. Difallah, Andrew Pavlo, and Philippe Cudre-Mauroux. Benchmarking OLTP/web databases in the cloud : The OLTP-bench framework. In *The International Workshop on Cloud Data Management*, pages 17–20. ACM, Octobre 2012.

[38] Sudipto Das, Vivek R. Narasayya, Feng Li, and Manoj Syamala. CPU sharing techniques for performance isolation in multi-tenant relational database-as-a-service. *Proceedings of the VLDB Endowment*, 7(1) :37–48, Septembre 2013. doi : 10.14778/2732219.2732223.

[39] Datasheet. Samsung ssd 840 pro series. Technical report, Samsung, Janvier 2013. URL http://americas.as.techdata.com/na/en-us/suppliers/GCC/Documents/Samsung/Samsung_SSD_840_PRO_Brochure.pdf.

[40] Datasheet. Samsung ssd 850 pro. Technical report, Samsung, Janvier 2017. URL http://downloadcenter.samsung.com/content/UM/201711/20171115103115156/Samsung_SSD_850_PRO_Data_Sheet_Rev_3.pdf.

[41] Peter Desnoyers. Empirical evaluation of nand flash memory performance. *ACM SIGOPS Operating Systems Review*, 44(1) :50–54, Mars 2010. doi : 10.1145/1740390.1740402.

[42] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. OLTP-bench : An extensible testbed for benchmarking relational databases. *Proceedings of the VLDB Endowment*, 7(4) :277–288, Decembre 2013. doi : 10.14778/2732240.2732246.

[43] Amit Kumar Dutta and Ragib Hasan. How Much Does Storage Really Cost? Towards a Full Cost Accounting Model for Data Storage. In *Economics of Grids, Clouds, Systems, and Services*, pages 29–43. Springer, Septembre 2013.

[44] Afaf Edinat. Cloud computing pricing models : A survey. *The International Journal of Scientific and Engineering Research*, 6 :22–26, Janvier 2019. doi : 10.14257/ijgdc.2013.6.5.09.

[45] I. Eisa, R. Salem, and H. Abdelkader. A fragmentation algorithm for storage management in cloud database environment. In *The International Conference on Computer Engineering and Systems*, pages 141–147. IEEE, Decembre 2017.

[46] Aaron J. Elmore, Carlo Curino, Divyakant Agrawal, and Amr El Abbadi. Towards database virtualization for database as a service. *Proceedings of the VLDB Endowment*, 6(11) :1194–1195, Août 2013. doi : 10.14778/2536222.2536256.

[47] Victor A. E. Farias, Flávio R. C. Sousa, José Gilvan R. Maia, João Paulo P. Gomes, and Javam C. Machado. Regression based performance modeling and provisioning for NoSQL cloud databases. *Future Generation Computer Systems*, 79(1) :72–81, Février 2018. doi : 10.1016/j.future.2017.08.061.

[48] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop*, pages 1–10. IEEE, Novembre 2008.

[49] Saurabh Kumar Garg, Srinivasa K. Gopalaiyengar, and Rajkumar Buyya. Sla-based resource provisioning for heterogeneous workloads in a virtualized cloud datacenter. In *The International Conference on Algorithms and Architectures for Parallel Processing*, pages 371–384. Springer-Verlag, Octobre 2011.

[50] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, Guillaume Belrose, Tom Turicchi, and Alfons Kemper. An integrated approach to resource pool management : Policies, efficiency and quality metrics. In *The International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 326–335. IEEE, Juin 2008.

[51] Daniel Gmach, Jerry Rolia, and Ludmila Cherkasova. Resource and virtualization costs up in the cloud : Models and design choices. In *The International Conference on Dependable Systems&Networks*, pages 395–402. IEEE, Juin 2011.

[52] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201157675.

[53] Aaron Grattafiori. Understanding and hardening linux containers. Technical report, NCC Group, Juin 2016. URL https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2016/april/ncc_group_understanding_hardening_linux_containers-1-1.pdf.

[54] Jorge Guerra, Himabindu Pucha, Joseph Glider, Wendy Belluomini, and Raju Rangaswami. Cost effective storage using extent based dynamic tiering. In *USENIX Conference on File and Stroage Technologies*, pages 20–20. USENIX Association, Février 2011.

[55] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM : dynamic speed control for power management in server class disks. In *The International Symposium on Computer Architecture*, pages 169–179. IEEE, Juin 2003.

[56] S. He, Y. Wang, Z. Li, X. H. Sun, and C. Xu. Cost-aware region-level data placement in multi-tiered parallel i/o systems. *IEEE Transactions on Parallel and Distributed Systems*, 28(7) :1853–1865, Juillet 2007. doi : 10.1109/TPDS.2016.2636837.

[57] S. He, X. H. Sun, B. Feng, X. Huang, and K. Feng. A cost-aware region-level data placement scheme for hybrid parallel i/o systems. In *The International Conference on Cluster Computing*, pages 1–8. IEEE, Janvier 2013.

[58] S. He, Y. Liu, and X. H. Sun. PSA : A performance and space-aware data layout scheme for hybrid parallel file systems. In *The International Workshop on Data Intensive Scalable Computing Systems*, pages 41–48. IEEE, Novembre 2014.

[59] S. He, X. H. Sun, Y. Wang, A. Kougkas, and A. Haider. A heterogeneity-aware region-level data layout for hybrid parallel file systems. In *The International Conference on Parallel Processing*, pages 340–349. IEEE, Septembre 2015.

[60] Shuibing He, Xian-He Sun, Bo Feng, and Kun Feng. Performance-aware data placement in hybrid parallel file systems. In *Algorithms and Architectures for Parallel Processing*, pages 563–576. Springer, Cham, Août 2014.

[61] A. Hylick, R. Sohan, A. Rice, and B. Jones. An analysis of hard drive energy consumption. In *The International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*, pages 1–10. IEEE, Septembre 2008.

[62] A. A. Z. A. Ibrahim, S. Varrette, and P. Bouvry. PRESENCE : Toward a novel approach for performance evaluation of mobile cloud SaaS web services. In *The International Conference on Information Networking (ICOIN)*, pages 50–55. IEEE, Janvier 2018.

[63] N. Jain. The future of database services : Cloud database. In *The International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies*, pages 1–5. IEEE, Février 2017.

[64] N. Jain and S. Choudhary. Overview of virtualization in cloud computing. In *Symposium on Colossal Data Analysis and Networking*, pages 1–4. IEEE, Mars 2016.

[65] L Harrington Jan. *Relational Database Design and Implementation*. Morgan Kaufmann, 2010. ISBN 0128043997.

[66] Zhiwen Jiang, Yong Zhang, Jin Wang, and Chunxiao Xing. A Cost-aware Buffer Management Policy for Flash-based Storage Devices. In *The International Conference on Database Systems for Advanced Applications*, pages 175–190. Springer, Cham, Avril 2015.

[67] Heeseung Jo, Youngjin Kwon, Hwanju Kim, Euiseong Seo, Joonwon Lee, and Seungryoul Maeng. Ssd-hdd-hybrid virtual disk in consolidated environments. In *The European Conference on Parallel Processing*, pages 375–384. Springer, Août 2009.

[68] Woon-Hak Kang, Sang-Won Lee, and Bongki Moon. Flash as cache extension for online transactional workloads. *The VLDB Journal*, 25(5) :1–22, Octobre 2015. doi : 10.1007/s00778-015-0414-1.

[69] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1) :41–50, Janvier 2003. doi : 10.1109/MC.2003.1160055.

[70] Arijit Khan, Xifeng Yan, Shu Tao, and Nikos Anerousis. Workload characterization and prediction in the cloud : A multiple time series approach. In *Network Operations and Management Symposium*, pages 1287–1294. IEEE, Avril 2012.

[71] Youngjae Kim, Aayush Gupta, Bhuvan Urgaonkar, Piotr Berman, and Anand Sivasubramani. HybridStore : A cost-efficient, high-performance storage system combining ssds and hdds. In *The Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 227–236. IEEE, Août 2011.

[72] Youngjae Kim, Aayush Gupta, Bhuvan Urgaonkar, Piotr Berman, and Anand Sivasubramaniam. HybridPlan : a capacity planning technique for projecting storage requirements in hybrid storage systems. *The Journal of Supercomputing*, 67(1) :277–303, Janvier 2013. doi : s11227-013-0999-3.

[73] K. R. Krish, A. Anwar, and A. R. Butt. hatS : A Heterogeneity-Aware Tiered Storage for Hadoop. In *The International Symposium on Cluster, Cloud and Grid Computing*, pages 502–511. IEEE, Juillet 2014.

[74] D. Kusnetzky. *Virtualization : A Manager's Guide*. O'Reilly Media, Incorporated, 2011. ISBN 9781449306458.

[75] Arezki Laga. *Optimisation des performance des logiciels de traitement de donnees sur les peripheriques de stockage SSD*. phdthesis, Université de Bretagne occidentale, Decembre 2018.

[76] Willis Lang, Frank Bertsch, David J. DeWitt, and Nigel Ellis. Microsoft Azure SQL Database Telemetry. In *The ACM Symposium on Cloud Computing*, pages 189–194. ACM, Août 2015.

[77] D. Lee, C. Min, and Y. I. Eom. Effective flash-based SSD caching for high performance home cloud server. *IEEE Transactions on Consumer Electronics*, 61(2) :215–221, Mai 2015. doi : 10.1109/TCE.2015.7150596.

[78] Zhichao Li, Amanpreet Mukker, and Erez Zadok. On the importance of evaluating storage systems' $costs. In *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems*, pages 6–6. USENIX Association, Juin 2014.

[79] L. Lin, Y. Zhu, J. Yue, Z. Cai, and B. Segee. Hot Random Off-Loading : A Hybrid Storage System with Dynamic Data Migration. In *The International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems*, pages 318–325. IEEE, Juillet 2011.

[80] Xin Liu and Kenneth Salem. Hybrid Storage Management for Database Systems. *Proceedings of the VLDB Endowment*, 6(8) :541–552, Juin 2013. doi : 10.14778/2536354.2536355.

[81] Ziyang Liu, Hakan Hacıgümüş, Hyun Jin Moon, Yun Chi, and Wang-Pin Hsiung. PMAX : Tenant placement in multitenant databases for profit maximization. In *International Conference on Extending Database Technology*, pages 442–453. ACM, Mars 2013.

[82] Chenyang Lu, Guillermo A. Alvarez, and John Wilkes. Aqueduct : Online data migration with performance guarantees. In *USENIX Conference on File and Storage Technologies*, pages 21–21. USENIX Association, Janvier 2002.

[83] Yao Lu, John Panneerselvam, Lu Liu, and Yan Wu. RVLBPNN : A workload forecasting model for smart cloud computing. *Scientific Programming*, 2016 :1–9, Novembre 2016. doi : 10.1155/2016/5635673.

[84] Tian Luo, Rubao Lee, Michael Mesnier, Feng Chen, and Xiaodong Zhang. hStorage-DB : Heterogeneity-aware Data Management to Exploit the Full Capability of Hybrid Storage Systems. *Proceedings of the VLDB Endowment*, 5(10) :1076–1087, Juin 2012. doi : 10.14778/2336664.2336679.

[85] A.A. Mamun, G.X. Guo, and C. Bi. *Hard Disk Drive : Mechatronics and Control*. CRC Press, 2006. ISBN 9781420004106.

[86] Zolt Mann. Allocation of virtual machines in cloud data centers : A survey of problem models and optimization algorithms. *ACM Computing Surveys*, 48(1) :1–34, Août 2015. doi : 10.1145/2797211.

[87] Toni Mastelic, Ariel Oleksiak, Holger Claussen, Ivona Brandic, Jean-Marc Pierson, and Athanasios V. Vasilakos. Cloud computing : Survey on energy efficiency. *ACM Computing Surveys*, 47(2) :1–36, Decembre 2014. doi : 10.1145/2656204.

[88] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, National Institute of Standards & Technology, Septembre 2011. URL https://csrc.nist.gov/publications/detail/sp/800-145/final.

[89] Hyun Jin Moon, Yun Chi, and Hakan Hacigümüş. Performance evaluation of scheduling algorithms for database services with soft and hard slas. In *The International Workshop on Data Intensive Computing in the Clouds*, pages 81–90. ACM, Novembre 2011.

[90] Richard L. Moore, Jim D'Aoust, Robert H. McDonald, and David Minor. Disk and tape storage cost models. In *Archiving Conference*, pages 29–32. Society for Imaging Science and Technology, Mai 2007.

[91] Vivek Narasayya, Sudipto Das, Manoj Syamala, Surajit Chaudhuri, Feng Li, and Hyunjung Park. A Demonstration of SQLVM : Performance Isolation in Multi-tenant Relational Database-as-a-service. In *The ACM SIGMOD International Conference on Management of Data*, pages 1077–1080. ACM, Juin 2013.

[92] Jiacai Ni, Guoliang Li, Jun Zhang, Lei Li, and Jianhua Feng. Adapt : Adaptive Database Schema Design for Multi-tenant Applications. In *The International Conference on Information and Knowledge Management*, pages 2199–2203. IEEE, Juin 2012.

[93] Jiacai Ni, Jianhua Feng, and Yongwei Wu. A customized schema design framework for multi-tenant database. In *The International Conference on Web-Age Information Management*, pages 530–534. Springer International Publishing, Juin 2015.

[94] Pierre Olivier. *Estimation de performances et de consommation energetique de systemes de stockage*. phdthesis, Université de Bretagne Sud, Decembre 2014.

[95] Hamza Ouarnoughi. *Placement autonomique de machines virtuelles sur un système de stockage hybride dans un cloud IaaS*. phdthesis, Université de Bretagne occidentale - Brest, Juillet 2017.

[96] Hamza Ouarnoughi, Jalil Boukhobza, Frank Singhoff, and Stephane Rubini. A multi-level I/O tracer for timing and performance storage systems in iaas cloud. In *The International Workshop on Real-time and distributed computing in emerging applications*, pages 1–1. IEEE, December 2014.

[97] Hamza Ouarnoughi, Jalil Boukhobza, Frank Singhoff, and Stephane Rubini. A Cost Model for Virtual Machine Storage in Cloud IaaS Context. In *Euromicro conference on Parallel, Distributed, and Network-Based Processing*, pages 664—-671. IEEE, Février 2016.

[98] Sanjukta Pal, Amit Kr Mandal, and Anirban Sarkar. Application Multi-Tenancy for Software As a Service. *ACM SIGSOFT Software Engineering Notes*, 40(2) :1–8, Mars 2015. doi : 10.1007/s00778-013-0334-x.

[99] Stan Park and Kai Shen. FIOS : a fair, efficient flash i/o scheduler. In *USENIX Conference on File and Storage Technologies*, page 13. USENIX, Février 2012.

[100] Sarada Prasanna Pati and Prasant Kumar Pattnaik. Criterions for databases in cloud computing environment. In *Intelligent Computing, Communication and Devices*, pages 385–392. Springer New Delhi, Août 2014.

[101] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. Failure trends in a large disk drive population. In *USENIX Conference on File and Storage Technologies*, pages 2–2. USENIX Association, Février 2007.

[102] Feng Qiu, Bin Zhang, and Jun Guo. A deep learning approach for vm workload prediction in the cloud. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 319–324. IEEE, Mai 2016.

[103] Michael K. Reiter. Side channels in multi-tenant environments. In *The Workshop on Cloud Computing Security Workshop*, pages 1–1. ACM, Octobre 2015.

[104] Detlev Richter. Fundamentals of reliability for flash memories. In *Flash Memories : Economic Principles of Performance, Cost and Reliability Optimization*, pages 149–166. Springer Netherlands, Septembre 2014.

[105] Paul Ruth, Anirban Mandal, Claris Castillo, Robert Fowler, Jeff Tilson, Ilya Baldin, and Yufeng Xin. Achieving Performance Isolation on Multi-Tenant Networked Clouds Using Advanced Block Storage Mechanisms. In *The Workshop on Scientific Cloud Computing*, pages 29–32. ACM, Juin 2015.

[106] Reza Salkhordeh, Hossein Asadi, and Shahriar Ebrahimi. Operating system level data tiering using online workload characterization. *The Journal of Supercomputing*, 71(4) :1534–1562, Avril 2015. doi : 10.1007/s11227-015-1377-0.

[107] Oliver Schiller, Nazario Cipriani, and Bernhard Mitschang. ProRea : Live Database Migration for Multi-tenant RDBMS with Snapshot Isolation. In

The International Conference on Extending Database Technology, pages 53–64. ACM, Mars 2013.

[108] Jiri Schindler. Profiling and analyzing the i/o performance of NoSQL DBs. In *The International Conference on Measurement and Modeling of Computer Systems*, pages 389–390. ACM, Juin 2013.

[109] Bhanu Sharma, Ruppa K. Thulasiram, Parimala Thulasiraman, Saurabh K. Garg, and Rajkumar Buyya. Pricing cloud compute commodities : A novel financial economic model. In *The International Symposium on Cluster, Cloud and Grid Computing*, pages 451–457. IEEE, Mai 2012.

[110] Gitanjali Sharma and Pankaj Deep Kaur. Architecting solutions for scalable databases in cloud. In *The International Symposium on Women in Computing and Informatics*, pages 469–476. ACM, Août 2015.

[111] Elizabeth Shriver. *Performance Modeling for Realistic Storage Devices*. phdthesis, New York University, Janvier 1997.

[112] Sukhpal Singh and Inderveer Chana. A survey on resource scheduling in cloud computing : Issues and challenges. *Journal of Grid Computing*, 14(2) : 217–264, Février 2016. doi : 10.1007/s10723-015-9359-2.

[113] S. W. Son, G. Chen, and M. Kandemir. Disk layout optimization for reducing energy consumption. In *The International Conference on Supercomputing*, pages 274–283. ACM, Juin 2005.

[114] H. Song, Y. Yin, X. H. Sun, R. Thakur, and S. Lang. A segment-level adaptive data layout scheme for improved load balance in parallel file systems. In *The International Symposium on Cluster, Cloud and Grid Computing*, pages 414–423. IEEE, Mai 2011.

[115] V. Sundaram, T. Wood, and P. Shenoy. Efficient Data Migration in Self-managing Storage Systems. In *The International Conference on Autonomic Computing*, pages 297–300. IEEE, Juin 2006.

[116] Jun Suzuki, Shivaram Venkataraman, Sameer Agarwal, Michael Franklin, and Ion Stoica. Record Placement Based on Data Skew Using Solid State Drives. In *Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, pages 181–193. Springer, Mars 2014.

[117] J. Tai, B. Sheng, Y. Yao, and N. Mi. Live Data Migration for Reducing SLA Violations in Multi-tiered Storage Systems. In *The International Conference on Cloud Engineering*, pages 361–366. IEEE, Mars 2014.

[118] J Tai, Bo Sheng, and Yi Yao. Sla-aware data migration in a shared hybrid storage cluster. *Cluster Computing*, 18(4) :1581–1593, Decembre 2015. doi : 10.1007/s10586-015-0461-9.

[119] Arie Tal. Two flash technologies compared : Nor vs nand. Technical report, White Paper of M-SYstems, Novembre 2002. URL https://www.csd.uoc.gr/~hy428/reading/M-Systems_NANDvsNOR.pdf.

[120] Hana Teyeb, Ali Balma, Samir Tata, and Nejib Ben Hadj-Alouane. Traffic-aware virtual machine migration scheduling problem in geographically distributed data centers. In *The International Conference on Cloud Computing*, pages 798–801. IEEE, Juillet 2016.

[121] Jean Theodore. *Sécurité et Disponibilité des Données Stockées dans les Nuages*. phdthesis, Université de Toulouse 3 Paul Sabatier (UT3 Paul Sabatier), Decembre 2016.

[122] Gregory Vial. Different databases for different strokes. *IEEE Software*, 35 (2) :80–85, Mars 2018. doi : 10.1109/MS.2018.1661308.

[123] L. Wan, Z. Lu, Q. Cao, F. Wang, S. Oral, and B. Settlemyer. SSD-optimized workload placement with adaptive learning and classification in HPC environments. In *Symposium on Mass Storage Systems and Technologies*, pages 1–6. IEEE, Juin 2014.

[124] Guoxi Wang and Jianfeng Tang. The NoSQL principles and basic application of cassandra model. In *The International Conference on Computer Science and Service System*, pages 1332–1335. IEEE, Août 2012.

[125] Xiaojian Wu and AL Narasimha Reddy. Managing storage space in a flash and disk hybrid storage system. In *The International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems*, pages 1–4. IEEE, Septembre 2009.

[126] W. Xiao, X. Lei, R. Li, N. Park, and D. J. Lilja. PASS : A Hybrid Storage System for Performance-Synchronization Tradeoffs Using SSDs. In *The International Symposium on Parallel and Distributed Processing with Applications*, pages 403–410. IEEE, Juillet 2012.

[127] Y. Xiaoyong, T. Hongyan, L. Ying, J. Tong, L. Tiancheng, and W. Zhonghai. A competitive penalty model for availability based cloud SLA. In *The International Conference on Cloud Computing (CLOUD)*, pages 964–970. IEEE, Juin 2015.

[128] Haitham Yaish, Madhu Goyal, and George Feuerlicht. A Method of Optimizing Multi-tenant Database Query Access. In *The Web Information Systems Engineering – WISE 2013 Workshops*, pages 194–212. Springer, Octobre 2013.

[129] J. Yang and Feng-Bin Sun. A comprehensive review of hard-disk drive reliability. In *Annual Reliability and Maintainability. Symposium.*, pages 403–409. IEEE, Janvier 1999.

[130] G. Zhang, L. Chiu, and L. Liu. Adaptive Data Migration in Multi-tiered Storage Based Cloud Environment. In *The International Conference on Cloud Computing (CLOUD)*, pages 148–155. IEEE, Juillet 2010.

[131] Ning Zhang, Junichi Tatemura, Jignesh Patel, and Hakan Hacigumus. Re-evaluating Designs for Multi-tenant OLTP Workloads on SSD-basedI/O Subsystems. In *The ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM, Mars 2014.

[132] Ning Zhang, Junichi Tatemura, Jignesh M. Patel, and Hakan Hacigumus. Toward cost-effective storage provisioning for DBMSs. *The VLDB Journal*, 23(2) :329–354, Avril 2014. doi : 10.1007/s00778-013-0334-x.

[133] 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, Mai 2010. doi : 10.1007/s13174-010-0007-6.