# Placement autonomique de machines virtuelles sur un système de stockage hybride dans un cloud IaaS

Hamza Ouarnoughi


**THÈSE / UNIVERSITÉ DE BRETAGNE OCCIDENTALE**
sous le sceau de l'Université Bretagne Loire

Pour obtenir le titre de
**DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE OCCIDENTALE**
*Mention : Science et Technologie de l'Information et de la Communication*
***École Doctorale Santé, Information, Communication, Mathématique, Matière***

présentée par
Hamza Ouarnoughi
Préparée à l'IRT b<>com et au
*Laboratoire CNRS Lab-STICC*

# Placement autonomique de machines virtuelles sur un système de stockage hybride dans un *cloud IaaS*

**Thèse soutenue le 03 juillet 2017**
devant le jury composé de :

**Olivier BARAIS**
*Professeur, Université de Rennes 1 / rapporteur*
**Daniel HAGIMONT**
*Professeur, INP Toulouse ENSEEIHT / rapporteur*
**Kaoutar El MAGHRAOUI**
*Chercheuse, IBM Thomas J. Watson Research Center / examinateur*
**Smaïl NIAR**
*Professeur, Université de Valenciennes / examinateur*
**Jalil BOUKHOBZA**
*Maître de conférence, Université de Bretagne Occidentale / co-encadrant de thèse*
**Stéphane RUBINI**
*Maître de conférence, Université de Bretagne Occidentale / co-encadrant de thèse*
**Frank SINGHOFF**
*Professeur, Université de Bretagne Occidentale / directeur de thèse*
**David TARDIVEL**
*Directeur du laboratoire Cloud Computing, IRT b<>com / invité*

# REMERCIEMENTS

**Je tiens à remercier** ...

**Frank Singhoff**, de m'avoir accompagné tout au long de ces années de thèse. Je le remercie pour l'encadrement, pour m'avoir transmis l'esprit scientifique rigoureux, et pour le temps qui m'a consacré afin de mener à bien ce travail.

**Stéphane Rubini**, pour son travail de suivi, pour ses contributions et ses remarques enrichissantes, tout en soulignant sa gentillesse et sa bienveillance.

Spécialement et chaleureusement, **Jalil Boukhobza**, sans qui je n'aurais pas pu aller aussi loin. Celui qui a cru en moi et m'a donné ma chance. Celui qui a su m'encadrer et guider mes premiers pas, aussi bien au niveau professionnel qu'au niveau personnel. J'espère que l'on continuera à travailler ensemble.

**Olivier Barais, Daniel Hagimont, Samail Niar, et Kaoutar El Maghraoui** d'avoir accepté l'évaluation de mon manuscrit de thèse et de la qualité de ma soutenance.

**Antoine Cabot, David Tardivel, Jean-Èmile Dartois** et tous mes collègues et responsables de l'IRT b<>com de m'avoir mis à disposition tous les moyens pour l'accomplissement de ce travail.

**Pierre Olivier, Djilali Boukhlef, Hai Nam Tran, Islam Naas**, et tous mes collègues du laboratoire Lab-STICC de l'UBO que j'ai pu côtoyer et avec qui j'ai passé des moments formidables.

**Mes parents, mes frères et sœurs**, et tous les membres de la famille **Ouarnoughi** qui ont toujours été d'un soutien continu et inconditionnel, et une source de motivation inépuisable. Je n'oublierai jamais l'aide et le soutien d'**Abd Elmalek Toumiat**, de ma tante bien aimée **Souheila** et son mari **Ben Aissa**.

«وَمَا أُوتِيتُم مِّنَ الْعِلْمِ إِلَّا قَلِيلًا»

سورة الإسراء

**« Et on ne vous a donné que peu de connaissance »**

Sourate Al Israa

**Résumé**

Les opérateurs de *cloud IaaS* (*Infrastructure as a Service*) proposent à leurs clients des ressources virtualisées (CPU, stockage et réseau) sous forme la de machines virtuelles (VM). 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. En plus des investissements liés à l'achat des infrastructures et de leur coût d'utilisation, la consommation énergétique apparaît comme un point de dépense important (2% de la consommation mondiale) et en constante augmentation. Sa maîtrise représente pour ces opérateurs un levier très intéressant à exploiter. D'un point de vue technique, le contrôle de la consommation énergétique s'appuie essentiellement sur les méthodes de consolidation. Or la plupart d'entre elles ne prennent en compte que l'utilisation CPU des machines physiques (PM) pour le placement de VM. En effet, des études récentes ont montré que les systèmes de stockage et les E/S disque constituent une part considérable de la consommation énergétique d'un centre de données (entre 14% et 40%).

Dans cette thèse nous introduisons un nouveau modèle autonomique d'optimisation de placement de VM inspiré de MAPE-K (*Monitor, Analyze, Plan, Execute, Knwoledge*), et prenant en compte en plus du CPU, les E/S des VM ainsi que les systèmes de stockage associés. Ainsi, notre première contribution est relative au développement d'un outil de trace des E/S de VM multi-niveaux. Les traces collectées alimentent, dans l'étape *Analyze*, un modèle de coût étendu dont l'originalité consiste à prendre en compte le profil d'accès des VM, les caractéristiques du système de stockage, ainsi que les contraintes économiques de l'environnement cloud. Nous analysons par ailleurs les caractéristiques des deux principales classes de stockage, pour aboutir à un modèle hybride exploitant au mieux les avantages de chacune. 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. L'étape *Plan* a donné lieu, d'une part, à une extension de l'outil de simulation *CloudSim* pour la prise en compte des E/S des VM, du caractère hybride du système de stockage, ainsi que la mise en œuvre du modèle de coût proposé dans l'étape Analyze. Nous avons proposé d'autre part, plusieurs heuristiques se basant sur notre modèle de coût et que nous avons intégrées dans *CloudSim*. Nous montrons finalement que notre approche permet d'améliorer d'un facteur trois le coût de placement de VM obtenu par les approches existantes.

**Abstract**

IaaS cloud providers offer virtualized resources (CPU, storage, and network) as Virtual Machines (VM). 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. In addition to investments related to infrastructure purchase and cost of use, energy efficiency is a major point of expenditure (2% of world consumption) and is constantly increasing. Its control represents a vital opportunity. From a technical point of view, the control of energy consumption is mainly based on consolidation approaches. These approaches, which exclusively take into account the CPU use of physical machines (PM) for the VM's placement, present however many drawbacks. Indeed, recent studies have shown that storage systems and disk I/O represent a significant part of the data center energy consumption (between 14% and 40%).

In this thesis we propose a new autonomic model for VM placement optimization based on MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) whereby in addition to CPU, VM I/O and related storage systems are considered. Our first contribution proposes a multilevel VM I/O tracer which overcomes the limitations of existing I/O monitoring tools. In the Analyze step, the collected I/O traces are introduced in a cost model which takes into account the VM I/O profile, the storage system characteristics, and the cloud environment constraints. We also analyze the complementarity between the two main storage classes, resulting in a hybrid storage model exploiting the advantages of each. 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 Plan phase has initially resulted in an extension of *CloudSim* to take into account VM I/O, the hybrid nature of the storage system, as well as the implementation of the previously proposed cost model. Secondly, we proposed several heuristics based on our cost model, integrated and evaluated using *CloudSim*. Finally, we showed that our contribution improves existing approaches of VM placement optimization by a factor of three.

# PUBLICATIONS

**Revues**

1. **(*)** H. Ouarnoughi, J. Boukhobza, F. Singhoff, S. Rubini, "**Integrating I/Os in Cloudsim for Performance and Energy Estimation**", *ACM SIGOPS Operating Systems Review*, Volume 50 Issue 1, Pages 27-36, December 2016.

2. P. Olivier, J. Boukhobza, E. Senn, H. Ouarnoughi, "**A Methodology for Estimating Performance and Power Consumption of Embedded Flash File Systems**", *ACM Transactions on Embedded Computing Systems*, Volume 15, Issue 4, Article No. 79, August 2016.

3. H. Ouarnoughi, J. Boukhobza, P. Olivier, L. Plassart, L. Bellatreche , "**Performance analysis and modeling of SQLite embedded databases on flash file systems**", *Design Automation for Embedded Systems*, Volume 17, Issue 3, Pages 507–542, September 2013.

**Conférences internationales & Workshops**

4. D. Boukhlef, K. Boukhalfa, J. Boukhobza, H. Ouarnoughi, L. Lemarchand "**COPS : Cost Based Object Placement Strategies on Hybrid Storage System for DBaaS Cloud**", *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, Pages 659-664 , Madrid, Spain, May 2017 .

5. **(*)** H. Ouarnoughi, J. Boukhobza, F. Singhoff, S. Rubini, "**Considering I/O Processing in CloudSim for Performance and Energy Evaluation**", *ISC High Performance International Workshops (WOPSSS)*, Lecture Notes in Computer Science, Volume 9945, Pages 591-603, Frankfurt, Germany, June 2016.

6. **(*)** H. Ouarnoughi, J. Boukhobza, F. Singhoff, S. Rubini, "**A Cost Model for Virtual Machine Storage in Cloud IaaS Context**", *24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, Pages 664–671, Crete, Greece, February 2016.

7. **(*)** H. Ouarnoughi, J. Boukhobza, F. Singhoff, S. Rubini, "**A multi-level I/O tracer for timing and performance storage systems in IaaS cloud**", *3rd IEEE International Workshop On Real-Time and Distributed Computing in Emerging Applications (REACTION)*, Rome, Italy, December 2014.

1 Les publications précédées par **(*)** sont les contributions issues des travaux de cette thèse.

8. J. Boukhobza, P. Olivier, L. Plassart, H. Ouarnoughi, L. Bellatreche, "**Embedded Databases on Flash Memories : Performance and Lifetime Issues, the case of SQLite**", *Embedded Real-time Software and Systems (ERTSS)*, Toulouse, France, February 2014.

**Communications**

9. **(*)** H. Ouarnoughi, J. Boukhobza, F. Singhoff, S. Rubini, "**A Cost Model for Virtual Machine Storage in Cloud IaaS Context** (poster)", *5th Workshop On Storage and data analytics (WOS5)*, Rennes, France, November 2015 ;

10. **(*)** H. Ouarnoughi, J. Boukhobza, F. Singhoff, S. Rubini, "**A Cost Model for Virtual Machine Storage in Cloud IaaS Context** (présentation)", *2nd Workshop On Performance and Scalablity of Storage Systems (Per3S)*, Versailles, France, January 2016 ;

**Rapports techniques**

11. **(*)** H. Ouarnoughi, J. Boukhobza, "**Etat de l'art et étude des mécanismes de stockage et de gestion de données massives dans un univers réparti**", Livrable ANR. Projet : INDEED, IRT b<>com, 2015.

# TABLE DES MATIÈRES

# TABLE DES FIGURES

# LISTE DES TABLEAUX

# ACRONYMES

**AFR** Annualized Failure Rate

**API** Application Programming Interface

**BFD** Best Fit Decreasing

**CIFS** Common Internet File System

**CPU** Central Processing Unit

**DAS** Direct Attached Storage

**EEPROM** Electrically Erasable Programmable Read-Only Memory

**E/S** Entrée/Sortie

**FC** Fibre Channel

**FTL** Flash Translation Layer

**FS** File System

**GDB** Gnu DeBugger

**HDD** Hard Disk Drive

**HPC** High Performance Computing

**HPPM** Heuristic with Packing into Physical Machines

**HPSD** Heuristic with Packing into Storage Devices

**IaaS** Infrastructure as a Service

**IOPS** Input/Output Operations Per Second

**IP** Internet Protocol

**IQR** InterQuartile Range

**iSCSI** Internet Small Computer Systems Interface

**LAN** Local Area Network

**LRR** Local Regression Robust

**MAD** Median Absolute Deviation

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

**MARS** Multivariate Adaptive Regression Splines

**MC** Maximum Correlation

**MI** Million Instructions

**MIPS** Million Instructions Per Second

**MLC** Multi-Level Cell

**MMT** Minimum Migration Time

**MPI** Message Passing Interface

**MTBF** Mean Time Between Failures

**MTTF** Mean Time To Failure

**MU** Minimum Utilization

**NAS** Network Attached Storage

**NFS** Network File System

**NRC** Non-Recurring Cost

**OLS** Ordinary Least Squares

**OOB** Out Of Band

**PaaS** Platform as a Service

**PABFD** Power Aware Best Fit Decreasing

**PDU** Power Distribution Unit

**PM** Physical Machine

**QoS** Quality of Service

**RC** Random Choice

**RAM** Random Acess Memory

**RAMAC** Random Access Memory ACcounting

**SaaS** Software as a Service

**SAN** Storage Area Network

**SCSI** Small Computer System Interface

**SD** Secure Digital

**SLC** Single-Level Cell

**SLA** Server Level Agreement

**SSD** Solid-State Drive

**TCO** Total Cost of Ownership

**TCP** Transmission Control Protocol

**TLC** Triple-Level Cell

**USB** Universal Serial Bus

**VFS** Virtual File System

**VM** Virtual Machine

**VMM** Virtual Machine Manager/Monitor

# Chapitre 1

# INTRODUCTION

**Sommaire**



## 1.1 CONTEXTE GÉNÉRAL

Le *cloud computing* est l'une des technologies émergentes dont l'utilisation est devenue omniprésente, tant dans la vie privée, que dans la vie professionnelle. Le *cloud Infrastructure as a Service* (IaaS), est la catégorie du *cloud* qui propose une infrastructure matérielle (i.e. CPU, stockage, réseau), et logicielle (i.e. technologies de virtualisation), sous la forme d'un seul service. Les ressources offertes par ce service sont des ressources virtualisées, à savoir des machines virtuelles (VM pour *Virtual Machine*).

Offrir des services attractifs et à moindre coût constitue un grand défi pour les fournisseurs de *cloud*. Les coûts des infrastructures peuvent dépasser 50% des coûts totaux des centres de données [4]. À titre d'exemple, Google a dépensé près de 5 milliards de dollars pour ses centres de données dans le deuxième trimestre de l'année 2014 [4]. Plus de 2.8 milliards de dollars de cette somme étaient destinés à l'achat et au fonctionnement des infrastructures matérielles. La figure 1 présente le coût par composant d'un centre de données.

Comme le montre la figure 1, les coûts des ressources matérielles (i.e. CPU, RAM, stockage), ainsi que les coûts liés à l'énergie, représentent près de 60% de la totalité des coûts d'un centre de données. Des études récentes [133] évoquent que près de 2% de l'électricité produite dans le monde est consommée par les centres de données. De plus, le coût de l'énergie consommée par une machine

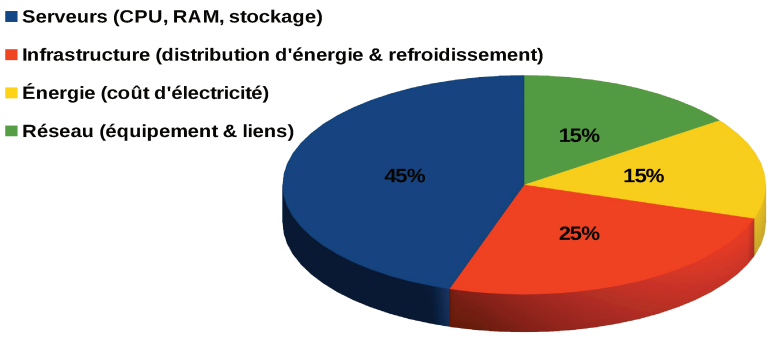


FIGURE 1 : coûts d'un centre de données (extrait de [77])

physique (PM pour *Physical Machine*) durant sa durée de vie dépasse son coût d'achat [21].

Différentes approches ont été proposées afin de minimiser le coût énergétique des centres de données [151]. Parmi celles-ci, nous pouvons citer les approches de consolidation des serveurs [182][203]. La figure 2 illustre le principe des algorithmes de consolidation.

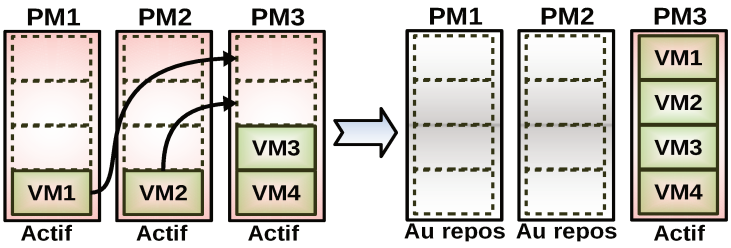


FIGURE 2 : principe de consolidation des serveurs

Comme le montre la figure 2, le principe d'un algorithme de consolidation est de concentrer l'utilisation sur certaines PM pour éteindre les autres.

Les approches de consolidation font appel à des méthodes d'optimisation afin de trouver l'allocation des VM sur les PM. Cette allocation peut être basée sur plusieurs objectifs et contraintes (e.g. énergie, qualité de service, pénalités). Ce type d'approche se base essentiellement sur l'utilisation CPU des PM [127].

Les systèmes de stockage et l'exécution des entrées/sorties (E/S pour Entrées/-Sorties) représentent une part considérable de la consommation totale de l'énergie. Leur contribution peut aller de 14% dans les environnements virtualisés [100], jusqu'à 40% dans un centre de données [35]. Ces chiffres mettent en évidence la nécessité de la prise en compte du stockage dans l'optimisation du placement de VM.

Les systèmes de stockages sont associés à différentes contraintes dans le *cloud* [16]. Les disques durs magnétiques (HDD pour *Hard Disk Drive*) représentent un goulet d'étranglement, et cela du fait de l'écart de performances entre les unités

de calcul (CPU et RAM) et celle du stockage. Par ailleurs, ce type de périphérique de stockage est énergivore à cause de ses composants mécaniques.

Les périphériques de stockage à base de mémoire flash (SSD pour *Solid-State Drive*) peuvent combler l'écart de performances, tout en économisant l'énergie. En revanche, ces périphériques souffrent de quelques lacunes. Leur prix reste élevé par rapport à celui des disques durs (~5 fois plus élevé en moyenne [99]). De plus, leur durée de vie est plus courte que celle des disques durs et dépend fortement de la quantité de données écrites sur le périphérique.

Les systèmes de stockage hybrides sont de plus en plus utilisés afin de tirer profit des avantages de chaque type de périphérique de stockage [142][175]. Dans la littérature, un système de stockage hybride fait référence à un système de stockage qui regroupe HDD et SSD [128][108][199].

Un système de stockage hybride sans politique de gestion adaptée ne peut pas résoudre tous les problèmes des systèmes de stockage (énergie, performances, prix, etc). Il faut donc que la gestion de ces systèmes utilisent des méthodes d'optimisation de placement de données, afin d'exploiter les avantages du stockage hybride. Ce problème devient complexe s'il est associé à l'optimisation du placement de VM.

Le travail présenté dans cette thèse s'inscrit dans le projet INDEED mené par l'Institut de Recherche Technologique b<>com. Le projet INDEED a pour objectif d'optimiser l'utilisation des ressources matérielles des centres de données. Cette optimisation vise les infrastructures du *cloud IaaS*. Comme objectif initial, le projet INDEED vise la réduction de la consommation énergétique des centres de données. Les méthodes et outils issues de ce projet sont le produit d'une collaboration entre des acteurs académiques (i.e. UBO, INRIA, ENSTA Bretagne, IRISA), et industriels (Orange). Le projet INDEED a donné naissance au module *OpenStack Watcher*[147]. *Watcher* permet l'orchestration de l'utilisation des ressources des centres de données, en allant de la remonté des métriques (i.e. énergie, performance) jusqu'à l'optimisation du placement de VM.

Cette thèse présente le travail mené sur l'optimisation de stockage des données des VM. Nous nous intéressons particulièrement aux systèmes de stockage hybrides pour nos approches d'optimisation. Notre rôle dans le projet INDEED était de fournir les outils et les méthodes, qui permettent de mieux exploiter les ressources de stockage dans le processus de placement de VM. Dans la prochaine partie, nous donnons plus de détails concernant les motivations et les problématiques visées par ce travail.

## 1.2 PROBLÉMATIQUES

Notre objectif est de quantifier et de prendre en compte les coûts d'exécution d'E/S dans le placement des VM. Dans cette partie, nous décrivons les problèmes traités dans ce travail de thèse.

### 1.2.1 *Monitoring des charges d'E/S des VM*

Les clients d'un *cloud* IaaS peuvent utiliser leurs VM pour exécuter différents types d'application (serveurs web, bases de données, calcul, etc). Dans ce cas d'usage, le fournisseur du service n'a pas d'idées préalables sur la nature de l'applicatif exécuté par les VM. Afin d'effectuer un placement optimal des données des VM, il est important de connaître le profil d'accès au stockage des VM. Il nous faut donc des fonctions de *monitoring* et de caractérisation de ces accès.

### 1.2.2 *Coût de stockage et d'exécution des E/S*

L'optimisation de placement de VM nécessite l'utilisation d'une fonction de coût. En effet, le placement des données des VM et l'exécution de leur charges d'E/S peuvent engendrer différents coûts. Ces coûts peuvent provenir de plusieurs parties (énergies, pénalités, etc). Il est donc important d'identifier l'ensemble de ces coûts sous forme d'une seule fonction. L'exactitude de cette fonction permet un meilleur placement de VM [127].

### 1.2.3 *Prise en compte du coût de stockage et des E/S pendant l'optimisation*

Plusieurs approches ont été proposées pour l'optimisation de placement de VM [92]. Ces approches ont souvent été implantées et validées à l'aide d'outils de simulation [127]. L'un des défauts de ces outils est qu'ils ne considèrent pas le traitement des E/S. Nos expérimentations ont montré que l'exécution des E/S peut engendrer une utilisation de CPU avoisinant les 38% [154]. Il est donc nécessaire de prendre en compte l'exécution des E/S et les systèmes de stockage dans l'implantation et l'évaluation des approches d'optimisation de placement de VM.

### 1.2.4 *Approches d'optimisation de placement de VM*

Il existe différents types d'approches d'optimisation appliquées au placement de VM [12][127] : des algorithmes gloutons, des approches exactes, des heuristiques et méta-heuristiques, etc. Ces approches ne considèrent pas l'ensemble des contraintes sur les infrastructures matérielles et sur l'environnement *Cloud*

[127]. Il est important de proposer des approches d'optimisation qui prennet en compte les contraintes les plus importantes, tout en permettant une prise de décision en un minimum de temps de calcul (de quelques millisecondes à quelques heures [108]). Il nous faut donc des approches d'optimisation de placement de VM qui visent à minimiser les coûts d'infrastructure. Elle doivent également considérer les contraintes des ressources matérielles et de l'environnement *cloud*, tout en respectant les contraintes temporelles d'une optimisation de placement en ligne.

## 1.3 CONTRIBUTIONS

Cette partie présente nos contributions aux problèmes soulevés dans la partie précédente. Chaque sous-partie résume brièvement chaque contribution ainsi que les résultats obtenus. Nos contributions sont présentées suivant le modèle MAPE-K (*Monitor*, *Analyze*, *Plan*, *Execute*, *Knowledge*) [103].

### 1.3.1 *Le modèle MAPE-K*

L'acronyme MAPE-K fait référence à *Monitor, Analyze, Plan, Execute, Knowledge*. C'est un modèle qui définit la gestion autonomique d'un système informatique [103]. Ce modèle se base sur la mise en œuvre d'une boucle contrôlant un système dont on attend un comportement autonomique [55].

Comme son acronyme l'indique, le modèle MAPE-K comporte cinq étapes essentielles. La première étape, *Monitor*, consiste au *monitoring* du système géré, en utilisant un ensemble de capteurs préalablement installés. La deuxième étape, *Analyze*, récupère les résultats de l'étape *Monitor* afin de les analyser et de déterminer l'état du système. Une fois les résultats de l'étape *Analyze* prêts, l'étape *Plan* permet de déterminer une nouvelle configuration du système qui répond aux objectifs visés. La nouvelle configuration est appliquée sur le système géré par la suite à l'étape *Execute*, 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. Le modèle MAPE-K est détaillé dans la partie 2.7.

### 1.3.2 *Comment MAPE-K s'applique-t-il à notre problème ?*

Nous présentons nos contributions suivant le modèle MAPE-K. L'adaptation de nos approches au modèle MAPE-K est la suivante :

- l'interaction entre les VM et le système de stockage doit être surveillée, afin d'estimer la charge d'E/S des VM. Il s'agit donc de l'étape *Monitoring* ;

- les profils d'E/S des VM nous permettent d'analyser leur impact sur le coût de l'infrastructure. Il s'agit de l'étape *Analyze* ;
- une fois les coûts d'exécution des E/S des VM évalués, le processus d'élaboration d'un plan de placement qui minimise ces coûts est enclenché. Il s'agit de l'étape *Plan* ;
- le nouveau plan de placement doit être mis en place. Il s'agit de l'étape *Execute* ;
- les états du système, les profils E/S des VM, les plans de placement correspondants, sont des éléments qui peuvent être regroupés et exploités lors des prochaines itérations du modèle MAPE-K. Il s'agit de l'étape *Knowledge* ;

La figure 3 présente notre adaptation du modèle MAPE-K, appliquée aux problématiques étudiées dans ce travail de thèse.

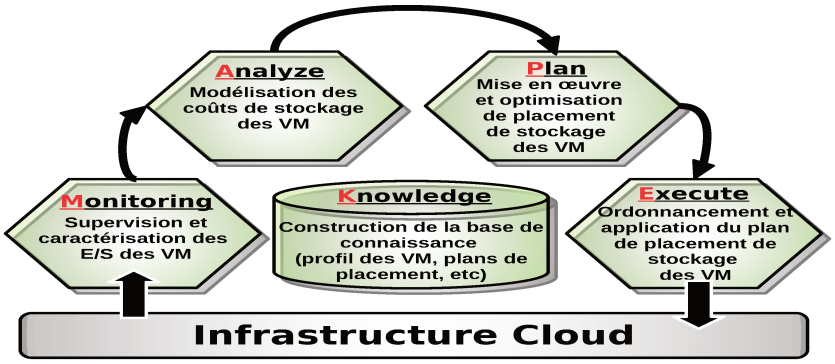


FIGURE 3 : modèle MAPE-K pour l'optimisation de placement des données de VM

Les prochaines parties décrivent nos contributions suivant la démarche MAPE-K.

### 1.3.3 *Monitoring et caractérisation des activités E/S des VM*

Notre première contribution vise la surveillance (*Monitoring*) des activités d'E/S des VM. Nous avons proposé un outil de trace multi-niveaux des E/S des VM. Le traceur proposé a été développé en deux versions.

La première version combine des bibliothèques et des outils de traces existants, pour en faire un seul outil capable de tracer plusieurs VM, et de fournir des traces et des statistiques sur les activités d'E/S de chaque VM en cours d'exécution sur le système. Ce travail a été publié dans [152].

La deuxième version du traceur a été développée sous la forme d'un module noyau Linux, dans le but de surpasser certaines limites de la premières version. Cette version a été mise en œuvre par le module *OpenStack Watcher* [147] developpé au sein de l'IRT b<>com, et il est disponible en accès libre [5]. Les deux versions ont été évaluées sur une plate-forme réelle.

### 1.3.4 *Modélisation des coûts de stockage des VM dans les infrastructures cloud*

Pour l'étape *Analyze*, nous avons proposé un modèle de coût qui inclut tous les coûts fixes et variables induits par le stockage des données des VM sur les périphériques de stockage. Notre modèle 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 et les besoins des clients en termes de qualité de service spécifiée dans le SLA (*Service Level Agreement*).

La fonction objectif de nos approches d'optimisation a été construite autour de ce modèle de coût. Notre modèle a été publié dans [153].

### 1.3.5 *Intégration et simulation des E/S et du système de stockage dans le cloud*

Après avoir traité les deux premières étapes *Monitoring* et *Analyse* du modèle MAPE-K, nous nous sommes appuyé sur ces étapes afin de construire la phase d'optimisation *Plan*.

La mise en œuvre de notre modèle de coût est évaluée au sein du simulateur de *cloud CloudSim* [40]. Cependant, *CloudSim* ne considère pas la variété des périphériques de stockage, ni le calcul du temps et de la consommation énergétique liée à l'exécution des charges d'E/S des VM.

Nous avons donc proposé une extension pour *CloudSim* qui prend en compte la simulation de différentes classes de stockage (i.e. HDD et SSD), ainsi que le temps et la consommation énergétique engendré par l'exécution des charges d'E/S des VM. L'extension proposée intègre un modèle de corrélation qui permet d'estimer l'utilisation CPU des PM, en fonction des charges d'E/S exécutées par les VM. Ce travail a été publié dans [154] et [155].

### 1.3.6 *Optimisation des coûts en considérant le stockage et les E/S des VM*

L'extension de *CloudSim* pour le support du stockage hybride et l'exécution des E/S des VM nous permet l'implantation de nos approches d'optimisation de placement de VM, tout en considérant les coûts de stockage. En effet, nous avons conçu et implanté un ensemble d'algorithmes d'optimisation de placement de VM dont le fonction objectif est construite autour de notre modèle de coût.

Nous avons proposé deux types de méthodes d'optimisation de placement de VM, à savoir une méthode exacte, et trois heuristiques. Ces algorithmes ont comme objectif la minimisation des coûts d'infrastructure tout en prenant en compte l'ensemble des contraintes liées à l'exécution des E/S des VM.

La partie suivante liste les hypothèses qui délimitent le cadre de notre travail.

## 1.4 HYPOTHÈSES

Nos approches ont été proposées en nous basant sur un certain nombre d'hypothèses. Cette partie présente l'ensemble des hypothèses prises en compte dans chacune de nos contributions.

- ***H1 : Type de cloud***. Notre étude s'inscrit dans le contexte d'un environnement de *cloud* de type *IaaS* dont le service proposé est sous une forme de location de VM à la demande (e.g. *Amazon EC2* [15], *Google CE*[75]).
- ***H2 : Virtualisation***. Nos approches ont été validées dans des environnement virtualisés utilisant des hyperviseurs de type *hosted* (voir partie 2.6.3).
- ***H3 : Coût du réseau***. La modélisation des coûts du réseau et des communications entre les PM et les VM ne fait pas partie de notre étude.
- ***H4 : Interférences entre VM***. Notre étude considère qu'il n'y a pas d'interférences entre des VM qui partagent les mêmes ressources.
- ***H5 : Topologie du système de stockage***. Nos propositions visent les systèmes de stockage dont les périphériques peuvent être directement attachés aux PM (i.e. en DAS) ou accédés à distance (i.e. en NAS ou SAN).
- ***H6 : Périphériques de stockage***. Notre étude a été réalisée autour de systèmes de stockage hybrides composés de HDD et des SSD exploités au même niveau (i.e. sans hiérarchisation).
- ***H7 : Migration de VM***. Dans notre contexte, la migration d'une VM comporte le transfert de son processus, son espace mémoire ainsi que ses disques virtuels d'une PM vers une autre.
- ***H8 : Réplication de stockage***. Nos approches et modèles considèrent que chaque VM utilise une seule image disque pour son stockage de données.

## 1.5 PLAN DU MÉMOIRE

Ce manuscrit de thèse est composé de huit chapitres organisés en trois parties. La première partie regroupe le contexte de notre étude et un état de l'art concernant nos contributions. La deuxième partie détaille chacune de nos contributions. La troisième partie conclut ce document.

**Première partie : contexte et état de l'art**

**Le chapitre 2** définit le contexte et les concepts généraux sur les systèmes de stockage et le *cloud* IaaS.

**Le chapitre** 3 présente un état de l'art concernant l'ensemble de nos contributions.

**Deuxième partie : contributions**

**Le chapitre** 4 décrit notre première contribution sur le *Monitoring*. D'abord, nous détaillons l'architecture et la démarche suivie pour la construction de notre processus de *monitoring* des activités d'E/S des VM. Ensuite, nous illustrons la caractérisation des charges d'E/S des VM. Enfin, nous y montrons également la méthode et les résultats de l'évaluation de nos outils.

**Le chapitre** 5 est consacré à la modélisation des coûts de stockage et d'exécution des E/S de VM. Ce chapitre introduit les différents coûts (i.e. exécution des E/S et migration des VM) constituant le modèle de coût global. Nous y décrivons ensuite la méthode et les résultats d'évaluation de ce modèle.

**Le chapitre** 6 développe la mise en œuvre du modèle de coût, en commençant par son intégration dans le simulateur *CloudSim*. Cette intégration a donné lieu à une extension de *CloudSim* pour la prise en compte des systèmes de stockage hybrides et de l'exécution des charges d'E/S des VM.

**Le chapitre** 7 présente nos approches d'optimisation de placement de VM, avec la prise en compte des systèmes de stockage et des E/S des VM. Nous commençons par introduire brièvement les principes d'optimisation de placement de VM dans *CloudSim*. Nous détaillons par la suite nos approches d'optimisation pour finir par une évaluation et une discussion des résultats obtenus.

**Troisième partie : conclusion**

**Le chapitre** 8 conclut le manuscrit en résumant les différentes contributions, et en présentant quelques perspectives.

# Première partie

# CONTEXTE ET ÉTAT DE L'ART

# Chapitre 2

# CONTEXTE

Afin de cerner le cadre de notre travail, nous présentons les concepts essentiels, à savoir les systèmes de stockage et les E/S sous Linux, le *cloud computing* et la virtualisation, et enfin le modèle MAPE-K. Ce chapitre présente le contexte de notre étude, ainsi que les outils utilisés tout au long de notre étude.

**Sommaire**

## 2.1 INTRODUCTION

Notre étude fait partie du projet INDEED qui a été mené au sein de l'IRT b<>com (voir partie 1.1). L'objectif de cette étude est l'optimisation du placement des VM en prenant en compte différentes contraintes. Ces contraintes dépendent de l'environnement *cloud* et des systèmes de stockage utilisés. La figure 4 montre un scénario représentatif du contexte général de notre étude.

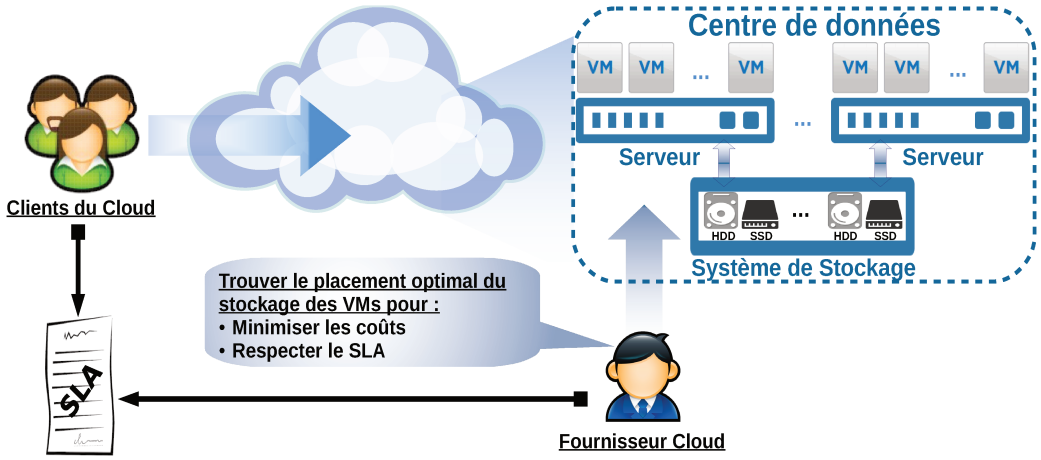


FIGURE 4 : contexte général de notre étude

Comme le présente la figure 4, notre scénario d'utilisation implique deux acteurs principaux :

1. les clients du service *cloud* ;
2. le fournisseur du service.

Les clients utilisent les services *cloud* en exécutant des VM, tandis que le fournisseur a pour rôle de satisfaire les demandes des clients tout en essayant de maximiser ses revenus. Notre approche prend en considération des contraintes en relation avec le système de stockage utilisé (e.g. architecture, types des périphériques), ainsi que des contraintes liées à l'environnement *cloud*. Cette partie expose l'ensemble de ces contraintes.

La suite de ce chapitre est divisée en trois parties. La première présente les prérequis pour comprendre les systèmes de stockage, les E/S disque, et le *benchmarking* des systèmes de stockage. La deuxième partie introduit les principes du *cloud* IaaS, ainsi que le domaine de la virtualisation. La dernière partie concerne le modèle MAPE-K.

## 2.2 SYSTÈMES DE STOCKAGE

Après avoir introduit le contexte général de notre étude, cette partie présente les systèmes de stockage. La partie suivante définit quelques termes utilisés tout au long de ce document.

### 2.2.1 *Définitions*

**Définition 2.1.** *(FTL) [34] : acronyme de Flash Translation Layer, une FTL est une couche matérielle/logicielle située dans le contrôleur d'un périphérique de stockage utilisant des mémoires flash (e.g. clés USB, SSD, carte SD).*

**Définition 2.2.** *(IOPS) [31] : acronyme de Input/Output Operations Per Second. Cette métrique permet d'évaluer le nombre d'opérations d'E/S exécutées par seconde sur un système de stockage. Par convention, la taille d'une opération d'E/S est de 4Ko. Les IOPS sont utilisés pour évaluer la performance d'un système de stockage exécutant des accès aléatoires.*

**Définition 2.3.** *(**débit**) [60] : le débit d'un système est le nombre de tâches, de programmes, ou de processus complétés dans une unité de temps. Le débit est également utilisé pour mesurer les performances des systèmes de stockage pour les accès séquentiels. Dans ce cas, le débit est le nombre d'octets transférés par unité de temps [60]. Son unité de mesure est le mégaoctet par seconde (Mo/s).*

**Définition 2.4.** *(**latence**) [150] : la latence est le temps séparant l'envoi d'une requête et la réception d'une réponse ou d'un bloc de données. Il s'agit également d'une des métriques utilisées pour mesurer les performances d'un système de stockage.*

### 2.2.2 *Systèmes de stockage à base de disques dur magnétiques*

Depuis leur invention par IBM en 1956 [94], les disques durs sont devenus les principaux périphériques de stockage de masse des données numériques. Le premier disque dur, le 305 RAMAC (*Random Access Memory ACcounting*), avait une capacité de stockage d'environ 4 Mo, et après 60 ans de recherche et d'innovation, la capacité de stockage des disques durs a atteint quelques téraoctet. Cette capacité de stockage est accompagnée d'une amélioration des performances en termes de débit de transfert de données et de temps d'accès.

Les disques durs sont des périphériques magnétiques de stockage de masse, composés de deux parties principales, une partie électromécanique et une autre électronique [125]. La partie électromécanique comporte des disques rotatifs et des bras de lecture/écriture. La partie électronique contient le contrôleur disque, une interface de connexion au contrôleur, et un cache. La figure 5 montre les composants typiques d'un disque dur.

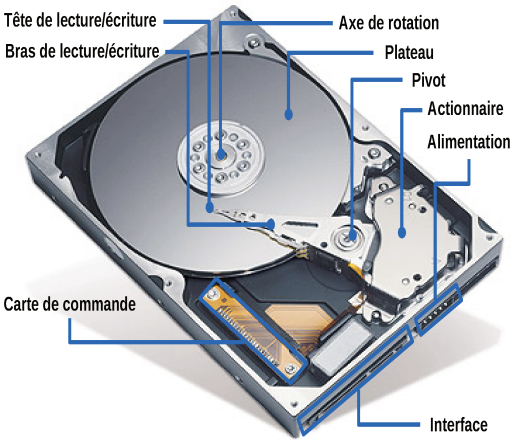


FIGURE 5 : composants d'un disque dur magnétique [125].

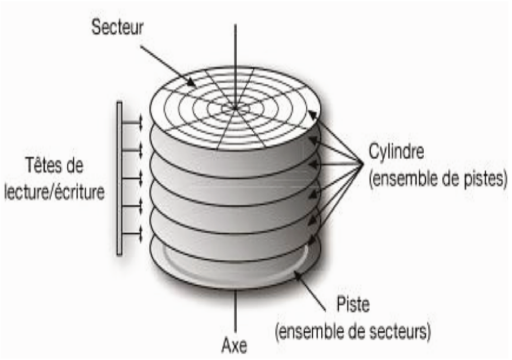


FIGURE 6 : architecture d'un disque dur magnétique [201].

Les disques durs sont composés d'un ensemble de plateaux superposés, et de têtes de lecture/écriture intercalées entre ces plateaux. Les plateaux sont divisés en pistes. L'ensemble des pistes se trouvant à la même position sur les différents plateaux constituent un cylindre. Les pistes sont divisées en secteurs dans lesquelles les bits sont stockés. Les secteurs sont généralement d'une taille de 512 octets. Nous pouvons également trouver des disques durs avec des secteurs d'une taille de 4 Ko. Cette catégorie est connue sous le nom de "*disques durs à format avancé*" ( *Advanced Format 4k* HDD). La lecture/écriture d'une donnée sur le disque dur se fait grâce à la combinaison du mouvement rotatif des plateaux avec le mouvement des têtes de lecture/écriture. La figure 6 montre l'architecture interne simplifiée d'un disque dur.

Les performances des disques durs dépendent essentiellement de la partie électromécanique (rotation des plateaux et mouvement de la tête de lecture/écriture), ainsi que de la densité de stockage. Les constructeurs ont longtemps misé sur la vitesse de rotation des plateaux afin d'améliorer les performances de stockage [178]. Il se heurtent finalement à une limite physique, et ainsi la vitesse de rotation n'évolue plus beaucoup depuis quelques années. Le mouvement des têtes de lecture/écriture durant les opérations d'E/S rajoute, lui aussi, un temps de latence, ce qui peut considérablement dégrader les performances du système de stockage.

Plusieurs travaux traitent du problème de la modélisation des performances des disques durs en prenant en compte les caractéristiques des parties mécaniques [168] [196]. Les activités mécaniques des disques durs ne dégradent pas seulement les performances, mais consomment également de l'énergie, et contribuent à leur usure.

Concernant sa consommation énergétique, un HDD peut se trouver dans l'un des trois modes suivants :

1. opérationnel (*operating mode*)
2. repos (*idle mode*)
3. prêt (*standby mode*) / sommeil (*sleep mode*)

Un HDD est en mode opérationnel durant l'exécution des opérations d'E/S. La puissance électrique qui correspond au mode opérationnel est la plus élevée des trois modes. Cela est dû au mouvement rotatif des plateaux du HDD et au déplacement de la tête de lecture/écriture. Le HDD passe du mode opérationnel au mode *idle* s'il ne traite pas ou ne reçoit pas de requêtes d'E/S pendant une période définie (appelée *timeout*). La puissance électrique qui correspond au mode *idle* est relative au mouvement de rotation des plateaux. La puissance la moins élevée est celle qui correspond au *standby mode*, car ce mode n'implique aucune activité mécanique du HDD.

La durée de vie d'un HDD dépend principalement de son temps de service. Nous pouvons trouver dans sa fiche technique différents paramètres à l'aide desquels nous estimons sa durée de vie. Ces paramètres sont calculés à partir de statistiques sur des grandes populations de HDD. Parmi ces paramètres citons le MTBF (*Mean Time Between Failures*), le MTTF (*Mean Time To Failure*), l'AFR (*Annualized Failure Rate*), et le nombre de cycles *start-stop*. Plus la valeur de MTTF (donnée en nombre d'heures) est proche de zéro, plus forte est la possibilité que le HDD tombe en panne [170].

### 2.2.3 *Systèmes de stockage à base de mémoires flash*

Les périphériques de stockage ont évolué au cours des années en termes de performance et de densité de stockage. Néanmoins, les disques durs ne suivent malheureusement pas les progrès des microprocesseurs et des mémoires principales en termes de performance temporelles. Cette tendance a creusé un écart de performance important entre les composants de traitement et ceux du stockage. Cet écart est l'une des motivations qui ont poussé les constructeurs de mémoires à réfléchir à des solutions de remplacement ou complémentaires. Une autre raison, tout aussi pertinente, est la consommation énergétique des disques durs causée par l'activité mécanique pendant les opérations d'E/S, notamment lors d'accès aléatoires. Ces réflexions ont donné naissance à une nouvelle technologie de stockage qui est celle des mémoires flash.

Une mémoire flash est une mémoire non-volatiles à base de semi-conducteurs. Ce type de mémoire EEPROM (*Electrically Erasable Programmable Read-Only Memory*) est une mémoire effaçable et programmable [179]. Il existe deux principaux types de mémoires flash différenciés par l'organisation des cellules de mémorisation utilisées dans leur conception : 1) NAND et 2) NOR.

Les mémoires flash NOR sont adressables par octet. Cette caractéristique fait que les mémoires flash NOR sont principalement utilisées au même niveau que la mémoire principale. Ce type de mémoire est souvent dédié au stockage du code exécuté sur place (*eXecute In Place, XIP*).

Les mémoires NAND sont caractérisées par un accès par pages, et leur densité leur permet d'être utilisées pour le stockage de données. Les périphériques de stockage de masse à base de mémoire flash sont généralement construits avec des mémoires NAND, comme les disques SSD (Solid State Drive) ou les cartes compact flash ou encore SD (*Secure Digital*). La partie suivante présente brièvement l'architecture des mémoires flash NAND et leurs principales caractéristiques.

#### 2.2.3.1 *L'architecture des mémoires flash NAND [145]*

Il existe différents variantes de mémoires flash NAND selon les types de cellules utilisées (nombre de bits stockés par cellule mémoire). Les mémoires flash NAND de type SLC (*Single Level Cell*) permettent de stocker un seul bit par cellule. Les mémoires MLC (*Multi Level Cell*) peuvent stocker 2 bits par cellule mémoire, tandis que les mémoires TLC (*Triple Level Cell*) stockent 3 bits. L'augmentation du nombre de bits par cellule mémoire permet de baisser le prix de la mémoire flash, mais engendre, en contrepartie, une diminution de la durée de vie de cette dernière, ainsi qu'une baisse de fiabilité et des performances. Nous retrouvons les types MLC et SLC dans les SSD.

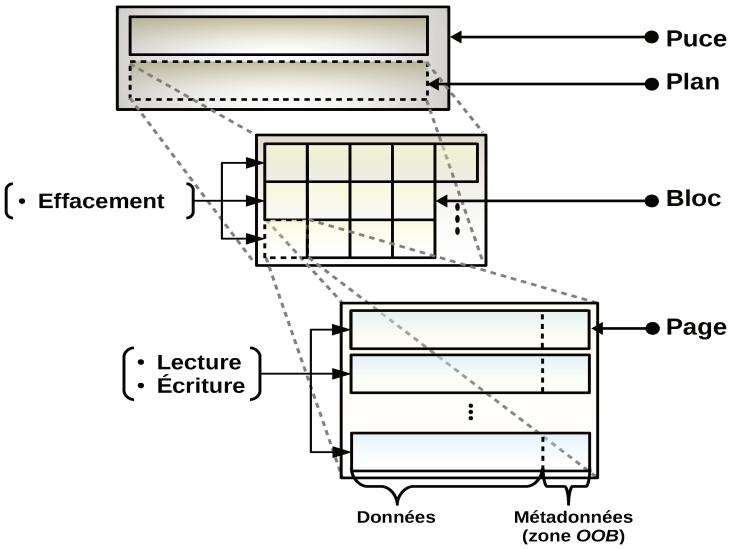


FIGURE 7 : architecture simplifiée d'une puce flash NAND

La figure 7 montre une architecture simplifiée d'une mémoire flash NAND, ainsi que les opérations principales d'E/S. Une puce flash NAND est composée d'un ensemble de plans. Les plans sont composés de blocs, et les blocs contiennent des pages [63][145].

***a) la page*** : dans une puce NAND, la page est la granularité sur laquelle s'effectuent les opérations de lectures et d'écritures. Une page est composée de deux zones (voir figure 7). La première constituant la plus grande partie de la page est dédiée aux données. La deuxième zone de la page est dédiée aux méta-données. Cette dernière est appelée zone hors bande (OOB *Out Of Band area*) et contient des informations sur l'état des cellules, les codes correcteurs d'erreur, etc. La taille d'une page varie en fonction des modèles de la puce NAND utilisée. Cette taille peut aller de 512 octets de données et 16 octets d'OOB [137], jusqu'à 8 Ko de données avec 128 octets d'OOB [79]. Une page peut être dans trois états :

- ***libre*** : la page est vide et peut stocker de nouvelles données ;
- ***valide*** : la page contient des données à jour ;
- ***invalide*** : la page contient des données obsolètes.

***b) le bloc*** : un bloc regroupe un ensemble de pages. Le nombre de pages par bloc varie selon le modèle de la puce NAND. Ce nombre est une puissance de deux qui peut être 32, 64, ou 128.

***c) le plan*** : le plan représente une matrice de blocs. Le nombre de bloc par plan ainsi que le nombre de plans par puce flash dépend du modèle de la puce NAND. En général, les puces flash NAND contiennent un ou deux plans. Il existe tout de même des puces flash pouvant contenir jusqu'à 4 plans par puce [138].

Les opérations élémentaires sur les mémoires flash sont décrites comme suit :

- ***l'écriture*** **:** les opérations d'écriture s'effectuent page par page. Une opération d'écriture est précédée par une opération d'effacement d'un bloc si la page contenait des données. Une modification d'une donnée ne se fait pas sur place. La page contenant l'ancienne version est marquée comme invalide, et la nouvelle version modifiée est écrite dans une nouvelle page libre. Lorsque toutes les pages d'un bloc sont marquées comme invalides, le bloc est effacé et ses pages deviennent à nouveau libres. Dans certains cas, il peut y avoir des opérations de fusion qui consistent à récupérer les pages valides d'un bloc, les écrire dans un bloc valide, puis effacer leur ancien bloc.
- ***la lecture*** **:** comme les opérations d'écriture, les opérations de lecture se font avec la granularité d'une page. Une lecture d'une page nécessite deux étapes : 1) une lecture de la zone OOB, 2) puis une lecture des données. Une page peut être lue plusieurs fois.
- ***l'effacement*** **:** les opérations d'effacement s'exécutent sur des blocs. Un bloc est effacé lorsque toutes ses pages sont marquées comme invalides. Le

nombre d'opérations d'effacement d'un bloc est limité et varie en fonction de la densité de la flash NAND.

#### 2.2.3.2 *Les contraintes des mémoires flash*

Leurs avantages en termes de performance et de consommation énergétique mis à part, les mémoires flash NAND exhibent certaines limitations/contraintes liées à leurs caractéristiques architecturales et physiques. Il existe trois principales contraintes [136] :

1. L'usure liée aux opérations d'écriture et d'effacement : le nombre de cycles d'écriture/effacement toléré par une cellule est limité et dépend de son type [32]. 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. L'impossibilité de mise à jour des données sur place : cette deuxième contrainte est liée au fait qu'une écriture sur une page contenant des données doit être précédée d'un effacement. Pour faire des modifications de données d'une page sur place, il faudrait d'abord lire le reste des données du bloc, effacer le bloc puis réécrire les données mises à jour. Ces opérations engendrent un surcoût en terme de performance et sont donc évitées si possible. Les modifications sont reportées sur d'autres pages.

3. La non-fiabilité des opérations de lecture et d'écriture : les principes physiques sur lesquels les mémoires flash NAND sont construites font qu'une erreur peut avoir lieu lors d'une opération d'écriture ou de lecture. Ces erreurs sont des inversions de bits causées par les tensions appliquées sur les cellules mémoires voisines [48].

Différentes solutions ont été proposées afin de gérer les contraintes citées précédemment. Ces solutions peuvent être logicielles et/ou matérielles. Il faut noter qu'une mémoire flash NAND ne peut être utilisée de la même façon qu'un disque dur sans avoir un système de gestion dédié à ces contraintes.

- L'impossibilité de mise à jour sur place est gérée en utilisant des techniques de traduction d'adresse. La traduction d'adresse sert à garder une trace du déplacement des données suite à une mise à jour. La donnée originelle est invalidée et la nouvelle est placée à une autre adresse. Cette adresse est maintenue dans une table de traduction. Différentes granularités de traduction d'adresse existent, notamment au niveau des pages, au niveau des blocs, ou hybride (i.e. pages et blocs) [46].

- Le problème de l'usure de la mémoire flash NAND due aux cycles d'écriture/effacement est compensé par des techniques de répartition de l'usure (*Wear Leveling*). Certaines cellules verront leur durée de vie expirée avant d'autres du fait de la localité spatiale et temporelle des E/S. Différentes techniques existent pour répartir l'usure d'une manière équilibrée sur la totalité de la surface de la mémoire NAND. Ces techniques sont implantées au niveau d'une couche matérielle et logicielle au sein des mémoires à base de flash NAND. Cette couche est appelée la couche FTL (*Flash Translation Layer*) et peut intégrer d'autres fonctionnalités [106]. Il existe d'autres approches pour allonger la durée de vie d'une mémoire flash comme la technique de mise en tampon (*buffering*) des écritures avant de les reporter sur le périphérique physique [105].

- Un autre mécanisme des couches de gestion des mémoires flash qui est le ramasse miettes (*garbage collector*). Son rôle est de recycler l'espace marqué comme invalide à la suite d'une mise à jour de pages. Le ramasse miettes est également un service de la FTL.

- La gestion des erreurs (inversion de bits) est assurée par des algorithmes de détection et/ou correction d'erreurs. Des codes correcteurs d'erreurs peuvent être intégrés au niveau des systèmes de fichiers ou de la couche FTL. Le codage de Hamming [85] peut être implanté dans les couches de gestion de la mémoire flash. Ces informations sont stockées dans la zone des métadonnées des pages.

### 2.2.4 *Intégration des périphériques de stockage dans les centres de données*

Après avoir décrit les principaux types des périphériques de stockage, à savoir les disques durs magnétiques et les mémoires flash, cette partie présente leur intégration dans les centres de données.

Il existe différentes manières d'intégrer les périphériques de stockage au sein des systèmes informatiques. Les topologies diffèrent dans la manière dont le stockage est connecté aux serveurs. Cette partie présente les différentes topologies des systèmes de stockage.

#### 2.2.4.1 *DAS (Direct Attached Storage)*

Cela représente un périphérique de stockage attaché directement à un ordinateur via des interfaces classiques [169]. Un disque dur dans un PC est une forme simple d'un DAS.

#### 2.2.4.2 *NAS (Network Attached Storage)*

Il s'agit d'un périphérique de stockage en réseau doté d'un microprocesseur et d'une mémoire principale, et géré par un système d'exploitation embarqué. Il est attaché à un réseau de communication (e.g. TCP/IP), et accessible via des protocoles de partage de fichiers (e.g. NFS, CIFS, etc). Les requêtes sur les fichiers reçues par le NAS sont interprétées par le système interne en commandes du périphérique de stockage.

#### 2.2.4.3 *SAN (Storage Area Network)*

Un SAN est un sous-réseau dédié au stockage dont les entités sont des périphériques de stockage simples (e.g. baies de disques durs). Comme pour un DAS, les requêtes d'E/S accèdent directement aux périphériques de stockage, en passant par les routeurs, via deux principaux protocoles : iSCSI[1] (*Internet Small Computer System Interface*) ou FC (*Fiber Channel*).

## 2.3 LA PILE LOGICIELLE DES E/S SOUS LINUX

Cette partie présente des notions nécessaires pour la compréhension de l'exécution des E/S sur le système d'exploitation Linux. Avant de décrire les couches logicielles des E/S sous Linux, il est important de définir quelques concepts fondamentaux.

### 2.3.1 *Définitions*

**Définition 2.5.** *(**système de fichiers**) [195] : un système de fichiers est un ensemble de fonctions et de structures de données utilisées par un système d'exploitation pour organiser les fichiers sur disque.*

**Définition 2.6.** *(**mémoire cache**) [183] : une mémoire cache est un composant situé entre la mémoire principale et le microprocesseur. Son objectif est de réduire les latences dues aux accès à la mémoire principale. Le même principe peut être implanté au niveau de la mémoire principale pour réduire les latences dues aux accès disque.*

**Définition 2.7.** *(**inode**) [166] : inode est la contraction de index node qui est une structure décrivant et représentant un fichier dans un système de fichiers.*

Pour accéder aux périphériques de stockage en lecture/écriture, une requête doit traverser différentes couches logicielles avant d'atteindre sa destination. La figure 8 montre la pile logicielle des E/S sur un système Linux.

1 Les protocoles iSCSI et FC sont des adaptations du standard SCSI opérant sur la couche transport de la pile TCP/IP. Ces deux protocoles sont utilisés pour l'accès à distance entre un ordinateur et son système de stockage en réseau.

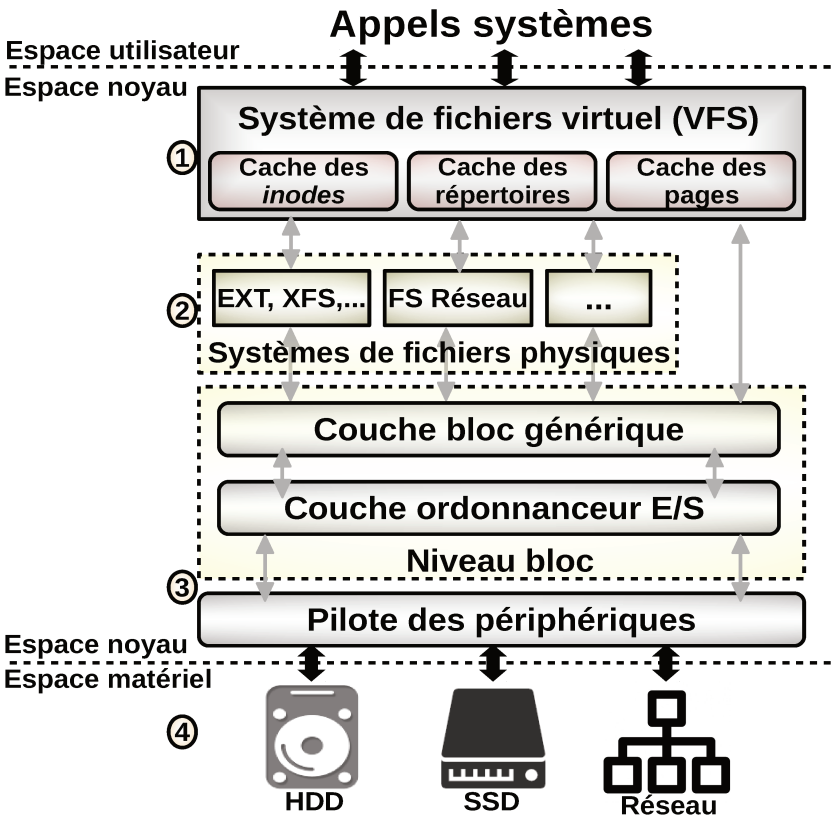


FIGURE 8 : pile logicielle des E/S sous Linux

Les processus exécutent des appels systèmes *open*, *read*, *write* depuis le niveau utilisateur. Les appels systèmes traversent les différentes couches logicielles en passant principalement par :

1. la couche du système de fichiers virtuel VFS ;
2. les systèmes de fichiers physiques ;
3. le niveau bloc et le pilote du périphérique de stockage ;
4. le périphérique de stockage.

La prochaine partie décrit la couche VFS et les systèmes de fichiers physiques. Le niveau bloc est détaillé dans l'annexe (voir partie C.2.3)

### 2.3.2 *Le système de fichiers virtuel Linux (VFS)*

Le système de fichiers virtuel Linux est une couche d'abstraction offrant une interface uniforme pour tous les systèmes de fichiers pouvant exister sous Linux [36]. À travers la couche VFS, un noyau peut monter plusieurs systèmes de fichiers de différents types en même temps. Cette couche permet aux processus du niveau utilisateur d'accéder aux différents systèmes de fichiers (e.g. EXT4 et FAT32), via les mêmes appels systèmes (*open*, *read*, *write*, etc).

Comme le présente la figure 8, la couche VFS n'assure pas que la liaison entre les processus du niveau utilisateur et les systèmes de fichiers physiques, mais contient également des caches tamponnant les données lues et écrites, afin d'augmenter les performances des opérations d'E/S. La couche VFS contient trois systèmes de cache : le cache des *inodes*, le cache des entrées de répertoires, et le cache des pages.

**1) Cache des *inodes* (*inode cache*)** [36] : au cours de l'exécution des appels systèmes sur les fichiers stockés sur les périphériques de stockage, leur *inode* est lu et parfois écrit au niveau de la couche VFS. Afin d'éviter des accès disques, et ainsi accélérer les performances pour chaque opération, la couche VFS maintient les *inodes* récemment accédés au niveau de la mémoire principale à l'aide du cache des *inodes*.

**2) Cache des entrées de répertoires (*Directory Entry Cache* ou *dcache*)** [36] : comme nous l'avons précisé précédemment, la couche VFS reçoit, dans un premier temps, les appels systèmes depuis le niveau utilisateur. Ces appels systèmes prennent comme argument le chemin ou le descripteur du fichier concerné. Afin d'accélérer le processus de recherche du chemin dans les systèmes de fichiers, VFS garde en mémoire un arbre représentant une partie de la structure des répertoires, telle qu'elle est organisée au niveau du périphérique de stockage.

**3) Cache de pages (*page cache*)** [36] : ce cache garde en mémoire principale les pages des fichiers récemment lues/écrites. Lors d'une opération d'E/S sur une page, le noyau cherche la page concernée dans le cache des pages. Si elle y est, le noyau renvoie la page demandée au lieu d'aller la chercher sur le disque.

Lorsqu'une donnée est lue à partir du périphérique de stockage, le *page cache* la garde en mémoire en utilisant un *buffer* circulaire. La lecture anticipée (*read ahead*) est l'un des principaux mécanismes implantés dans le *page cache*. Il consiste à charger en avance un fichier, ou un ensemble de blocs, susceptibles d'être lus dans le futur.

Les algorithmes du *page cache* fonctionnent aussi bien pour les écritures que pour les lectures [194]. Pour les écritures, il existe principalement deux algorithmes : 1) *write-through* et 2) *write-back*. Lors d'une écriture, l'algorithme *write-through* écrit simultanément en mémoire et sur le périphérique de stockage, alors que *write-back* écrit d'abord en mémoire et diffère l'écriture sur disque (e.g. en cas de modification de la donnée).

La majorité des systèmes de fichiers UNIX ont des structures similaires [195], comme les *super blocs*, *inodes*, *blocs de données*, etc. La figure 9 montre la structure de base d'un disque formaté avec un système de fichiers destiné aux systèmes UNIX.

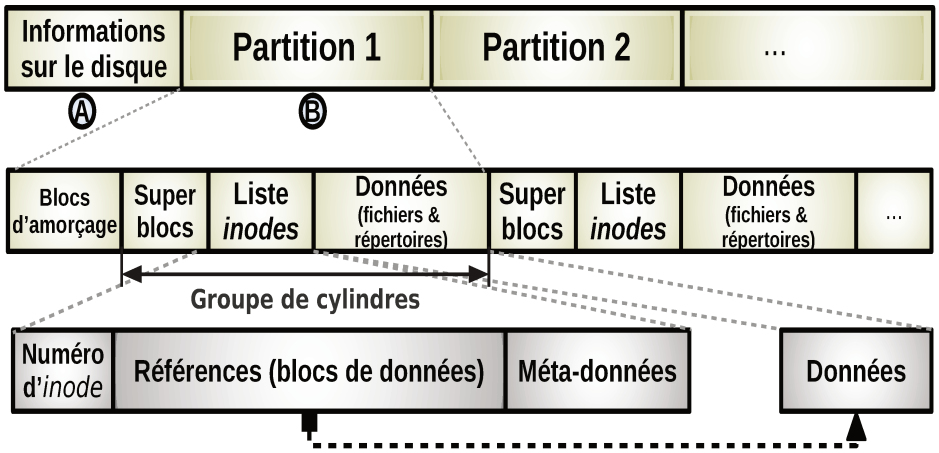


FIGURE 9 : structure des systèmes de fichiers UNIX

Un disque se compose principalement :

- (**A**) d'un espace au début réservé à des informations globales sur le disque ;
- (**B**) d'un ensemble de partitions formatées avec un/plusieurs système(s) de fichiers.

La structure générale d'une partition formatée est la suivante :

1. **Les blocs d'amorçage/démarrage (*boot*)** se trouvent au début du système de fichiers et sont facilement accessible par le BIOS. Les blocs de démarrage de la partition d'amorçage (*bootable*) contiennent le code nécessaire pour l'initialisation du système d'exploitation.

2. **Les super blocs** sont des zones de métadonnées contenant des informations sur le système de fichiers (e.g. la taille du système de fichiers, le nombre de fichiers qu'il peut stocker, les blocs libres/utilisés, etc).

3. **La liste des *inodes*** est implantée sous la forme d'un tableau où les *inodes* sont indexés par leur numéros. Cette liste est souvent associée à une liste de bits afin de maintenir le suivi des accès aux fichiers. Comme le présente la figure 9, chaque *inode* référencé par cette liste contient :
   - le numéro d'*inode* qui représente l'identifiant du fichier dans le système de fichiers ;
   - la liste de références vers les blocs contenant les données du fichier ;
   - les méta-données du fichier (e.g. propriétaire, type, taille, etc).

4. **Les blocs de données** : ce sont les blocs stockant les données des fichiers [166]. Ces blocs sont pointés par les références des *inodes* et sont alloués sur demande. Les blocs de données d'un fichier peuvent être éparpillés sur le système de fichiers, on parle alors de fragmentation du fichier. Notons que

les blocs de données occupent le plus grand espace sur le périphérique de stockage.

Le tableau 1 montre les systèmes de fichiers développés dans les systèmes Linux. Les systèmes de fichiers EXT sont les plus utilisés actuellement.

## 2.4 *benchmarks* DESTINÉS AUX SYSTÈMES DE FICHIERS

Afin de modéliser les performances des systèmes de stockage, de nombreux *benchmarks* ont été proposés et utilisés dans la littérature.

Avant de présenter les *benchmarks* couramment utilisés pour les systèmes de fichiers, nous allons d'abord définir les principaux types de *benchmarks*.

### 2.4.1 *Types des benchmarks*

Un *benchmark* est une application utilisée pour étudier le fonctionnement d'un composant système et également d'en mesurer les performances. Il existe deux principaux types de *benchmark* [200] : a) les *micro-benchmarks*, et b) les *macro-benchmarks*.

#### 2.4.1.1 *Micro-benchmark*

Un *micro-benchmark* est un outil utilisant des charges synthétiques, destinées à tester un type particulier d'opération (e.g. requêtes de base de données, E/S des systèmes de fichiers, instructions CPU) [78].

L'objectif est donc de comprendre le comportement d'un composant en exécutant des opérations élémentaires. Les tests réalisés par un *micro-benchmark* sont facilement reproductibles, car le nombre de composants impliqués est limité. Voici quelques exemples de *micro-benchmark* :

- *benchmark* pour le CPU : *UnixBench* [87], *SysBench* [110] ;
- *benchmark* pour la RAM : *lmbench* [132] ;
- *benchmark* pour les systèmes de fichiers : *IoZone* [144], *fio* [43], *Bonnie* [53] ;
- *benchmark* pour les réseaux : *iperf* [62].

#### 2.4.1.2 *Macro-benchmark*

Un *macro-benchmark* est un outil simulant des charges d'applications réelles [78]. Autrement dit, les *macro-benchmarks* se basent sur une caractérisation des systèmes réels et des environnements de production afin de déterminer le modèle

| Système de fichiers | Description | Taille Max. FS | Taille Max. fichier | Taille Max. nom de fichier |
|---|---|---|---|---|
| **S5FS** | Faisant référence à *Unix System V File System*, S5FS est le premier systèmes de fichiers Unix (paru en 1974). Il est constitué d'un super bloc, des inodes, suivis des blocs de données. | 2 Go | 2 Go | 14 caractères |
| **UFS** | Appelé aussi FFS (*berkeley Fast File System*), c'est le successeur de S5FS dans les systèmes Unix. Les systèmes EXT2 et EXT3 sont essentiellement basés sur UFS. | 8 Zo | 8 Zo | 255 caractères |
| **EXT** | Implanté en 1992, EXT est le premier système de fichiers développé exclusivement pour le noyau Linux et intégré dans sa version standard 0.96c. | 2 Go | 2 Go | 255 caractères |
| **EXT2** | Il a été conçu et implanté pour résoudre certain problèmes de EXT[42], comme le support de bloc de tailles variables et l'extensibilité d'un système de fichiers en cours d'exécution du noyau. | 4 To | 2 Go | 255 caractères |
| **EXT3** | En plus de toutes les fonctionnalités de EXT2, EXT3 implante la journalisation pour réduire le temps de récupération des données en cas d'un crash système. | 32 To | 2 To | 255 caractères |
| **EXT4** | Diffusé en 2006 avec le noyau Linux 2.6.19. EXT4 a résolu le problème de passage à l'échelle de EXT3. | 1 Eo | 16 To | 255 caractères |

TABLE 1 : systèmes de fichiers Linux

à évaluer. Le passage de la caractérisation d'une charge à son modèle est souvent assuré par des approches statistiques et/ou probabilistes [190] (e.g. files d'attentes et chaînes de Markov).

L'objectif des *macro-benchmark* est d'étudier le comportement d'un composant particulier sous des conditions d'utilisation habituelles. Le choix d'un *macro-benchmark* se base sur le type d'application visée. Par exemple, TPC-C et TPC-H reproduisent le comportement des charges des bases de données [56], tandis que *Filebench* reproduit la charge des serveurs de production (web, mail, vidéo, proxy, etc).

Après avoir défini les deux types de *benchmarks*, nous présentons quatre *benchmarks* (2 *micro-benchmarks* et 2 *macro-benchmarks*) fréquemment utilisés dans les études des performances des E/S et des systèmes de stockage.

### 2.4.2 *Iozone*

*Iozone* est un *micro-benchmark* de système de fichiers [144]. Il génère différentes opérations d'E/S (e.g. *read, write, aio_read, aio_write, mmap*, etc), et produit un rapport sur les mesures de performances de ces opérations.

### 2.4.3 *fio*

*fio* est l'acronyme de *flexible* I/O. Il s'agit d'un *micro-benchmark* de système de fichiers développé par *Jens Axboe*, developpeur de la couche bloc de *blktrace* [19]. *fio* a été initialement développé afin de tester et de vérifier les mises à jour sur la pile logicielle d'E/S Linux. La diversité des paramètres de contrôle de *fio* permet de construire une variété de scénarios de charge d'E/S. *fio* supporte une vingtaine de mécanismes d'E/S regroupant chacun un ensemble d'appels systèmes (e.g. *libio* pour *aio_read* et *aio_write*, *sync* pour *read* et *write*, etc). La méthode conventionnelle pour utiliser *fio* est de construire un fichier décrivant un/plusieurs scénario(s) de *benchmarking*.

### 2.4.4 *Filebench*

*Filebench* est un *macro-benchmark* de système de fichiers et de systèmes de stockage [187]. *Filebench* permet de générer une variété de charges d'E/S prédéfinies ou personnalisables. Ces charges simulent l'exécution d'applications complexes telles que des serveurs web, mail, ou bases de données. De nouvelles charges de travail peuvent être proposées par les utilisateurs en utilisant son langage de modélisation WML (*Workload Model Language*).

### 2.4.5 *Postmark*

*Postmark* est un *macro-benchmark* conçu pour reproduire le comportement d'un serveur mail [101]. Le scénario d'exécution de *Postmark* peut se résumer en trois phases principales :

1. la création d'un ensemble de fichiers de différentes tailles ;
2. l'exécution des lectures et des écritures sur les fichiers initialement créés ;
3. la suppression des fichiers après la fin de toutes les E/S.

Le choix des E/S à exécuter et les fichiers correspondant est aléatoire, afin de minimiser l'impact des caches système, de la lecture anticipée des fichiers (i.e. *file read ahead*), ainsi que les caches disques. Un rapport détaillé contenant des statistiques et des mesures de performances est produit à la fin de chaque exécution.

Après avoir présenté les concepts liés aux systèmes de stockage, aux E/S sous Linux, et aux outils de *benchmarking*, la partie suivante introduit les notions fondamentales concernant le *cloud computing*.

## 2.5 LE *cloud computing*

### 2.5.1 *Définitions*

**Définition 2.8.** *(**cloud computing**) [134] : selon la définition du NIST (National Institute of Standards and Technology) [134], le cloud computing est un modèle permettant un accès pratique, omniprésent, et à la demande d'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.*

**Définition 2.9.** *(SLA) [13] : le SLA (Service Level Agreement) est un document qui inclut la description des services accordés par le fournisseur du cloud, les différents paramètres des services, les garanties, et les actions à prendre en cas de violation. L'objectif principal du SLA est de donner une définition claire des accords sur les termes des services tel que les performances, la disponibilité, et la facturation.*

Après avoir défini les concepts fondamentaux, la prochaine partie présente les différents services du *cloud*.

### 2.5.2 *Architecture et types des services cloud*

Les services offerts par le *cloud* peuvent être divisés en trois catégories principales [118] :

- SaaS "*Software as a Service*" ;
- PaaS "*Platform as a Service*" ;
- IaaS "*Infrastructure as a Service*".

La figure 10 décrit les services offerts par le *cloud computing* :


FIGURE 10 : pile des services cloud [118]

- IaaS est le service *cloud* qui offre une abstraction des ressources matérielles en utilisant les technologies de virtualisation. Trois types de ressources matérielles sont virtualisées : 1) le calcul, 2) le réseau, et 3) le stockage. Ce service peut être vu comme un service de location de ressources (CPU, RAM, HDD/SSD) pour le déploiement des environnements virtualisés.
- PaaS est le service qui fournit aux utilisateurs un environnement de programmation (Django pour Python, Symfony pour PHP, etc) et un environnement d'exécution (Google's App Engine, Microsoft Azure, etc). Un exemple de PaaS est l'hébergement des sites web.

- SaaS offre aux utilisateurs des applications prêtes à être utilisées sans configuration ni modification. Plusieurs exemples d'applications SaaS peuvent être cités : bureautique (Google Docs, Microsoft Office Live, etc), stockage (Dropbox, iCloud, pCloud).

Comme nous l'avons précisé auparavant (voir partie 1.1), le service *cloud* IaaS consiste à proposer l'utilisation des infrastructures de calcul et de stockage sous forme de VM. La prochaine partie propose un tour d'horizon des types et outils de virtualisation les plus utilisés actuellement.

## 2.6 VIRTUALISATION DES RESSOURCES MATÉRIELLES

**Définition 2.10.** *(**virtualisation**) [113] : la virtualisation est représentée par une couche d'abstraction et d'isolation située entre les applications et les ressources matérielles utilisées. La virtualisation permet d'améliorer les points suivants : la performance, le passage à l'échelle, la fiabilité/disponibilité, la flexibilité.*

*D'un point de vue matériel, la virtualisation est le fait d'exécuter plusieurs systèmes d'exploitation sur une même PM, par le biais d'un jeu d'instructions supplémentaires du microprocesseur (e.g. VT-x d'intel, AMD-V de AMD, etc).*

*Partant de cette définition, la virtualisation nécessite un moyen qui peut assurer les tâches élémentaires suivantes :*

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

Dans le domaine du *cloud*, les outils mettant en œuvre l'isolation sont appelés ***isolateurs*** ou ***conteneurs***. Dans la virtualisation des ressources matérielles, le moyen permettant d'assurer l'abstraction et l'isolation est appelé ***hyperviseur*** ou plus généralement ***gestionnaire de VM*** (VMM pour *Virtual Machine Manager/Monitor*). La figure 11 montre le principe de l'isolation et les types d'hyperviseurs, à savoir l'hyperviseur *bare metal*, et l'hyperviseur *hosted*.

Ces trois techniques permettent à plusieurs applications de cohabiter dans le même système, et ainsi de partager les ressources comme si chaque application était seule sur le système.

### 2.6.1 *Isolateur*

Appelé aussi virtualisation au niveau du système d'exploitation (*OS-level virtualization*) [76], l'isolation est un principe utilisé principalement par les systèmes

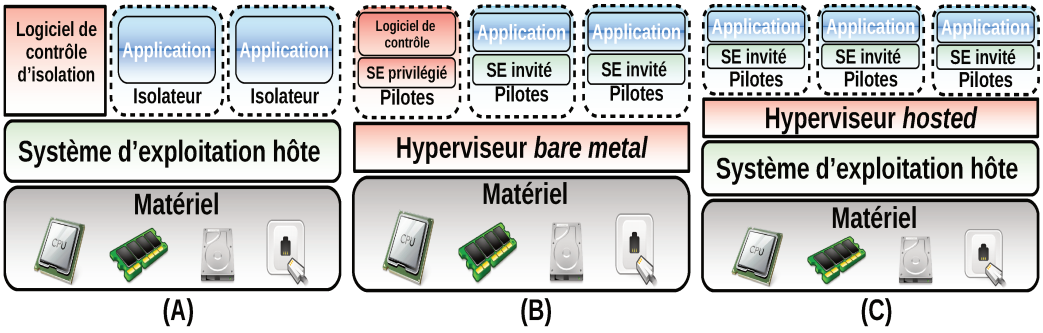


FIGURE 11 : types de virtualisation des ressources matérielles

UNIX. L'isolation consiste à utiliser des fonctionnalités pour créer des contextes/zones d'exécution pour chaque application/utilisateur (voir figure 11 (A)). Parmi les outils d'isolation existants, nous pouvons citer *OpenVZ*, *Linux-Vserver*, ou *LXC* (Linux). Il existe également d'autres solutions telles que *jail* (basée sur FreeBSD), Sandboxie (basée sur Windows). Les conteneurs Docker (une version étendue de LXC) représentent actuellement une solution de plus en plus utilisée dans le domaine du *cloud* [27].

### 2.6.2 *Hyperviseur bare metal*

Ce type d'hyperviseur s'exécute directement sur la plateforme matérielle du système, sans l'intermédiaire d'un système d'exploitation (voir figure 11 (B)). Ce type de virtualisation est appelé *bare-metal*, à cause de l'absence d'une couche qui sépare l'hyperviseur du matériel. La gestion et l'allocation des ressources matérielles sont assurées par un système d'exploitation minimaliste accompagnant l'hyperviseur. Parmi les hyperviseurs les plus utilisés actuellement, nous pouvons citer Xen [20], VMware ESX [141], ou Hyper-V [188].

### 2.6.3 *Hyperviseur hosted*

Comme l'illustre la figure 11 (C), un hyperviseur de type *hosted* fait le lien entre le système hôte et les systèmes invités. Le système hôte est le système d'exploitation utilisé par la PM, et qui se charge de la gestion et de l'allocation des ressources physiques. Les systèmes invités sont les systèmes d'exploitations utilisés par les VM. Parmi les hyperviseurs de type *hosted* les plus connus, nous pouvons citer KVM/QEMU (KVM pour la virtualisation [149] et QEMU pour l'émulation [23]), VirtualBox [149], VMware Workstation [192].

Les outils de gestion du *cloud* IaaS doivent faire face à l'hétérogénéité des techniques et outils de virtualisation dans un même centre de données. Par exemple, OpenStack peut gérer simultanément des serveurs avec différents types/outils de virtualisation (e.g. Xen, ESX, KVM, LXC, etc) [148].

La bibliothèque *libvirt* se présente comme une solution pour faire face à l'hétérogénéité et la gestion des VM d'une manière uniforme. Dans la suite nous présentons cette bibliothèque.

### 2.6.4 *La bibliothèque libvirt*

*libvirt* est une API et un ensemble d'outils permettant la gestion des environnements virtualisés. La figure 12 montre une application basée sur *libvirt* pour gérer des opérations de virtualisation.

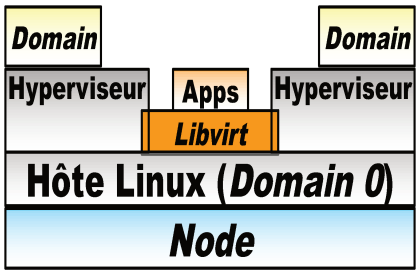


FIGURE 12 : application de gestion basée sur *libvirt*

Comme le montre la figure 12, *libvirt* permet de construire des applications permettant l'accès et la gestion des *hyperviseurs*.

*libvirt* offre une couche d'abstraction pour plusieurs types d'hyperviseurs. Elle a été conçue initialement comme un *wrapper* pour l'hyperviseur Xen, puis a été étendue pour supporter une multitude d'hyperviseurs [30].

La partie suivante donne un aperçu sur le principe de stockage secondaire virtuel, sur lequel notre travail est basé.

### 2.6.5 *Stockage de VM et disques virtuels*

Comme nous l'avons introduit dans la précédente partie, le principe de la virtualisation consiste à abstraire des ressources physiques dont le stockage. Les VM doivent disposer d'un moyen pour stocker séparément leurs données (e.g. le système de fichiers du système d'exploitation invité, les données des applications). Parmi les solutions de stockage possibles, il existe le stockage :

- **local** qui s'appuie sur les périphériques de stockage du système hôte ;
- **partagé** de type SAN ou NAS via des protocoles dédiés (voir partie 2.2.4) ;
- **réparti** qui peut être basé sur les deux premiers, et utilise des systèmes de fichiers dédiés (e.g. GlusterFS, HDFS, Ceph, etc).

Dans la majorité des cas, une VM utilise un ou plusieurs fichiers pour stocker ses données dans le système de la PM. Ces fichiers sont appelés *disques virtuels*

ou *images disques*, car ils jouent le rôle d'un disque physique pour les VM. Nous donnons plus de détails sur les *disques virtuels* en annexe (voir partie C.1.2).

Nous avons introduit l'ensemble des notions fondamentales et essentielles pour aborder chacune de nos contributions. La prochaine partie présente le modèle MAPE-K.

## 2.7 LE MODÈLE MAPE-K

Le modèle MAPE-K a été initialement introduit par IBM en 2001 par Paul Horn [90], et officiellement publié par Jeff Kephart et *al.*[103] en 2003 sous le nom de l'*Autonomic Computing*. L'objectif de l'informatique autonomique (*Autonomic Computing*) est de construire des systèmes informatiques capables de garantir leur propre gestion et administration. Le constat de départ est que les systèmes informatiques deviennent de plus en plus complexes, ce qui complique leur gestion manuelle. Partant de ce constat, les systèmes autonomiques doivent garantir la prise en charge de la gestion des différents éléments du système, tout en limitant au maximum l'intervention d'un acteur humain.

L'administrateur définit au départ les fonctions et les modèles qui doivent être suivis par chaque composant constituant le système autonomique. La figure 13 présente le modèle MAPE-K.

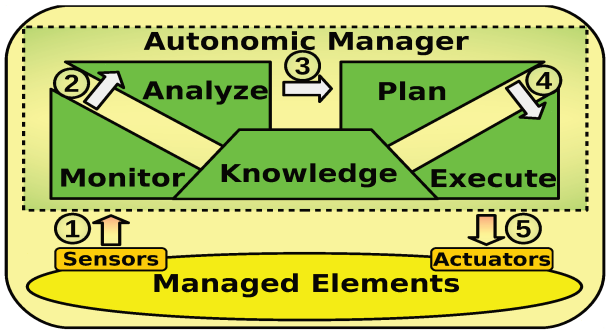


FIGURE 13 : modèle de contrôle d'adaptation MAPE-K (extrait de [103])

Comme le montre la figure 13, un système autonomique est constitué d'un ensembles d'éléments à gérer (*Managed Elements*), et d'un gestionnaire autonomique (*Autonomic Manager*). Les éléments gérés constituent l'ensemble des composants matériels et/ou logiciels auxquels le principe d'auto-adaptation est appliqué. Ce modèle est incorporé dans le gestionnaire autonomique.

Les éléments gérés interagissent avec le gestionnaire autonomique via les capteurs et les actionneurs.

Les **capteurs (*sensors/probes*)** se chargent de la collecte des valeurs d'un ensemble de métriques. Ces métriques sont définies par l'acteur humain, et doivent représenter l'état des éléments gérés. Les métriques remontées par les capteurs alimentent le gestionnaire autonomique (transition 1 figure 13).

Les **actionneurs (*actuators*)** exécutent les actions reçues du gestionnaire autonomique (transition 5 figure 13). Les changements à effectuer sont élaborés grâce aux métriques précédemment collectées, et en fonction des étapes définissant le modèle MAPE-K.

Le gestionnaire autonomique prend comme entrée les métriques remontées par les capteurs (transition 1 figure 13). Le gestionnaire autonomique est composé de quatre étapes et une base de connaissance. Les étapes sont : 1) *Monitor*, 2) *Analyze*, 3) *Plan*, et 4) *Execute*. La base de connaissance est appelée *Knowledge*.

***Monitor (M)*** : cette étape permet de récupérer et de traiter les valeurs des métriques remontées par les capteurs, en les rendant compréhensibles pour la prochaine étape du gestionnaire autonomique. Cette phase est essentielle dans le processus d'auto-adaptation, car c'est le moyen utilisé par le gestionnaire autonomique pour acquérir une connaissance détaillée du système.

***Analyze (A)*** : cette étape se charge d'analyser l'état actuel du système en utilisant les données produites par l'étape *Monitor* (transition 2 figure 13). L'analyse de l'état du système s'effectue selon des modèles prenant en compte les objectifs que nous cherchons à atteindre, ainsi que les résultats de l'étape précédente. À la sortie de cette phase, le gestionnaire autonomique doit fournir un bilan détaillé de l'état actuel du système.

***Plan (P)*** : l'étape *Plan* prend en entrée les résultats de l'étape *Analyze* (transition 3 figure 13), puis décide si des changements doivent avoir lieu. Dans le cas d'un processus d'optimisation, l'étape *Plan* doit disposer de la configuration optimale souhaitée. Cette dernière est comparée avec la configuration actuelle afin de déterminer les opérations à appliquer.

***Execute (E)*** : après l'élaboration de la nouvelle configuration du système, l'étape *Execute* applique le résultat obtenu à l'issu de l'étape *Plan* (transition 4 figure 13). La nouvelle configuration obtenue à l'aide de l'étape *Plan* est appliquée par l'étape *Execute* via les actionneurs gérant le système (transition 5 figure 13).

***Knowledge (K)*** : la base de connaissance *Knowledge* s'enrichit au fur et à mesure de l'exécution du modèle MAPE-K. Celle-ci regroupe des informations provenant de toutes les étapes, et aussi des acteurs humains. Par exemple, la base de connaissance peut détenir un historique de l'étape *Monitor*, des analyses préalablement faites dans l'étape *Analyze*, ou des configurations déjà effectuées dans l'étape *Plan*.

## 2.8 CONCLUSION

Ce chapitre a introduit les connaissances et outils essentiels pour la présentation de nos contributions. Nous avons résumé les principes de base concernant les systèmes de stockage, notamment le stockage à base de HDD et de SSD. Nous avons décrit les E/S sous Linux et les *benchmarks* fréquemment utilisés pour les systèmes de stockage et les E/S. Nous avons également présenté les principaux concepts du *cloud* et de la virtualisation des ressources matérielles. Le modèle MAPE-K et l'ensemble de ses étapes ont été également détaillées dans ce chapitre. Le prochain chapitre liste et discute les travaux de l'état de l'art dans les domaines de nos contributions.

# Chapitre 3

# ÉTAT DE L'ART

Cette partie présente les travaux de l'état de l'art liés à nos contributions. Cet état de l'art est organisé en suivant le modèle MAPE-K. Premièrement, nous présentons les outils et travaux concernant le *monitoring* des opérations d'E/S. Dans un second temps, nous nous intéressons aux approches de caractérisation des charges d'E/S. La troisième partie résume les modèles de coûts existants. La dernière partie présente les approches d'optimisations de placement de données et de VM.

**Sommaire**

## 3.1 MONITORING : SURVEILLANCE DES ACTIVITÉS D'E/S

Dans ce travail de thèse, nous proposons un outil de *monitoring* des E/S des VM. Dans le modèle MAPE-K, le *Monitoring* consiste en la surveillance des activités E/S des VM vis-à-vis du système de stockage. La présente partie donne un aperçu des outils de *monitoring* classés en deux catégories. La première partie présente des traceurs génériques, pouvant être utilisés en particulier pour surveiller les E/S. La seconde partie montre les outils de trace spécifiques aux E/S.

### 3.1.1 *Traceurs génériques*

***SystemTap*** est un logiciel libre qui permet de collecter des informations sur un système Linux en cours d'exécution. L'utilisation de *SystemTap* est basée sur des scripts. Un script *SystemTap* est exécuté au niveau du noyau Linux afin de collecter et de traiter les données extraites pour une/plusieurs fonctions noyau. Les étapes d'exécution de *SystemTap* peuvent être résumées comme suit :

- convertir le script `SystemTap` en un programme C ;
- compiler le programme C obtenu en tant que module noyau Linux ;
- insérer le module et le charger dans le noyau ;
- collecter les données sur les fonctions tracées et les transférer dans l'espace utilisateur pour traitement et affichage.

***ftrace*** fait référence à *function tracer*, qui est un traceur interne intégré dans le noyau Linux depuis la version 2.6.27. Il a été initialement développé dans le but d'aider les développeurs du noyau à comprendre le comportement du système à partir de l'espace utilisateur, sans pour autant modifier les fonctions du noyau. Même si *ftrace* est considéré comme un outil de trace, il s'agit d'un environnement incluant plusieurs outils de traces et de traitement de données. *ftrace* utilise le système de fichiers *debugfs* afin d'enregistrer les fichiers de contrôle de *ftrace* et permet l'accès aux fichiers de sortie.

La manipulation de *ftrace* en utilisant `debugfs` peut s'avérer difficile. *trace-cmd* est un outil fournissant une interface d'utilisation de *ftrace* en ligne de commande. Il existe également l'outil *kernelshark* qui permet de visualiser graphiquement le fichier de trace généré par *trace-cmd*.

***perf*** est un outil de trace et de profilage intégré dans le noyau Linux. Il permet de tracer non seulement les fonctions noyau, mais également des applications utilisateurs en dehors du contexte du noyau.

***LTTng*** l'acronyme fait référence à *Linux Trace Toolkit next generation*. *LTTng* est un outil open source destiné à tracer les fonctions du noyau Linux, des applications utilisateurs et des bibliothèques. Il est composé de deux principaux composants : 1) des modules noyau Linux (pour la trace du noyau), et 2) des bibliothèques dynamiquement chargeables (pour la trace des applications utilisateurs).

***Kprobes et Jprobes :*** à la différence des outils présentés auparavant, *Kprobe* (*Kernel probe*) et *Jprobes* (*Jumper probe*) ne sont pas des outils à part entière. *Kprobe* et *Jprobe* sont des mécanismes de débogage du noyau Linux, qui peuvent être utilisés pour surveiller les événements dans un environnement de production. *Kprobe* a été développé par IBM comme un mécanisme sous-jacent de *Dprobe*. L'outil *Dprobe* offre plus de fonctionnalités avec un langage de script pour écrire les *handlers* des sondes (*probes*). Finalement, seul *kprobe* a été intégré dans le noyau standard Linux.

La figure 14 montre un schéma simplifié du fonctionnement de *Kprobe*.

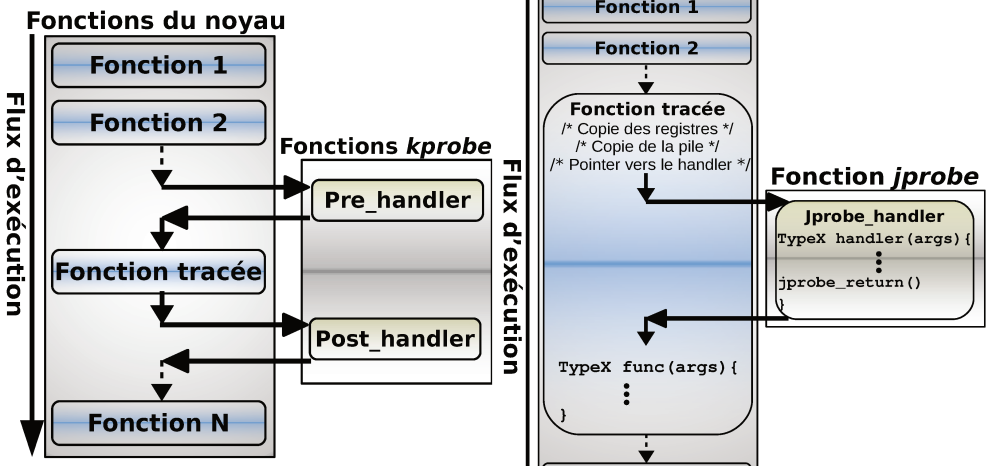


FIGURE 14 : fonctionnement *Kprobe*




FIGURE 15 : fonctionnement *Jprobe*

Les informations indispensables pour *Kprobe* sont les suivantes : l'adresse de la fonction à tracer, la fonction exécutée avant l'appel de la fonction tracée (i.e. `Pre_handler`), et la fonction exécutée après l'exécution de la fonction tracée (i.e. `Post_handler`).

Après avoir placé les *probes* et défini les *handlers*, il faut compiler le code produit en tant que module en utilisant les sources du noyau Linux. Le problème majeur du *Kprobe* est son incapacité à manipuler les arguments et la valeur de retour de la fonction sondée. C'est pour cette raison que le mécanisme *Jprobe* a été développé.

*Jprobe* utilise un principe de *mirroring* (i.e. copie des registres de la fonction et de la pile) afin de permettre un accès transparent aux arguments de la fonction

tracée. Contrairement à *Kprobe*, *Jprobe* utilise une seule fonction *handler*. Cette fonction doit avoir la même signature que celle de la fonction sondée (i.e. liste des arguments et type de retour). La fonction *handler* doit se terminer par un appel de la fonction `jprobe_return()`. Cet appel permet à la fonction sondée de poursuivre son exécution. La figure 15 montre le principe simplifié de fonctionnement de *Jprobe*.

### 3.1.2 *Traceurs spécifiques*

***strace*** est un outil permettant de tracer les appels systèmes et les signaux envoyés et reçus par un processus en cours d'exécution.

L'outil *strace* est essentiellement basé sur l'appel système *ptrace*. *ptrace* permet à un processus (parent) de contrôler l'exécution d'un autre processus et d'éditer son image mémoire. La figure 16 (A) montre le niveau de trace de *strace*.

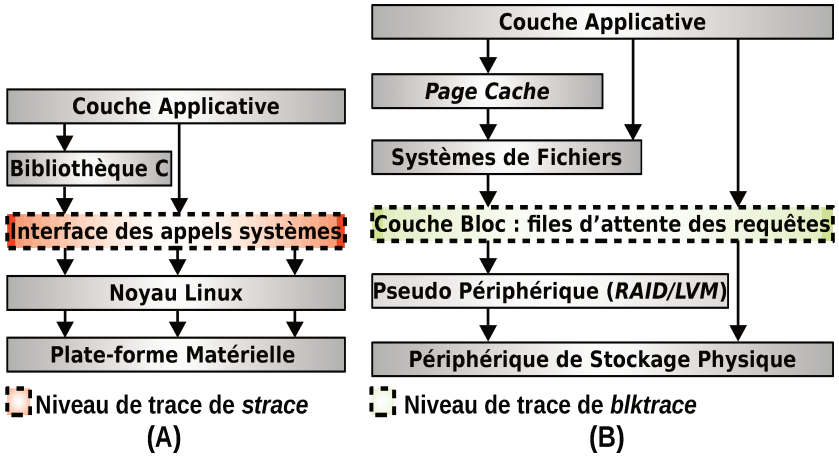


FIGURE 16 : niveaux de trace de *strace* et *blktrace*

Comme le montre la figure 16 **(A)**, *strace* opère au niveau de l'interface des appels systèmes de la pile des E/S.

***blktrace*** est un outil de trace open source, fournissant des informations détaillées sur les opérations d'E/S disque au niveau bloc. La figure 16 **(B)** montre un schéma simplifié du niveau de trace ciblé par *blktrace*. Cette figure montre le niveau de trace de *blktrace* qui se situe entre le système de fichiers et les périphériques de stockage. *blktrace* offre trois composants :

- *un patch noyau Linux* permettant d'insérer des sondes au niveau des fonctions de la couche bloc d'E/S du système Linux. Pour les versions récentes du noyau (à partir de 2.6.17-rc1), le *patch* est déjà intégré ;

- *blktrace* qui se charge de transférer la trace non formatée des évènements à partir de l'espace noyau à l'espace utilisateur, en utilisant le système de fichiers *debugfs* ;
- *blkparse* qui permet de formater les traces obtenues avec *blktrace*, stockées dans des fichiers ou directement en ligne en redirigeant la sortie de *blktrace* vers *blkparse*.

### 3.1.3 *Discussion sur les outils de monitoring des E/S*

Il existe différents outils pouvant être utilisés pour le *monitoring* des E/S dans le système Linux. Nous avons listé les outils les plus fréquemment utilisés dans Linux, et nous les avons regroupé en deux catégories. La première catégorie réunit des outils de *monitoring* génériques pouvant être configurés pour surveiller des fonctions d'E/S. La deuxième catégorie correspond aux outils destinés au *monitoring* des E/S sous Linux.

La généricité des outils de la première catégorie a des avantages et des inconvénients. Ces outils offrent une flexibilité dans la mise en œuvre, mais nécessitent une expertise pour leur utilisation. Dans notre cas par exemple, il faut disposer d'une connaissance préalable sur les fonctions à surveiller avant de pouvoir utiliser ces outils. Cela représente un travail fastidieux et coûteux en terme d'exploration.

Les outils de la deuxième catégorie ont été conçus pour le *monitoring* des E/S sous Linux. Néanmoins, chaque outil opère à un niveau spécifique de la pile des E/S Linux, tandis que nous avons besoin de tracer le chemin parcouru par les opérations d'E/S en partant d'une VM jusqu'au périphérique de stockage physique.

## 3.2 MONITORING : CARACTÉRISATION DES CHARGES - D'E/S DES VM

Comme deuxième composant de l'étape de *monitoring*, nous proposons la phase de caractérisation des charges d'E/S. De nombreux travaux ont déjà caractérisé les charges d'E/S, exécutées sur un système de stockage [163] [165] [164] [11] [83] [82] [84] [38]. Nous pouvons organiser ces travaux en deux catégories. La première regroupe des approches indépendantes du contexte du *cloud*. Dans la deuxième catégorie, les travaux proposés ciblent les environnements virtualisés. Cette partie présente les deux catégories, puis une discussion concernant ces approches.

### 3.2.1 *Caractérisation généraliste des charges d'E/S*

Dans la littérature, les approches de caractérisation des charges d'E/S se basent sur l'analyse des traces réelles des périphériques ou des systèmes de stockage utilisés dans des environnements de production. Parmi les approches de référence dans ce domaine, nous pouvons citer celles de A. Riska et *al.* [163] [165] [164]. Leurs propositions sont basées sur une étude des *logs* d'un ensemble de disques durs, utilisés dans différents environnements. Ces travaux montrent que les caractéristiques liées aux performances, telles que le taux d'arrivée des requêtes, la latence, les performances des opérations de lecture/écriture, dépendent fortement de l'environnement (serveurs, PC personnel, etc), tandis que le taux de lecture/écriture, le profil d'accès séquentiel/aléatoire dépendent de l'applicatif (serveur mail, calcul scientifique, traitement de texte, etc) [163]. Concernant ces travaux, deux points essentiels sont à retenir :

- ils exploitent une analyse hors ligne des traces des périphériques de stockage (i.e. après la fin de leur service) ;
- les résultats obtenus sont des modèles représentant les profils E/S des périphériques de stockage selon les environnements dans lesquels ils ont été utilisés.

### 3.2.2 *Caractérisation des charges d'E/S et virtualisation*

Les environnements virtualisés peuvent influencer les charges d'E/S [70] [83]. En effet, les opérations d'E/S peuvent avoir différents comportements vis-à-vis des systèmes de stockage, et cela à cause des couches séparant ces derniers des applications exécutées par les VM.

Les travaux visant la caractérisation de charges d'E/S dans les environnements virtualisés partagent plusieurs points communs. Dans [11] [83] [82] [84], les auteurs proposent et utilisent des outils permettant de caractériser en ligne une charge d'E/S dans un environnement virtualisé. Les outils ont été implantés au niveau de l'hyperviseur VMware ESX. Tous ces travaux conduisent au même modèle de charges d'E/S (étude du profil d'accès, tailles de requêtes, temps d'inter arrivée des requêtes, etc). Pour résumer, ces approches :

- caractérisent les charges d'E/S dans un environnement virtualisé ;
- opèrent au niveau de l'hyperviseur ;
- spécifient des fonctions identiques des charges d'E/S.

### 3.2.3 *Discussion sur la caractérisation des E/S*

Il existe donc de multiples travaux traitant la caractérisation des charges d'E/S. La première partie regroupe des approches caractérisant les charges d'E/S à partir des historiques d'utilisation des périphériques de stockage. Ces approches ne répondent pas à tous les besoins de notre étude. Par rapport aux besoins de notre étude, ces approches présentent les contraintes suivantes :

- elles modélisent les profils E/S par périphérique de stockage, tandis que nous cherchons à optimiser le placement des VM, et donc nous avons besoin des profils d'E/S par VM ;
- pour caractériser les charges d'E/S, ces approches nécessitent une étude hors ligne et à long terme (i.e. durée de service des périphériques de stockage). Afin de faire face aux changements fréquents de comportement des VM, nous avons besoin de caractériser les charges d'E/S des VM en ligne et à court terme ;
- la caractérisation est basée sur des périphériques de stockage utilisés pour un type d'application spécifique, connu au préalable (e.g. serveurs web, bases de données, calcul scientifique). Notre caractérisation doit être en mesure d'ignorer le type d'application pouvant être exécuté par les VM.

La deuxième catégorie des approches de caractérisation est mieux adaptée à notre étude. Dans notre étude, la caractérisation des charges d'E/S doit s'effectuer au niveau du système hôte, afin d'avoir un profil d'E/S pour chaque VM. Ceci implique une analyse à différents niveaux des PM, tout en considérant les VM comme des boites noires. Pour ces raisons, nous nous sommes inspirés de [11] [83] [82] [84]. Néanmoins, ces travaux visent un niveau particulier (i.e. souvent l'hyperviseur) pour capturer les E/S et ainsi caractériser les charges d'E/S. Le fait de se restreindre à un seul niveau peut conduire à une mauvaise caractérisation et donc à des résultats erronés. Les paramètres utilisés peuvent avoir différentes valeurs selon le niveau adressé. Cela mène à une perte d'information d'un niveau à un autre. Il est donc nécessaire de suivre le chemin des opérations d'E/S des VM sur l'ensemble des niveaux du système.

Une fois les charges d'E/S des VM caractérisées et les profils E/S des VM dressés, l'étape de *Monitoring* est achevée. La seconde étape comporte l'analyse des coûts d'exécution des charges d'E/S sur les périphériques de stockage, ce que nous présentons dans la partie suivante.

## 3.3 ANALYZE : MODÉLISATION DES COÛTS DU STOCKAGE DES VM

Cette partie concerne l'étape *Analyze* du modèle MAPE-K. Nous y menons une discussion sur les travaux proposant des modèles de coût du stockage de données dans un environnement *cloud* IaaS.

Nous avons classé les travaux sur les modèles de coût en quatre catégories en fonction des paramètres pris en compte (voir tableau 2). La première colonne regroupe des approches proposant des modèles de coût prenant en compte les VM. La seconde colonne réunit des travaux considérant la consommation énergétique. La troisième colonne rassemble des travaux dédiés aux périphériques de stockage. La dernière colonne correspond aux approches se focalisant sur des modèles de coût dans le contexte du *Cloud computing* (i.e. qui inclut les SLA et le coût de migration des VM).

| **Références** | **Machine Virtuelle** | **Énergie** | **Stockage** | ***Cloud computing*** |
|---|---|---|---|---|
| [29][100][176] | ✓ | ✓ | ✓ | ✗ |
| [54][91][107][111][177] | ✓ | ✓ | ✗ | ✗ |
| [71][204] | ✗ | ✗ | ✓ | ✓ |
| [108][199][33] | ✗ | ✓ | ✓ | ✓ |
| [115][112][156] | ✓ | ✓ | ✗ | ✓ |
| [122][198] | ✗ | ✓ | ✓ | ✗ |

TABLE 2 : modèles de coût. ✓ signifie que les références (en ligne) prennent en compte les contraintes (en colonne), et ✗ signifie et que les références (en ligne) ne prennent pas en compte les contraintes (en colonne).

### 3.3.1 *Modèles de coût de l'énergie et des VM*

Dans [112], Kurpicz et *al.* proposent un modèle basé sur le calcul de la consommation énergétique des VM. Ce modèle considère que l'énergie totale consommée par un centre de données, peut être divisée sur l'ensemble des VM. Même si la motivation qui sous-tend la proposition de ce modèle est la facturation équitable pour les clients dans un environnement de *cloud*, cette proposition ne considère pas les coûts qui peuvent être dus aux pénalités et/ou à l'utilisation des ressources matérielles. Colmant et *al.* [54] proposent une modélisation fine (i.e. au niveau processus) pour estimer la consommation de puissance d'une VM.

Le modèle proposé se focalise sur la puissance consommée par le CPU durant l'exécution des différents processus d'une VM donnée. Cette approche ne prend pas en considération la consommation de puissance associée aux périphériques de stockage. Même si le modèle proposé traite une problématique liée à un environnement virtualisé, les contraintes dues à l'environnement *cloud* ne sont pas prises en compte. Stoess et *al.* étudient dans [176] la consommation énergétique des VM, en surveillant les activités des différents pilotes au niveau de l'hyperviseur. Ce type d'approche nécessite des modifications au niveau de l'hyperviseur, et donc limite sa ré-utilisabilité. De plus, elle ne considère pas les coûts liés au système de stockage. Enfin, il existe d'autres modèles de coût de la consommation énergétique des VM [100] [29] [111], mais aucun ne considère l'impact des systèmes de stockage et de l'environnement *cloud*.

### 3.3.2 *Modèles de coût des périphériques de stockage*

Plusieurs travaux ont proposé des modèles de coût pour les périphériques de stockage. Le modèle proposé par Kim et *al.* dans [108] considère la diversité des périphériques de stockage, et tente d'optimiser le système de stockage en fonction de sa consommation énergétique et de son usure. Ce modèle d'optimisation produit un placement de données pour une charge d'E/S, mais néglige le coût associé à la migration entre les périphériques de stockage. D'autres approches proposent des modèles de coûts considérant la consommation énergétique et le coût de l'usure des périphériques de stockage [122] [198] [204]. Toutefois, ces propositions ne peuvent être appliquées dans un environnement virtualisé où les performances et la consommation énergétique dépendent de plusieurs facteurs (e.g. type de virtualisation, l'hyperviseur, la configuration des VM, etc).

### 3.3.3 *Modèles de coût dans le contexte du cloud computing*

Le contexte du *cloud* implique différentes contraintes qui rendent la modélisation des coûts plus complexe. Par exemple, la migration des VM est souvent utilisée dans les centres de données, pour minimiser la consommation d'énergie ou établir un équilibrage des charges. Les travaux présentés dans [123] [115] [177] s'appuient sur l'utilisation CPU et mémoire afin d'estimer la consommation énergétique d'une VM.

### 3.3.4 *Discussion sur les modèles de coût*

Après avoir parcouru les différentes catégories de modèles de coûts de l'état de l'art, nous avons constaté que leurs principaux inconvénients peuvent être résumés comme suit :

- les modèles de coût ciblant la consommation énergétique des VM négligent l'énergie consommée par le système de stockage lors de l'exécution des charges d'E/S ;
- les modèles de coût des périphériques de stockage se basent sur une modélisation haut niveau des charges d'E/S et des périphériques de stockage. Ils ignorent l'impact de l'environnement *cloud* (i.e. virtualisation, SLA, migration), qui peut engendrer des coûts supplémentaires.

Il est donc nécessaire de proposer un modèle de coût prenant en compte les différents paramètres cités ci-dessus.

Une fois ce modèle établi, l'étape d'optimisation (i.e. étape *Plan* de MAPE-K) peut être entamée. La prochaine partie parcourt les différentes approches présentées dans la littérature traitant des problèmes de placement de données et de VM.

## 3.4 PLAN : OPTIMISATION DE PLACEMENT DU STOCKAGE DES VM

L'objectif est de minimiser le coût de stockage et d'exécution des charges d'E/S des VM, et ce dans le contexte d'un *cloud* IaaS utilisant un système de stockage hybride. Cette partie présente les travaux de l'état de l'art traitant l'optimisation de placement de VM, en deux catégories. La première catégorie réunit les approches visant à optimiser le placement de données dans les systèmes de stockage, indépendamment du contexte du *cloud* IaaS et des VM. La seconde catégorie regroupe les travaux traitant de l'optimisation de placement de VM dans les centres de données, dans le contexte du *cloud*. Une discussion sur les différentes approches est présentée à la fin de cette partie.

### 3.4.1 *Optimisation du stockage des données*

Les travaux [108] [199] appliquent un algorithme exact d'optimisation sur un système de stockage hybride. Ces approches utilisent la programmation linéaire en nombre entier dans l'objectif de minimiser les coûts d'installation des périphériques de stockage ainsi que d'autres coûts récurrents (i.e. consommation énergé-

tique, refroidissement, gestion, et maintenance). Ces approches sont contraintes par les performances du système de stockage et l'usure des SSD.

Cheng *et al.* [49] propose la solution CAST (*Cloud Analytics Storage Tiering*) pour réduire les coûts monétaires et améliorer les performances du stockage afin d'accélérer des traitements *Hadoop* [193] dans le *cloud*. Le système de stockage considéré est composé de quatre classes de stockage : 1) stockage SSD éphémère (i.e. en DAS sur les PM), 2) stockage SSD persistant (i.e. en NAS), 3) stockage HDD persistant (i.e. en NAS), et 4) stockage en objets (i.e. *Google Cloud Object Storage*). Cette méthode utilise une métaheuristique basée sur le recuit simulée. La proposition est comparée à un algorithme glouton.

Dans le même domaine, nous pouvons citer également [191]. Ce travail propose une approche d'optimisation de placement de données dans un environnement HPC (*High Performance Computing*). Elle combine chaînes de Markov et programmation linaire, dans le but de maximiser les performances du système de stockage.

Certains travaux traitent le problème d'optimisation de stockage hybride appliqués à d'autres types d'application. À titre d'exemple, [202] et [41] proposent des heuristiques pour le placement d'objets de bases de données dans le contexte du *cloud*.

[202] vise à minimiser le coût total d'exploitation qui inclut le coût amorti des ressources matérielles et le coût énergétique. Elle prend comme contrainte les performances demandées par les clients exprimées par le SLA.

Le travail présenté dans [41] a pour objectif l'amélioration des performances des bases de données. Cette proposition modélise le problème d'optimisation sous la forme d'un problème de sac à dos. La résolution est assurée par la programmation dynamique et par un algorithme glouton.

### 3.4.2 *Optimisation du placement des VM*

Dans cette partie, nous nous focalisons sur le placement de VM dans les centres de données pour différents objectifs (consolidation des serveurs, équilibrage de charge, respect du SLA, etc). Vu le nombre important de travaux traitant ce type de problème, nous avons choisi de les regrouper selon l'objectif visé et la méthode d'optimisation utilisée. Les fonctions objectifs sont généralement : 1) la minimisation des coûts (énergie, prix du service, pénalités, etc), et/ou 2) la maximisation des performances pouvant comporter l'équilibrage de charge. Nous catégorisons les méthodes d'optimisation en trois classes : 1) les méthodes exactes (e.g. programmation linéaire/non-linéaire, en nombre entier/binaire), 2) les heuristiques (e.g. *first-fit decreasing*, *best-fit decreasing*), et 3) les métaheuristiques (e.g. algorithmes génétiques, colonie de fourmis, recuit simulé). Le tableau 3 présente les travaux de l'état de l'art qui nous semblent les plus pertinents.

| Objectif | Méthodes exactes | Heuristiques | Métaheuristiques |
|---|---|---|---|
| **Minimisation des coûts** | [22] [181] [81] [185][114] [80] | [119] [121] [80] [26] [24] [197] | [121] [114] [66] [93] [72] [66] [157] |
| **Maximisation des performances** | [81] [120] [126] [140] [22] | [140][135] [126] [86] | [72] [197] [65] |

TABLE 3 : travaux sur le placement de VM. Certaines références peuvent apparaître plusieurs fois car, soit elles ont plusieurs objectifs, soit elles utilisent plusieurs approches de résolution.

Nous pouvons constater que la majorité des approches visent la minimisation des coûts, notamment la consommation énergétique. La plupart de ces approches utilisent des heuristiques et des métaheuristiques. Les méthodes exactes sont utilisées dans trois circonstances :

- des problèmes de petite taille (e.g 100 PM, 800 VM dans [120]) ;
- combinées avec des approches approchées ;
- comme une base de comparaison avec les approches approchées.

Que ce soit pour la consommation énergétique ou pour les performances, le placement de VM se base essentiellement sur l'utilisation CPU. Peu d'approches étudient d'autres ressources [92], telles que le stockage [86], la mémoire [140], ou le réseau [135].

### 3.4.3 *Discussion sur les approches d'optimisation*

Nous avons présenté plusieurs approches d'optimisation appliquées au stockage de données et de placement de VM. La première catégorie regroupe les approches de placement de données appliquées aux systèmes de stockage hybrides. Par rapport à notre étude, ces approches ont les inconvénients suivants :

- les approches ne prennent pas en compte la virtualisation, ce qui peut avoir un impact important sur la qualité des résultats ;
- ces travaux visent des types spécifiques d'applicatifs (e.g. HPC, bases de données, etc). Cela suppose la connaissance *a priori* du type d'application, ce qui n'est pas en accord avec le contexte de notre étude ;

- dans le cas où le type d'applicatif n'est pas connu à l'avance, ces approches exigent une étape de caractérisation hors ligne des charges de travail. Or, nous cherchons à proposer une méthode d'optimisation en ligne, pouvant s'adapter aux changements de comportement des E/S des VM.

La deuxième catégorie regroupe les travaux ayant comme objectif l'optimisation du placement de VM. En les confrontant à notre problématique, ces approches souffrent des faiblesses suivantes :

- ces méthodes se basent exclusivement sur l'utilisation CPU pour placer les VM. Pour notre étude, nous nous intéressons plutôt aux opérations E/S et au placement de stockage de données de VM ;
- les propositions prenant en compte le stockage ne considèrent pas toutes les caractéristiques des périphériques de stockage (seulement la capacité ou les performances) ;
- les systèmes de stockage hybrides n'ont pas été pris en compte dans les approches actuelles.

L'avantage de ces méthodes est qu'elles ont été établies en considérant l'environnement *cloud* (virtualisation, SLA, migration, etc). Afin de construire une approche de placement de VM, nous avons combiné les avantages de chacune des catégories présentées dans cette partie.

## 3.5 PLAN : IMPLANTATION ET ÉVALUATION DES APPROCHES D'OPTIMISATION DE PLACEMENT DE VM

Dans la littérature, la modélisation, l'optimisation, et l'évaluation des approches de placement de VM s'effectuent sur des environnements réels, des environnements simulés, ou une combinaison des deux [127]. L'évaluation sur des environnements réels souffre du problème de passage à l'échelle. Ce type d'environnement se limite souvent à quelques PM et VM [127]. De plus, des environnements *cloud* réels ne sont pas toujours à la disposition des chercheurs pour des problèmes de coûts. C'est pour ces raisons que les chercheurs se dirigent vers l'utilisation des environnements de *cloud* simulés [127]. Cette partie présente une liste de simulateurs utilisés dans l'état de l'art.

### 3.5.1 *SimGrid*

*SimGrid* est un outil de simulation qui permet d'étudier différents types de systèmes informatiques (e.g. *Grid*, *Cloud*, HPC, *P2P*) [45]. Ce simulateur peut être

utilisé pour évaluer des heuristiques, des prototypes ou des applications de calcul parallèle utilisant MPI [7]. Les premières versions de *SimGrid* visaient la simulation et l'évaluation des problèmes d'ordonnancement des applications distribuées [44]. Les aspects liés au *cloud* IaaS, notamment le support des VM et leur migration ont été ajoutés plus tard [89] [88]. Le support de stockage a également été inclus via une extension de *SimGrid* [116].

*SimGrid* souffrent tout de même de quelques lacunes. Le problème majeur de *SimGrid* est le passage à l'échelle. En effet, la taille de l'environnement simulé est limité par la mémoire du système utilisé. Pour une machine avec 8GB de RAM, cette limite est atteinte avec une simulation comportant 4000 PM [59]. Le deuxième problème réside dans le fait que *SimGrid* ne dispose pas des fonctionnalités permettant l'implantation de méthodes d'optimisation [117].

### 3.5.2 *GridSim*

*GridSim* est un outil permettant la modélisation et la simulation des systèmes de calcul parallèles et distribués à grande échelle [39]. Ce simulateur a pour vocation l'implantation et l'évaluation des politiques d'allocation des ressources de calcul aux applications parallèles et distribuées. *GridSim* introduit la notion de coût d'utilisation des ressources matérielles en tant que fonction objectif pour les approches d'ordonnancement ou d'allocation de ces ressources.

Initialement, *GridSim* n'était pas destiné à la simulation des environnements de *cloud* IaaS. Néanmoins, *GridSim* dispose d'une plateforme qui permet l'extension de ce simulateur pour la prise en compte des environnements *cloud*.

### 3.5.3 *CloudSim*

*CloudSim* est un simulateur de *cloud* IaaS fréquemment utilisé pour l'implantation et l'expérimentation de méthodes d'optimisation [40]. Il est le plus utilisé dans l'état de l'art [127]. 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 avons fait le choix d'expérimenter nos contributions sur *CloudSim* pour les raisons suivantes :

- nous positionner par rapport aux travaux de l'état de l'art qui ont utilisés ce simulateur [26] ;
- la possibilité d'extension de *CloudSim* pour la prise en compte des E/S et des systèmes de stockage ;

- la disponibilité des modèles de placement de VM qui sont simple à étendre ou à réimplanter.

Nous détaillons l'architecture et le placement de VM dans *CloudSim* dans les parties 6.2.1 et 7.2.

## 3.6 CONCLUSION

Ce chapitre présente un état de l'art concernant l'ensemble de nos contributions. Ces travaux sont organisés suivant les étapes du modèle MAPE-K.

Pour l'étape *Monitor*, nous avons d'abord exposé les outils de *monitoring* existant et pouvant être utilisés pour surveiller les E/S des VM. Les outils abordés dans cette partie ont été regroupés en deux catégories : outils de *monitoring* 1) génériques, et 2) spécifiques aux E/S.

Les outils de la première catégorie offrent une flexibilité dans la mise en œuvre, mais nécessitent une expertise pour leur utilisation. Il nous faut une connaissance préalable des fonctions à surveiller avant d'utiliser ces outils. Cela représente un travail fastidieux et coûteux en terme de temps d'exploration.

Chaque outil de la deuxième catégorie opère à un niveau spécifique de la pile des E/S Linux, mais nous avons besoin de tracer les E/S en partant d'une VM jusqu'au périphérique de stockage.

Nous avons également présenté les travaux proposant des approches de caractérisation des charges d'E/S. Ces travaux ont été classés en deux catégories : modèles de caractérisation des charges d'E/S 1) généralistes, et 2) spécifiques aux environnements virtualisés.

Pour chaque périphérique de stockage, les approches de la première catégorie appliquent une modélisation hors ligne et à long terme des charges d'E/S. Pour pallier le changements fréquents de comportement des VM, nous avons besoin de caractériser les charges d'E/S par VM, en ligne, et à court terme.

La majorité des travaux de la deuxième catégorie vise un niveau particulier (i.e. souvent l'hyperviseur) pour capturer les E/S et ainsi caractériser les charges d'E/S. Le fait de se restreindre à un seul niveau peut conduire à une mauvaise caractérisation et donc à des résultats erronés.

Pour l'étape *Analyze*, nous avons présenté les travaux traitant du problème de la modélisation des coûts dans les centres de données. Ces modèles ont été divisé en trois classes : modèles de coût 1) de l'énergie et des VM, 2) des périphériques de stockage, et 3) dans le contexte du *cloud computing*.

Nous avons constaté que les modèles de la première et de la troisième catégorie ne prennent pas en compte l'exécution des charges d'E/S des VM. Les

approches de la deuxième catégorie négligent l'impact de l'environnement *cloud*.

Pour l'étape *Plan*, nous avons regroupé les approches d'optimisation destinées aux systèmes de stockage et de placement de VM. Cette partie a été divisée en deux classes : les approches d'optimisation 1) du stockage de données, et 2) du placement des VM.

Les méthodes de la première classe ne considèrent pas la virtualisation et visent des types spécifiques d'applicatifs (e.g. HPC, bases de données, etc). Cela suppose la connaissance a priori du type d'application, ce qui n'est pas en accord avec le contexte de notre étude.

Les approches de la deuxième classe s'appuient sur l'utilisation CPU. Il existent des travaux prenant en compte le stockage, mais qui ne considèrent pas toutes les caractéristiques des périphériques et des systèmes de stockage hybrides.

Finalement, nous avons proposé une liste des simulateurs de *cloud* les plus utilisées, à savoir *SimGrid*, *GridSim*, et *CloudSim*.

Après avoir présenté le contexte et les travaux de la littérature liés à notre étude, nous présentons dans la partie suivante l'ensemble de nos contributions.

# Deuxième partie

# CONTRIBUTIONS

# Chapitre 4

# ÉTAPE *MONITORING* : SUPERVISION ET CARACTÉRISATION DES ACTIVITÉS D'ENTRÉES/SORTIES DES VM

Le placement de VM doit se faire en fonction d'un ensemble de contraintes, dans le but d'optimiser une fonction de coût. Pour cette dernière, les VM ont plusieurs caractéristiques qui dépendent des ressources de l'infrastructure matérielle. En effet, chaque VM (ou ensemble de VM) a un profil d'utilisation des ressources qui lui sont allouées. Une fois déterminé, il est possible d'estimer son coût d'utilisation qui est l'élément clé pour l'optimisation.

Cette partie présente notre contribution sur l'étape *Monitor* du modèle MAPE-K. Cette contribution comprend un outil de *monitoring* et un outil de caractérisation des charges d'E/S des VM. L'outil de *monitoring* permet une surveillance périodique des opérations d'E/S exécutées par les VM pendant une durée déterminée. L'outil de caractérisation permet d'établir les profils d'E/S pour chaque VM.

La première partie de ce chapitre présente la problématique qui nous a motivé à proposer nos outils de monitoring. En second temps, nous donnons un aperçu de nos contributions sur l'étape *monitor*, ensuite, nous détaillons chacune d'elles. Enfin, nous présentons l'évaluation et les résultats de nos contributions, puis nous concluons ce chapitre.

**Sommaire**

## 4.1 PROBLÉMATIQUE

Dans un environnement *cloud* de type IaaS, le service est proposé sous forme de VM conçues selon différents objectifs. Afin d'offrir des services adaptés aux besoins des clients, le fournisseur du *cloud* doit avoir une idée sur l'interaction des VM avec le système de stockage. Néanmoins, l'administrateur du *cloud* ne connaît pas en détail l'applicatif exécuté par une VM donnée. Il est donc nécessaire de disposer d'un moyen permettant d'établir le profil E/S exact d'une VM.

La deuxième problématique se présente au niveau des différentes couches logicielles qui séparent une VM du support de stockage physique. Chaque couche peut avoir un impact sur le comportement d'une VM vis-à-vis du système de stockage. Il existe des outils qui permettent de tracer des opérations d'E/S ou des appels systèmes (voir partie 3.1). Mais chacun de ces outils opère sur un niveau précis de la pile logicielle d'E/S. Un seul niveau de trace ne permet pas d'avoir le profil exact d'une VM donnée.

Tout au long de ce chapitre, nous présentons l'implantation d'un outil de trace des E/S des VM multi-niveaux. En second temps, nous détaillons la caractérisation des charges d'E/S des VM en utilisant les résultats de l'outil de trace. La méthode et les résultats d'évaluation des outils de trace et de caractérisation des charges E/S sont présentés dans la dernière partie de ce chapitre.

## 4.2 OUTILS DE L'ÉTAPE DE *monitoring*

Dans notre contexte, l'étape de *monitoring* est constituée de deux phases : 1) la surveillance des E/S, et 2) la caractérisation des charges d'E/S des VM. Nous introduisons brièvement dans cette partie chacune de ces étapes.

### 4.2.1 *Versions et aperçu de l'outil de monitoring des E/S*

L'outil de trace (que nous appelons "traceur" tout au long du manuscrit) a été réalisé en deux versions.

La première version combine des outils de trace (i.e. *blktrace* et *strace*) et des bibliothèques (i.e. *libvirt* et *libext2fs*), afin de fournir les traces des opérations d'E/S des VM sur différents niveaux du système Linux. Cette version a été évaluée et a montré des résultats encourageants. Néanmoins, la combinaison des différents outils exhibe certaines faiblesses que nous allons souligner au fur et à mesure de la présentation de chaque niveau de traces.

Afin de surmonter les lacunes présentes dans la première version, nous avons développé une seconde version sous forme d'un module noyau Linux inspirée du traceur présenté dans [146].

Nous pouvons donner une vue globale des deux versions de l'outil et des niveaux de trace comme suit :

- **Niveau 1 : hyperviseur**
  - Version 1 : utilise la bibliothèque *libvirt* [162]
  - Version 2 : niveau pas pris en compte
- **Niveau 2 : VFS**
  - Version 1 : utilise l'outil *strace* [8]
  - Version 2 : utilise *jprobe* [102]
- **Niveau 3 : FS** (ce niveau est utilisé pour faire le lien entre les niveaux précédents et la couche bloc E/S)
  - Version 1 : liaison faite en utilisant *libext2fs*
  - Version 2 : liaison faite en utilisant les structures internes (i.e. *inode*)
- **Niveau 4 : bloc E/S**
  - Version 1 : utilise l'outil *blktrace*
  - Version 2 : utilise *jprobe* [102]

Dans la partie 4.3, nous détaillons l'implantation des deux versions et le rôle de chaque niveau de trace.

### 4.2.2 *Exploitation des traces pour la caractérisation des charges d'E/S*

L'objectif principal de l'étape de *monitoring* est de dresser un profil d'E/S des VM. Le profil obtenu doit décrire d'une manière précise la façon dont la VM interagit avec le système de stockage physique sous-jacent.

La deuxième phase du *Monitoring* consiste à exploiter les traces obtenues à l'aide du traceur d'E/S, afin de définir les profils d'E/S des VM. Le profil d'une VM est un ensemble de paramètres définissant la façon dont la VM accède au systèmes de stockage pour exécuter ses opérations d'E/S. Nous nous somme inspiré des travaux de la littérature pour concevoir un modèle de caractérisation des charges d'E/S des VM.

La partie 4.4 détaille l'identification et l'extraction des paramètres utilisés pour la caractérisation des charges d'E/S.

## 4.3 SURVEILLANCE MULTI-NIVEAUX DES E/S DES VM

Le principe de *Monitoring* dans le modèle MAPE-K (voir partie 2.7) consiste d'abord à définir un ensemble de métriques susceptibles d'avoir un impact sur le fonctionnement du système. Une fois les métriques définies, il est nécessaire de poser

des capteurs afin de relever les valeurs de ces métriques lors du fonctionnement du système.

Les requêtes d'E/S exécutées par une VM traversent de multiples couches logicielles avant d'atteindre le périphérique de stockage physique. Pour mettre en place une approche de *monitoring*, il est d'abord indispensable d'identifier les différentes couches qui participent à l'exécution d'une requête d'E/S lancée par une VM. Pour y parvenir, nous avons suivi une approche d'exploration descendante, en commençant par la VM jusqu'au périphérique physique de stockage.

La figure 17 montre les différentes couches traversées par les requêtes d'E/S des VM pour atteindre le système de stockage.

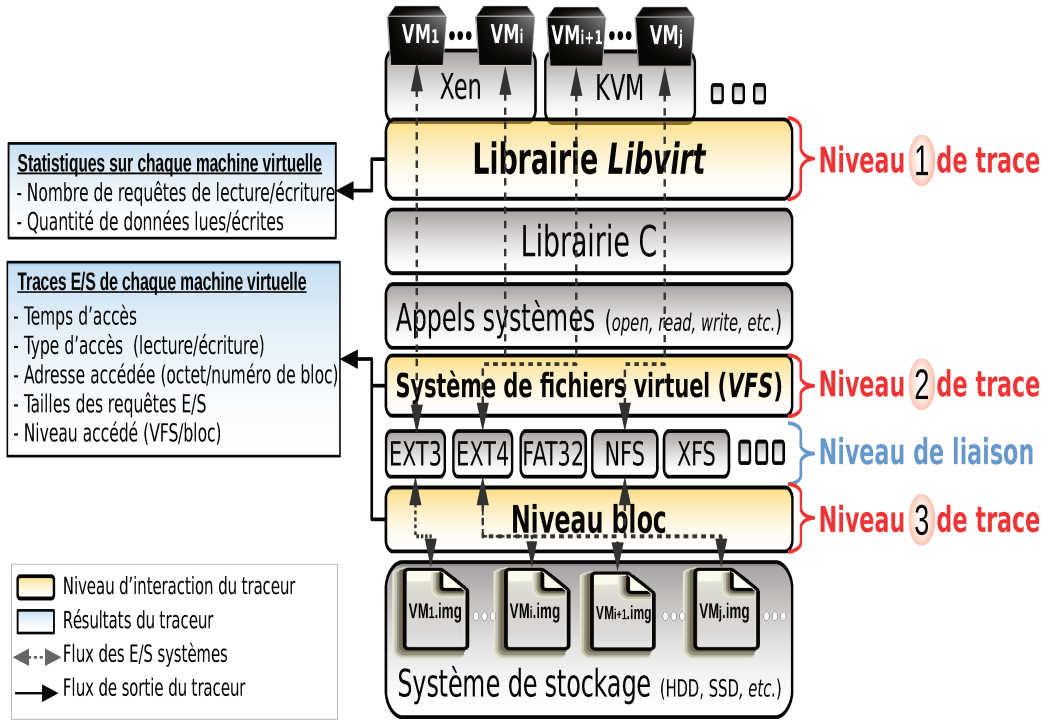


FIGURE 17 : architecture du traceur d'E/S multi-niveaux

La première couche accessible par le système hôte est l'hyperviseur de VM. Chaque VM gérée par l'hyperviseur est vue comme un simple processus au niveau du système d'exploitation hôte. Les VM accèdent à leurs stockages respectifs de la même façon qu'un processus accède à un fichier. En effet, le stockage d'une VM (appelé aussi disque virtuel) est habituellement un ou plusieurs fichiers. En pratique, le stockage d'une VM est enregistré dans un fichier qui représente son image disque. Le système utilisé pour stocker les disques virtuels peut être local à chaque PM, partagé par toutes les machines, ou distribué entre l'ensemble des hôtes (voir partie 2.2).

Dresser le profil d'E/S d'une VM nécessite de suivre les requêtes d'E/S depuis la VM jusqu'au périphérique de stockage physique. Pour exécuter des opérations d'E/S disque, les requêtes correspondantes, issues de la VM, suivent le même

chemin que celles issues d'un processus. C'est pourquoi nous proposons un traceur des opérations d'E/S des VM qui opère sur différents niveaux du système hôte.

L'ensemble des VM est géré par un hyperviseur. Pour palier l'hétérogénéité des hyperviseurs, il existe la bibliothèque *libvirt* qui prend en charge différentes opérations de gestion des VM (voir partie 2.6). Au vu du système d'exploitation hôte, chaque VM est considérée comme un processus indépendant. Ce processus exécute des appels systèmes (i.e. par l'intermédiaire de la bibliothèque *libC* par exemple) sur des fichiers afin d'accéder à son espace de stockage. Les appels système d'E/S passent d'abord par le système de fichiers virtuel VFS. La couche VFS représente le deuxième niveau d'observation du traceur. Les opérations d'E/S utilisent les fonctions des systèmes de fichiers physiques pour accéder à la couche bloc. La couche bloc est le dernier niveau d'accès du traceur, avant d'atteindre les pilotes matériels correspondant à chaque périphérique de stockage.

Les paragraphes ci-dessous détaillent les niveaux exploités par le traceur, ainsi que la nature des traces correspondantes.

### 4.3.1 *Niveau 1 : hyperviseur*

Il existe différents traceurs génériques permettant de générer des traces sur des fonctions implantées par l'hyperviseur utilisé pour la virtualisation (voir partie 3.1). Le problème rencontré à ce niveau est l'hétérogénéité des hyperviseurs. En effet, il n'existe pas un outil unique permettant de tracer les fonctions de différents hyperviseurs d'une manière identique.

Pour cette raison, nous avons fait le choix d'utiliser la bibliothèque *libvirt*, qui permet de mutualiser la gestion des hyperviseurs fréquemment utilisés (voir partie 2.6). La figure 18 montre le flux de données de la bibliothèque *libvirt* et l'intégration de son API dans le processus de *monitoring* des opérations d'E/S.

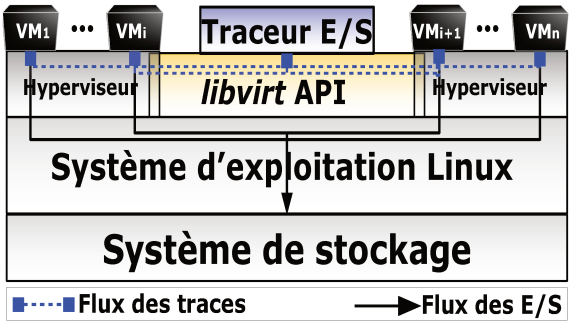


FIGURE 18 : intégration de *libvirt* dans le processus de monitoring

La trace collectée à ce niveau est représentée par un ensemble de statistiques sur le type (lecture/écriture) et le nombre d'opérations d'E/S, ainsi que la quantité de données traitées (lues/écrites).

Tracer toutes les fonctions des E/S effectuées par l'ensemble des VM, sans modifier le code source de l'hyperviseur utilisé, représente un travaille complexe, principalement due à la multitude des hyperviseurs utilisés dans les environnement de virtualisation. *libvirt* nous permet d'interroger l'hyperviseur sur l'utilisation de différentes ressources matérielles dont le stockage. L'interrogation de l'hyperviseur se fait périodiquement, avec une fréquence d'échantillonnage d'une mesure par seconde. Le pseudo-code détaillant le fonctionnement de la collecte de trace au niveau d'hyperviseur est présenté en annexe (voir algorithme 5 partie B.1.1). Un exemple de résultat d'une trace est y également donné (voir tableau 26 partie B.1.1).

### 4.3.2 *Niveau 2 : système de fichiers virtuel (VFS)*

Comme le montre la figure 17, le système de fichiers virtuel VFS représente le deuxième niveau de trace. La couche VFS est une couche d'abstraction permettant de mutualiser l'accès aux différents systèmes de fichiers physiques (e.g. BTRFS, EXT4, NFS). Cette couche sépare l'interface des appels `POSIX` du détail d'implantation des fonctions des systèmes de fichiers physiques sous-jacents. Les hyperviseurs exécutant les VM accèdent aux différents fichiers à l'aide des appels système (e.g. `open`, `close`, `read`, `write`, `mmap`, etc). Du point de vue du système hôte, une VM est un processus fils du processus parent qui est l'hyperviseur.

#### 4.3.2.1 *Utilisation de l'outil strace*

Il est possible de tracer les appels systèmes exécutés par un processus défini dans un système Linux (voir partie 3.1).

Dans le premier prototype du traceur d'E/S, nous avons utilisé l'outil *strace* (voir partie 3.1.2). L'outil *strace* permet de tracer les appels systèmes exécutés par un processus, en prenant le PID du processus en question comme paramètre. Afin de collecter les traces des VM au niveau VFS, nous procédons en suivant deux étapes essentielles.

Dans la première étape, nous traçons chaque processus représentant une VM. La deuxième étape a pour but d'isoler les traces par rapport aux disque virtuels des VM.

**a) Étape 1 (trace de processus et filtrage des fonctions) :** comme chaque VM est vue comme un processus au niveau du système hôte, cette VM doit posséder un PID unique. Nous avons utilisé l'API de la bibliothèque *libvirt* pour intercepter le PID d'une VM en donnant son nom ou son *uuid* (*Universal Unique IDentifier*). Le *pid* de chaque VM est passé comme argument à *strace*, afin de tracer ses appels systèmes. Les VM sont tracées conjointement par le lancement d'un *thread* système par VM.

Dans le but de minimiser le surcoût induit par *strace*, les appels systèmes sont filtrés à la volée pour ne garder que ceux effectuant des E/S. Nous avons donc tracé les appels systèmes des E/S manipulant les fichiers et générant des accès disque : `create()`, `open()`, `read()`, `write()`, `lseek()`, et `close()`.

L'estampillage de temps des appels systèmes est très important pour avoir un ordre des opérations d'E/S sur les différents niveaux de trace (hyperviseur, VFS, et bloc d'E/S). *strace* a une précision de l'ordre de microseconde pour l'estampillage de temps.

Cette étape produit $n$ fichiers de trace, où $n$ égale au nombre de VM en cours d'exécution. Chaque fichier contient l'ensemble des appels systèmes d'E/S exécutés par la VM. Le format du fichier de trace en sortie est présenté en annexe (voir partie C.2.2).

Il faut noter qu'une VM ne manipule pas seulement les fichiers des disques virtuels, mais aussi d'autres fichiers (e.g fichiers de configuration, fichiers binaires, bibliothèques, etc). Il est donc nécessaire d'appliquer un autre filtre, afin de garder exclusivement les accès aux fichiers qui représentent les disques virtuels d'une VM.

**b) Étape 2 (isolation des traces par disque virtuel) :** le deuxième étape constitue un filtrage pour garder uniquement les opérations d'E/S sur les disques virtuels.

Pour ce faire, l'API de la bibliothèque *libvirt* est utilisée pour identifier les fichiers des disques virtuels. En effet, en utilisant *libvirt* nous pouvons parcourir la configuration d'une VM et ainsi explorer l'ensemble des disques virtuels.

Pour un disque virtuel donné, *libvirt* permet d'accéder et de modifier différentes caractéristiques tel que son bus d'E/S (e.g. IDE, SATA, *virtio*, etc), le mode du cache utilisé (*writethrough, writeback*, etc), et les chemins exacts des fichiers physiques représentant les disques virtuels. Ces derniers nous intéressent particulièrement car ils sont utilisés dans la deuxième partie de filtrage. Ce filtrage est effectué en parcourant le fichier de trace afin de ne conserver que les appels sur les fichiers des disques virtuels.

Chaque VM peut accéder à différents fichiers pendant son exécution. Pour différencier ces fichiers, nous utilisons les descripteurs de fichiers.

La deuxième étape de filtrage est effectuée en utilisant les descripteurs des fichiers des disques virtuels. Sous le système Linux, nous pouvons retrouver l'ensemble des descripteurs des fichiers ouverts par un processus en parcourant le répertoire `/proc/PID/fd/` où `PID` est l'identificateur du processus en question. En utilisant les chemins exacts des fichiers obtenus à partir de l'API et les descripteurs des fichiers du processus de la VM, nous pouvons filtrer les opérations d'E/S sur les disques virtuels d'une VM.

#### 4.3.2.2 *Utilisation de jprobe*

La deuxième version du traceur d'E/S est développée sous la forme d'un module noyau Linux.

Cette tâche a été réalisée en deux étapes. La première étape consiste à inspecter les fonctions du noyau Linux en exécutant des opérations d'E/S, afin d'identifier les fonctions à sonder. Les fonctions ciblées sont les fonctions génériques interagissant avec tous les systèmes de fichiers physiques (e.g. EXT2, EXT4, XFS, etc).

**a) Étape 1 (inspection de l'exécution des E/S) :** dans cette étape, nous cherchons à trouver l'ensemble de fonctions d'E/S susceptibles d'être tracées. Les fonctions recherchées doivent :

- opérer au niveau du système de fichiers virtuel (VFS) ;
- être génériques (i.e. liées directement à tous les systèmes de fichiers sous-jacents) ;
- être "en sortie" de la couche VFS, avant l'accès aux différents systèmes de fichiers, afin d'éviter les différents caches de la couche VFS (i.e. cache des répertoires et cache des *inode*)

Il existe plusieurs outils permettant l'inspection des activités du noyau Linux en cours d'exécution (voir partie 3.1). L'un des outils les plus performants est GDB (*Gnu DeBugger*). Le support de GDB proposé dans *qemu* permet d'exploiter les fonctionnalités de GDB en rendant possible l'inspection de l'exécution d'un noyau exécuté par une VM *qemu*. Le processus d'inspection des fonctions en utilisant GDB est illustré en annexe (voir partie C.2.2.2).

Les fonctions choisies sont celles appelées par les systèmes de fichiers physiques. Pour les identifier, nous avons exécuté des opérations d'E/S dans une VM sur une partition formatée en EXT4. Les fonctions à caractère générique sont identifiées par le fait qu'elles font appel aux (ou sont appelées par les) fonctions EXT4. Si nous prenons l'exemple d'une opération de lecture de fichier, le système de fichiers EXT4 utilise la fonction `generic_file_read_iter`. À titre d'exemple, cette fonction est candidate pour être sondée à l'aide des sondes *jprobe* (voir 3.1.1).

**b) Étape 2 (placement des sondes) :** après l'identification de l'ensemble de fonctions d'E/S susceptibles d'être sondées, vient l'étape qui consiste à poser les sondes *jprobes*. Les fonctions sondées ainsi qu'un exemple des résultats obtenus sont présentés en annexe (voir partie C.2.2.2).

### 4.3.3 *Niveau 3 : système de fichiers physique (FS)*

Contrairement aux deux niveaux de traces détaillés précédemment, les systèmes de fichiers physiques ne représentent pas un niveau de trace à proprement parlé. La couche des systèmes de fichiers physiques permet essentiellement de faire le lien entre les couche hautes (i.e. hyperviseur et VFS), et les couches basses (i.e. couche bloc d'E/S).

En effet, la cartographie des fichiers représentant les disques virtuels peut être obtenue à partir des systèmes de fichiers physiques. Il faut noter qu'un fichier est représenté sous Linux par son *inode*. La liaison faite au niveau du système de fichiers est une correspondance entre les fichiers des disques virtuels, et les blocs correspondants au niveau de la couche bloc d'E/S.

Parmi la multitude des systèmes de fichiers, nous nous sommes basés sur les systèmes de fichiers EXT3 et EXT4 pour le développement du traceur, compte tenu de leur popularité [130].

Nous avons utilisé la librairie *libext2fs* qui permet d'explorer et de manipuler les systèmes de fichiers Linux EXT2/3/4 depuis l'espace utilisateur. Cela nous évite de toucher au code du noyau.

*libext2fs* dispose des fonctions suivantes :

- opérer au niveau des systèmes de fichiers physiques ;
- permettre de reproduire la cartographie des fichiers stockés sur le disque ;
- être facilement intégrable dans le processus de trace sans en perturber le fonctionnement.

*libext2fs* est une bibliothèque permettant d'accéder aux systèmes de fichiers de type EXT. Cette bibliothèque permet aux programmes s'exécutant dans l'espace utilisateur d'accéder et de manipuler le système de fichiers *ext2* [184]. En terme d'intégration, *libext2fs* offre une API en langage C qui peut être parfaitement intégrée dans le processus de trace. Les détails d'utilisation de la librairie *libext2fs* sont présentés en annexe (voir partie B.1.2).

### 4.3.4 *Niveau 4 : couche bloc d'E/S*

À l'image du niveau de trace VFS, la trace au niveau de la couche bloc a été réalisée en deux versions. La première version présentée dans [152] utilise l'outil de trace d'E/S bloc *blktrace* (voir partie 3.1). La deuxième version est implantée sous forme d'un module noyau Linux en utilisant *jprobe*.

#### 4.3.4.1 *Utilisation de blktrace*

*blktrace* permet de suivre une requête d'E/S durant toute sa traversé de la couche bloc. Autrement dit, *blktrace* montre toutes les étapes par lesquelles les requêtes passent depuis leur soumission jusqu'à leur traitement par le périphérique physique de stockage.

L'utilisation de *blktrace* est structurée en deux étapes. La première étape consiste à utiliser les outils *blktrace* et *blkparse* pour tracer les accès d'E/S sur un disque physique ou une partition. La deuxième étape consiste à isoler les traces par VM et disque virtuel.

**a) Étape 1 (trace d'un disque ou d'une partition) :** dans une PM possédant un stockage local, les fichiers représentant les disques virtuels des VM sont stockés sur les différents disques attachés à cette PM. Lors de l'exécution d'une charge de travail E/S, les VM accèdent aux disques virtuels, et par conséquent accèdent aux partitions des disques sur lesquelles les disques virtuels sont stockés. Ces accès aux disques physiques peuvent être tracés en utilisant *blktrace*. Comme nous l'avons montré précédemment (voir partie 3.1), l'outil *blktrace* requière comme paramètre une partition ou un disque. Dans ce contexte, nous supposons que les VM utilisent un disque dédié à stocker l'ensemble des fichiers des disques virtuels.

Contrairement à *strace*, nous ne pouvons pas lancer autant d'instances de *blktrace* que de VM, mais une seule instance sur le système d'exploitation hôte. La sortie produite par *blktrace* ne peut pas être utilisée directement pour être analysée. Nous devons lancer l'exécution de l'outil *blkparse* qui prend la sortie de *blktrace* en entrée et produit une trace lisible et prête à être utilisée pour des analyses et des statistiques.

**b) Étape 2 (isolation des traces par VM/disque virtuel) :** l'outil *blktrace* ne permet pas de tracer les accès d'un processus particulier, mais plutôt l'ensemble des accès sur une partition ou un disque. Les traces obtenues doivent donc être traitées pour :

1. isoler les traces en fonction des VM, avec un fichier de trace par VM ;
2. faire la correspondance entre les numéros de bloc et les disques virtuels pour les traces d'une même VM ;
3. filtrer les opérations d'E/S afin d'en garder uniquement les opérations de lecture et d'écriture ;
4. filtrer les opérations de l'ordonnanceur des requêtes d'E/S afin de garder les requêtes effectivement traitées par le périphérique de stockage.

Un exemple illustratif d'une trace obtenue à l'aide de *blktrace* est présenté en annexe (voir partie C.2.3).

**Bilan d'utilisation de *blktrace* :** l'utilisation de l'outil *blktrace* nous a permis de non seulement tracer les accès aux périphériques de stockage physique, mais aussi d'avoir des informations précises sur l'évolution de l'état de l'opération d'E/S dans la couche bloc d'E/S. Néanmoins, *blktrace* montre plusieurs lacunes que l'on peut résumer ainsi :

- *blktrace* ne permet pas de tracer simultanément différents périphériques de stockage. Il faut exécuter autant de "*blktrace*" que de périphériques sur une même PM ;
- la dépendance par rapport à *blkparse* rend le processus de trace plus compliqué, car la sortie de *blktrace* ne pas être utilisée sans passer par *blkparse* ;
- l'analyse pour faire la liaison entre les traces au niveau de bloc d'E/S et les couches hautes de trace nécessite un effort supplémentaire. En effet, il faut synchroniser l'estampillage temporelle ainsi que l'ordre des requêtes tracées sur les différentes couches.

Afin de dépasser ces lacunes, la deuxième version du traceur est développée sous la forme d'un module noyau Linux en utilisant *jprobe*.

#### 4.3.4.2 *Utilisation de jprobe*

Le processus d'implantation est le même que celui suivi pour la trace de la couche VFS (voir partie 4.3.2.2). Dans ce contexte, l'objectif de l'étape d'inspection est d'identifier les fonctions noyau d'E/S qui sont :

- utilisées par les pilotes des différents types de périphérique de stockage (afin d'être le plus proche possible du matériel) ;
- en sortie de la couche bloc (afin de capturer uniquement les requêtes au moment de leurs soumissions au périphérique de stockage).

Les fonctions choisies ainsi qu'un exemple des résultats sont décrits en annexe (voir partie C.2.3.2).

Maintenant que nous avons détaillé les composants et l'implantation du traceur d'E/S, la partie suivante détaille le processus de caractérisation des charges d'E/S à partir des fichiers de traces obtenus à l'aide de notre outil de trace.

## 4.4 PROPOSITION D'UNE CARACTÉRISATION DES CHARGES D'E/S DES VM

Cette partie détaille les éléments nécessaires et le processus mené pour caractériser les charges d'E/S de chaque VM à partir des données du traceur.

Il existe dans la littérature des méthodes de caractérisation de charges d'E/S, spécialement dans les environnements virtualisés (voir partie 3.2). Ces méthodes différent les unes des autres par plusieurs critères, telles que la granularité des données traitées (i.e. fichier, bloc, objet), du type de virtualisation et d'hyperviseur, etc. Nous présentons ci-dessous les principaux paramètres utilisés dans la littérature. Le choix des paramètres est argumenté dans la partie 4.4.2.

Avant de détailler la caractérisation des charges d'E/S, il est important de définir quelques notions utilisées tout au long de cette partie.

### 4.4.1 *Définitions*

**Définition 4.1.** *(**charge d'E/S**) une charge d'E/S est un ensemble de requêtes de lecture et d'écriture. Nous pouvons donner une représentation abstraite d'une charge d'E/S notée W comme suit :*

$$\{Req_1, Req_2, \ldots, Req_n\}$$

*où* $Req_1, Req_2, \ldots, Req_n$ *sont des requêtes d'E/S.*

**Définition 4.2.** *(**requête d'E/S**) nous notons une requête d'E/S par* $Req_i$ *:*

$$\{req_{type}, req_{addr}, req_{size}\}$$

*où :*

- $req_{type}$ *est le type de la requête d'E/S (i.e. lecture ou écriture) ;*
- $req_{addr}$ *est l'adresse de début de la requête (i.e numéro d'octet dans le fichier au niveau VFS, ou numéro de bloc au niveau bloc d'E/S) ;*
- $req_{size}$ *est la taille de la requête (quantité de données lues/écrites).*

*La taille d'une requête d'E/S* $req_{size}$ *est représentée comme suit :*

$$req_{size} = x \cdot bloc_{size} \tag{1}$$

*La taille d'une requête d'E/S* $req_{size}$ *est un multiple (i.e.* $x$*) de la taille d'un bloc de données* $bloc_{size}$*. Cette taille est définie par le système d'exploitation et peut être : 1) au minimum égale à la taille d'un secteur de disque (e.g. 512 octets), ou 2) au maximum égale à la taille d'une page mémoire (e.g. 4Ko).*

**Définition 4.3.** *(**accès aléatoire/séquentiel**) : les données stockées sur un périphérique de stockage peuvent être accédées selon trois motifs 1) séquentiels, 2) aléatoires, ou 3) composé d'accès séquentiels et aléatoires. Les figures 19 et 20 présentent une abstraction des accès séquentiels et aléatoires respectivement.*

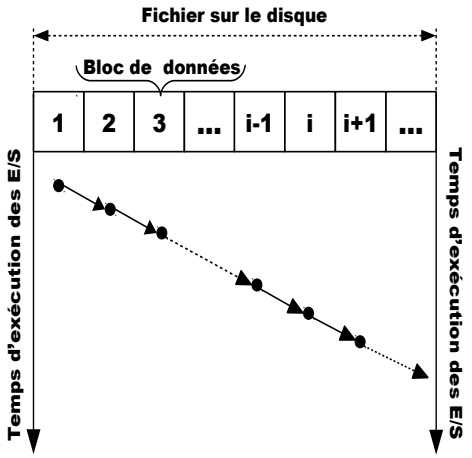


FIGURE 19 : motif d'accès séquentiel

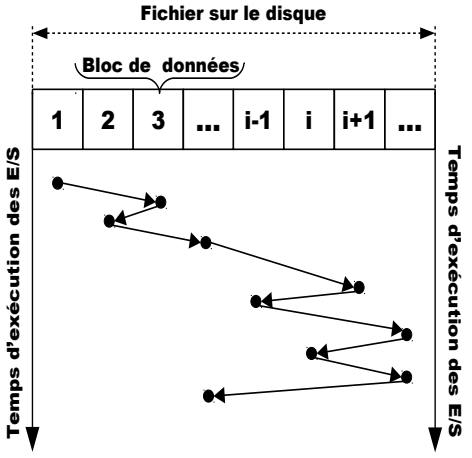


FIGURE 20 : motif d'accès aléatoire

*La figure 19 montre un motif où les requêtes d'E/S successives accèdent à des blocs de données adjacents sur le périphérique de stockage. Dans le cas des accès aléatoires (voir figure 20), les requêtes d'E/S successives n'accèdent pas forcement à des blocs de données contiguës.*

**Définition 4.4.** *(**taux de séquentialité**) : c'est le rapport entre le nombre de requêtes en accès séquentiel et le nombre totale de requêtes.*

**Définition 4.5.** *(**taux de lecture**) : c'est le rapport entre le nombre de requêtes de lecture et le nombre total de requêtes.*

**Définition 4.6.** *(**taille de requête d'E/S**) : c'est la taille des requêtes d'E/S extraite à partir de l'histogramme de fréquences de taille.*

**Définition 4.7.** *(**histogramme de fréquences de taille d'E/S**) : c'est la fréquence d'apparition de chaque taille de requête d'E/S.*

**Définition 4.8.** *(**taux d'arrivée des requêtes d'E/S**) : le nombre de requêtes d'E/S initiées par la VM pendant une seconde.*

**Définition 4.9.** *(**quantité totale de données transférées**) : le volume totale de données transférées entre la VM et le périphérique de stockage physique.*

**Définition 4.10.** *(**période de monitoring**) : notée $T_{mon}$, la période de monitoring désigne la période durant laquelle l'étape de Monitoring est exécutée continuellement afin d'extraire le profil d'accès des E/S produites par les VM.*

### 4.4.2 *Choix des paramètres à caractériser*

Dans cette thèse, le système de stockage est composé de deux classes de stockage : 1) disques durs magnétiques, et 2) disques SSD à base de mémoire flash. Le placement de disques virtuels des VM sur les disques physiques doit tirer profit de la complémentarité entre ces deux classes de stockage. Ainsi, la caractérisation des charges d'E/S des VM doit inclure des paramètres qui mettent en évidence les opportunités offertes par chaque type de périphérique de stockage. Le tableau 4 résume les caractéristiques des périphériques de stockages, ainsi que les paramètres correspondants.

| **Caractéristique** | **Disque dur** | **SSD** | **Paramètre de charge d'E/S** |
|---|---|---|---|
| **Performances** | **Asymétrie selon le motif d'accès** : moins performant en accès aléatoires qu'en accès séquentiels. Plus performant pour les **requêtes de grande taille** | **Asymétrie selon le type d'E/S** : plus performant en lecture qu'en écriture (moins performant en écriture aléatoire aussi) | Indiquent la **séquentialité** des E/S, le rapport **lecture/écriture**, et la **taille des requêtes** |
| **Consommation** | Consomme plus en accès **aléatoires** à cause des latences et des parties mécaniques | Consomme plus en **écriture** à cause des performances, et des opérations internes (i.e. ramasse miettes, nivellement de l'usure) | Indiquent la **séquentialité** des E/S, le rapport **lecture/écriture**, et la **taille des requêtes** |
| **Durée de vie** | Usure en fonction du temps d'utilisation et/ou de la quantité de données lues et écrites | Usure en fonction de la quantité de données écrites | Indiquent la **taille des opérations d'E/S** et/ou la **quantité totale de données** transférées |

TABLE 4 : choix des paramètres en fonction du périphérique de stockage

Le tableau 4 met en évidence les paramètres caractérisant les charges d'E/S des VM. Ces paramètres sont les suivants :

- taux de lecture/écriture ;
- taux de séquentialité/aléatoire ;
- taille de requête d'E/S ;
- taux d'arrivée de requêtes d'E/S ;
- quantité totale de données transférées.

### 4.4.3 *Taux de lecture/écriture*

Une charge d'E/S est principalement composée d'un ensemble de requêtes de lecture et d'écriture. Le taux de lecture/écriture est l'un des paramètres utilisé dans les différentes approches de caractérisation des charges d'E/S (voir partie 3.2). Le calcul de ce paramètre nécessite le parcours du fichier de trace en isolant un niveau de trace (i.e. VFS ou bloc d'E/S). Le pseudo code montrant le calcul du taux de lecture/écriture est donné en annexe (voir le pseudo code 7 partie B.2).

### 4.4.4 *Taux de séquentialité*

La façon dont une application accède aux données sur le disque a un impact important sur ses performances, notamment dans le cas des disques durs magnétiques. Dans le contexte de notre étude, les VM n'accèdent généralement pas aux disques virtuels d'une manière séquentielle ou aléatoire. Pour cette raison, de nombreuses approches de caractérisation des charges d'E/S définissent le degré de séquentialité des accès disque (voir partie 3.2). Ce paramètre est appelé taux de séquentialité, et permet de caractériser le motif d'accès d'une charge d'E/S. Dans notre contexte, l'extraction du motif d'accès d'une charge d'E/S d'une VM se fait en parcourant sa trace d'E/S sur un niveau de trace défini. Un pseudo code détaillant le calcul du taux de séquentialité est présenté en annexe (voir le pseudo code 8 partie B.2).

### 4.4.5 *Taille de requête d'E/S*

La taille moyenne des requêtes d'E/S est un paramètre communément utilisé dans la caractérisation des charges d'E/S (voir 3.2). Une moyenne des tailles de requêtes n'est pas totalement pertinente, notamment dans le cas d'un écart type important. Pour caractériser la taille des requêtes d'E/S, nous avons utilisé un histogramme [37]. Ce type de représentation est utilisé dans la plupart des travaux de l'état de l'art (voir partie 3.2), et permet de donner une idée précise de la granularité des requêtes d'E/S. Nous présentons en annexe le pseudo code

de calcul de l'histogramme des tailles des requêtes d'E/S (voir le pseudo-code 9 partie B.2).

Dans la pratique, nos expérimentations ont montré qu'une charge d'E/S d'une VM est souvent caractérisée par une taille de requête dominante par niveau de trace (e.g. pour une application de transcodage vidéo, ~90% des requêtes au niveau d'E/S VFS ont une taille de 4Ko). L'utilisation des résultats de cette fonction est détaillée dans la partie 4.4.7 (équation 3).

### 4.4.6 *Taux d'arrivée des requêtes d'E/S*

Par analogie avec le IOPS du coté des périphériques de stockage, le taux d'arrivée de requêtes d'E/S représente un paramètre communément utilisé pour évaluer les performances des applications interagissant avec les systèmes de stockage. Ce paramètre est considéré comme essentiel dans la littérature [11].

Dans notre contexte, l'étape de *monitoring* est effectuée périodiquement pendant une durée de temps déterminée que nous appelons "période de *monitoring*". Le processus de caractérisation de la charge d'E/S des VM est appliqué à l'issue de chaque période de *monitoring*. Le taux d'arrivée de requêtes d'E/S est extrait pour chaque période de *monitoring*. Pour une durée de *monitoring* de $T_{mon}$ unités de temps, le taux d'arrivée de requêtes $rate_{IO}$ est obtenu comme suit :

$$rate_{IO} = \frac{n}{T_{mon}} \tag{2}$$

où le paramètre $n$ représente le nombre total de requêtes d'E/S d'une même taille, exécutées durant le temps de *monitoring* obtenu par $T_{mon}$.

### 4.4.7 *Volume de données traitées*

Comme nous l'avons défini auparavant dans la partie 4.4.1, chaque requête lit/écrit une quantité définie de données (i.e. $req_{size}$). La taille cumulée de l'ensemble des requêtes capturées durant la période de *monitoring* $T_{mon}$ représente le volume de données traité par le périphérique de stockage durant cette période. Ce volume est obtenu en fonction des tailles des requêtes appartenant à la charge d'E/S d'une VM, et du taux d'apparition de cette requête dans l'histogramme des tailles de requêtes.

Dans le cas d'une charge d'E/S avec $n$ requêtes et un histogramme $Hist_{sizes}$ regroupant $m$ tailles de requêtes, le volume de données traitées $data_{amount}$ est obtenu comme suit :

$$data_{amount} = \sum_{i=1}^{m} Hist_{sizes}(req_{size_i}) \cdot n \cdot req_{size_i} \tag{3}$$

Dans le cas des SSD, ce paramètre doit être associé avec le taux de lecture/écriture pour être plus pertinent. En effet, il est important d'avoir une idée du volume de données écrites sur un SSD, afin de prédire son niveau d'usure.

## 4.5 MÉTHODE D'ÉVALUATION ET RÉSULTATS

Afin de quantifier l'impact de l'outil de *monitoring* proposé, une étape de validation et d'évaluation a été réalisée. Cette évaluation a pour objectif de valider le fonctionnement de traceur d'E/S, de quantifier son intrusivité au sein du système, et d'appliquer la caractérisation des charges d'E/S. Cette partie est divisée en deux parties.

La première partie concerne l'évaluation et les résultats du premier prototype du traceur d'E/S présenté dans [152].

La deuxième partie présente l'évaluation et les résultats de la version module noyau Linux du traceur.

Les deux versions du traceur ont été évaluées en suivant la même méthode et en utilisant les mêmes outils. La prochaine partie décrit la méthode d'évaluation des deux versions du traceur d'E/S.

### 4.5.1 *Méthode d'évaluation*

Afin d'évaluer le traceur d'E/S, nous avons fait appel à des *benchmarks* de systèmes de fichiers [101][187]. La figure 21 illustre la méthode suivie pour évaluer les deux versions du traceur d'E/S.

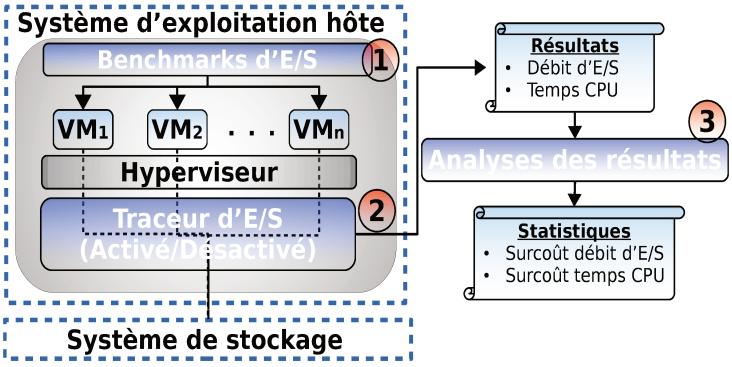


FIGURE 21 : méthode d'évaluation du traceur d'E/S

Cette évaluation comporte trois étapes principales :

1. nous avons d'abord exécuté différents scénarios du *benchmarks* d'E/S sur les VM ;
2. puis nous activons/désactivons le traceur pendant l'exécution des *benchmarks*, et collecte des mesures de performances d'E/S ;

3. enfin, nous analysons et comparons les résultats de performances obtenus en activant/désactivant le traceur.

### 4.5.2 *Outils de benchmarking et configuration*

Après avoir décrit la méthode d'évaluation de notre contribution, cette partie présente la configuration logicielle et matérielle utilisées pour évaluer les outils du *monitoring*.

#### 4.5.2.1 *Configuration des benchmarks*

L'évaluation du traceur a été réalisée à l'aide de deux types de *benchmarks* de systèmes de fichiers : le micro-*benchmark Postmark* et le macro-*benchmark Filebench* (voir partie 2.4). Les deux *benchmarks* ont été choisis pour les raisons suivantes :

- *Postmark* fournit des scénarios simples de traitement de systèmes de fichiers (i.e. création des fichiers, lecture/écriture, suppression des fichiers) ;
- *Filebench* fournit des scénarios émulant des charges d'E/S complexes prédéfinies et personnalisables (e.g. serveur web, serveur de fichiers, serveur de base de données, etc).

Dans la configuration des *benchmarks*, deux paramètres doivent être définis : 1) la taille moyenne des fichiers traités, et 2) la taille moyenne des requêtes.

Pour *Postmark*, il faut spécifier également le nombre de fichiers et le nombre de transactions à effectuer sur ces fichiers. Nous les avons fixés à 50000 et 300000 respectivement.

Le tableau 5 montre la configuration utilisée pour les deux *benchmarks* :

| ***Benchmark*** | **Taille moyenne de fichiers** | **Taille de requête d'E/S** |
|---|---|---|
| Postmark | 1KB-10KB | 512B |
| FileBench FileServer | 2MB | 2KB |
| FileBench Mail server | 16KB | 16KB |
| FileBench Web Server | 16KB | 1MB |

TABLE 5 : configurations des benchmarks

Les configurations présentées dans le tableau 5 ont été choisies afin d'avoir des expérimentations avec des durées moyennes équivalentes, couvrant le temps total de *monitoring*.

#### 4.5.2.2 *Configuration de la plateforme expérimentale*

L'évaluation du traceur d'E/S a été effectuée en utilisant trois VM. Toutes les VM exécutent la même charge d'E/S simultanément. Les VM utilisées pour les tests

ont la même configuration : 1 CPU virtuel, 1GB de mémoire et une image de taille de 8GB qui représente leur disque virtuel. Les VM s'exécutent sur une seule PM. Les VM ainsi que la PM utilisent Linux comme système d'exploitation, avec la version 3.0.2 du noyau Linux. L'hyperviseur utilisé pour le support de la virtualisation est KVM [109]. La PM utilisée pour les tests est une `Dell Precision T3610` avec un processeur Intel Xeon 4 cœurs cadencés à 3.7 GHz, 16GB de mémoire et un disque dur de 1TB.

La prochaine partie évalue la première version du tracer ainsi que les résultats obtenus.

### 4.5.3 *Résultats et évaluation de la première version (libvirt, strace, blktrace)*

Dans cette partie, nous présentons dans un premier les traces obtenues après l'exécution des expérimentations. Ensuite, nous évaluons l'intrusivité du traceur sur le système hôte.

L'objectif de cette partie est d'étudier les traces de différents applicatifs, ainsi que la quantification de l'impact du traceur en fonction de nombre de VM exécutées par le système hôte.

#### 4.5.3.1 *Résultats et discussion sur la première version du traceur*

Le tableau 6 donne les résultats obtenus après l'analyse des traces E/S, en terme de nombre de requêtes d'E/S capturées sur chaque niveau de traces.

Les résultats du tableau 6 ont été obtenus en analysant les traces d'une durée de 20 secondes de trois VM (i.e. **VM1**, **VM2**, et **VM3**). Cette durée est suffisante car elle comprend le temps d'expiration des données en mémoire [186] ("`/proc/sys/vm/dirty_expire_centisecs`" et "`/proc/sys/vm/dirty_writeback_centisecs`"), et nous permet également de minimiser l'intrusivité du tracer.

Les statistiques obtenues montrent qu'il y a un ratio d'écriture/lecture de 1% au niveau bloc pour quasiment tous les *benchmarks*. La deuxième observation concerne le nombre total de requêtes par niveau de trace. En effet, nous pouvons constater l'effet de cache en observant le rapport entre le nombre de requêtes au niveau de l'hyperviseur et au niveau bloc.

Les quatre figures 22, 24, 23, et 25 montrent les blocs accédés en lecture et en écriture, lors de l'exécution de quatre scénarios, à savoir *postmark*, serveur web, serveur de fichiers, et serveur mail respectivement.

La première observation concerne le motif d'accès pendant l'exécution de chacun de ces scénarios. Nous constatons que la VM exhibe un motif d'accès différent pour chaque scénario exécuté. Pour le benchmark *postmark*, nous pouvons observer trois phases de motifs d'accès. La première phase (de l'intervalle de 0µs à 300µs), la VM effectue des accès aléatoires. La deuxième phase (de 300µs

| *Benchmark* | Niveau de trace | Type E/S | VM1 | VM2 | VM3 |
|---|---|---|---|---|---|
| *Postmark* | Hyperviseur | Lecture | 56014 | 51562 | 66864 |
| | | Écriture | 1143842 | 1134154 | 1167091 |
| | VFS | Lecture | 306 | 367 | 345 |
| | | Écriture | 304 | 856 | 732 |
| | Bloc | Lecture | 3400 | 3028 | 4204 |
| | | Écriture | 46 | 46 | 49 |
| *FileBench*-web | Hyperviseur | Lecture | 52010 | 51674 | 52360 |
| | | Écriture | 640 | 594 | 610 |
| | VFS | Lecture | 1598 | 1511 | 1865 |
| | | Écriture | 487 | 454 | 571 |
| | Bloc | Lecture | 3588 | 3191 | 3321 |
| | | Écriture | 23 | 18 | 23 |
| *FileBench*-mail | Hyperviseur | Lecture | 52388 | 51898 | 51912 |
| | | Écriture | 718 | 713 | 724 |
| | VFS | Lecture | 346 | 354 | 294 |
| | | Écriture | 173 | 174 | 173 |
| | Bloc | Lecture | 840 | 837 | 842 |
| | | Écriture | 49 | 51 | 50 |
| *FileBench*-FS | Hyperviseur | Lecture | 52783 | 51492 | 52248 |
| | | Écriture | 634 | 632 | 602 |
| | VFS | Lecture | 799 | 809 | 783 |
| | | Écriture | 201 | 199 | 200 |
| | Bloc | Lecture | 3587 | 3093 | 3229 |
| | | Écriture | 51 | 48 | 46 |

TABLE 6 : nombre d'E/S par niveau de trace

à ~2200µs) est caractérisée par un accès séquentiel. Lors de la dernière phase, la VM effectue à nouveau des accès aléatoires.

Pour analyser plus en détail un des exemples précédents, la figure 26 reprend une partie de la figure 23 présentant les accès bloc lors de l'exécution du scénario du serveur de fichiers.

De la figure 26, nous pouvons distinguer les opérations d'écriture (en rouge) et de lecture (en vert). Les accès bloc peuvent être regroupés en trois parties A, B, et C. Nous remarquons que les lectures se font sur un groupe de blocs tout au

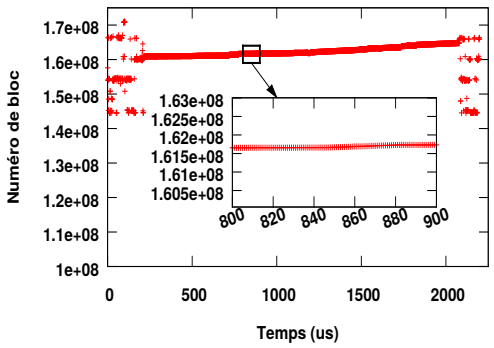


FIGURE 22 : blocs accédés *postmark*

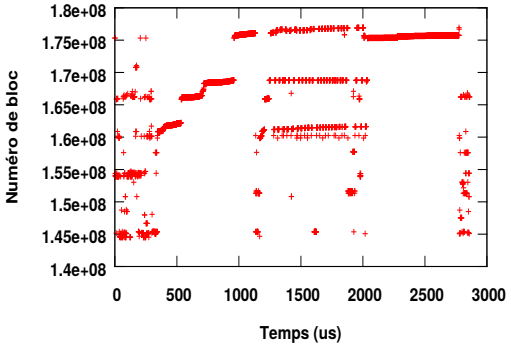


FIGURE 24 : blocs accédés *FileBench-web*

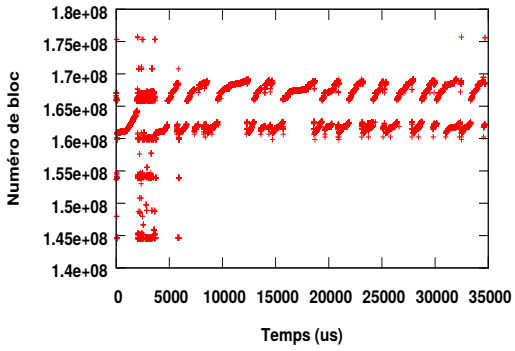


FIGURE 23 : blocs accédés *FileBench-FS*

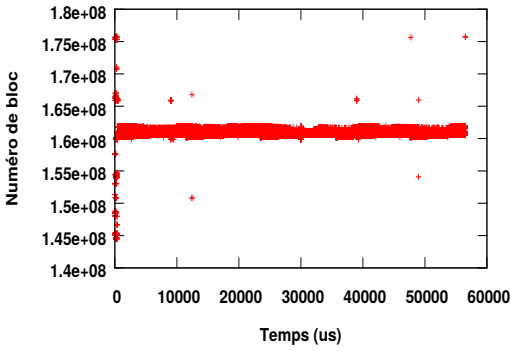


FIGURE 25 : blocs accédés *FileBench-mail*

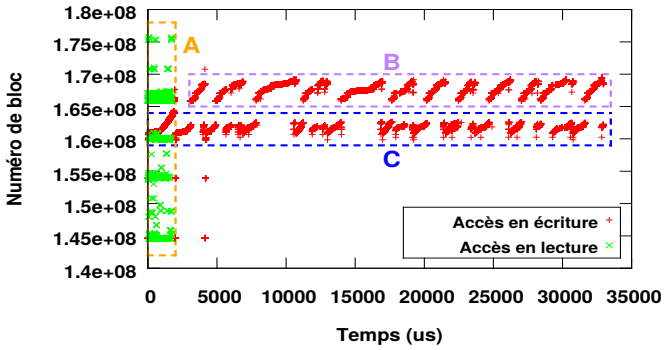


FIGURE 26 : analyse des blocs accédés *file server*

début de l'exécution (partie A). Cela est due aux opérations de lectures des métadonnées lors de la création des fichiers par *filebench*. Le motif d'accès devient par la suite plus aléatoire entre les deux groupes de blocs (parties C et B), avec des requêtes entrelacées entre les deux groupes de blocs.

Dans cette partie, nous avons présenté des exemples des traces obtenues à l'aide du traceur. Nous avons constaté que les caractéristiques des E/S (e.g. lecture/écriture, séquentiel/aléatoire) des différents niveaux dépendent de l'applicatif exécuté par les VM.

La prochaine partie étudie l'intrusivité du traceur sur le système hôte.

#### 4.5.3.2 *Évaluation de l'intrusivité du traceur*

L'évaluation du traceur d'E/S a été effectuée en exécutant les *benchmarks* présentés précédemment. Nous avons par la suite extrait des informations sur le débit de transfert des données et le temps CPU (en sortie de *Filebench*) de plusieurs VM, avec et sans utilisation du traceur.

Nous avons exécuté *Filebench* avec les scénarios des serveurs mail, web, et serveur de fichiers, dans une puis trois VM.

Les expérimentations avec une seule VM nous permettent de voir le surcoût due au traceur. Ce surcoût est calculé pour deux métriques de performances (i.e. débit de transfert de données et temps CPU), et quantifie la baisse des performances qui peut engendrer le processus de traces.

Les expérimentations avec trois VM nous permettent de voir si l'impact du traceur évolue en fonction du nombre de VM tracées.

Les figures 27, 29, 28, et 30 montrent la moyenne et l'écart type des métriques de performances (i.e. débit de transfert de données et temps CPU), obtenues pour 5 séries d'expérimentations.

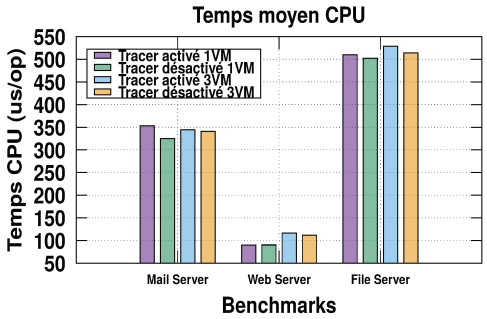


FIGURE 27 : temps CPU moyen

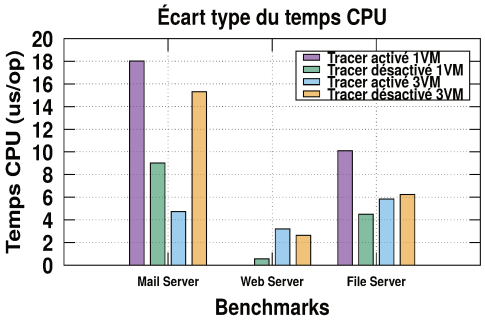


FIGURE 28 : écart type du temps CPU moyen

Concernant le temps CPU (figure 27), et mis à part le cas du *benchmark* serveur mail, nous constatons que l'état du traceur (activé/désactivé), et le nombre de VM en exécution (1 ou 3), n'ont pas d'impact considérable sur le fonctionnement du système. Le faible écart type du temps CPU (figure 28) qui est inférieur à 10 µs/op dans la majorité des cas, confirme la stabilité du système quel que soit l'état du traceur et le nombre de VM.

La figure 29 montre qu'il y a une forte dépendance entre le débit de transfert de données, et la nature du *benchmark* exécuté. Pour tous les cas d'étude, nous constatons que le traceur n'affecte quasiment pas le fonctionnement du système. La figure 30 vient confirmer la stabilité du système avec la traceur avec de plusieurs VM en d'exécution.

Pour conclure, les résultats obtenus montrent un surcoût sur le débit de transfert de données et le temps CPU en dessous de 10% dans la majorité des scénarios,

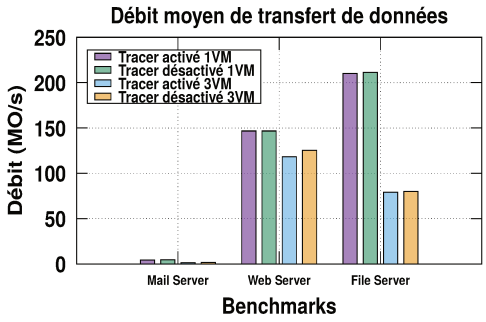


FIGURE 29 : débit moyen

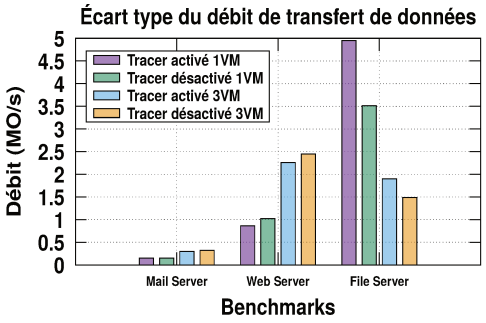


FIGURE 30 : écart type du débit moyen

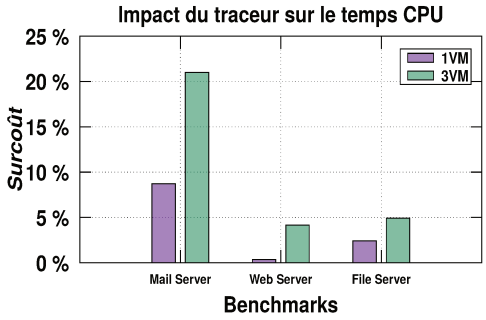


FIGURE 31 : surcoût du temps CPU

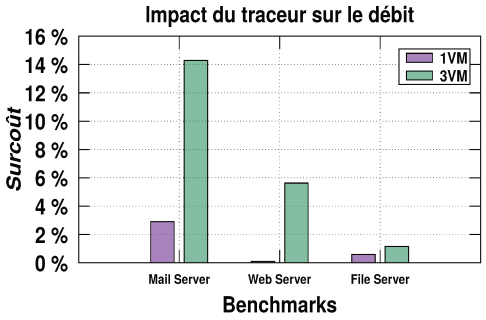


FIGURE 32 : surcoût du débit

à l'exception du scénario du serveur mail (14% de surcoût sur le débit, et 21% sur le temps CPU).

Ces surcoûts sont principalement dues à l'activité élevée sur ce scénario (1000 fichiers, 16 threads, et des requêtes de 16Ko), ce qui implique une activité élevée au niveau du traceur. Ce scénario montre également un écart type important (0.32 Mo/s pour un débit de 1.4 Mo/s), ce qui signifie que la haute activité du traceur perturbe le comportement du système.

### 4.5.4 *Résultats et évaluation de la version module noyau Linux*

Cette partie évalue la deuxième version du traceur en conservant la même procédure expérimentale et les mêmes configurations utilisées lors de l'évaluation de la première version.

Cette partie montre également l'application de la caractérisation des charges d'E/S en utilisant les traces obtenues. L'objectif est de donner un exemple d'utilisation des deux phases de l'étape de *monitoring* (i.e. surveillance et caractérisation des charges d'E/S).

Ensuite, nous évaluons l'intrusvité de l'outil dans le système hôte.

#### 4.5.4.1 *Résultats et discussion sur la deuxième version du traceur*

Cette partie présente les résultats du *monitoring* et la caractérisation des charges d'E/S des VM. Pour la caractérisation, trois paramètres sont présentés :

1. le taux de lecture ;
2. le taux de séquentialité ;
3. la taille des requêtes d'E/S.

Pour chaque paramètre, nous présentons les résultats obtenus en exécutant trois scénarios du *benchmarck Filebench* et du benchmark *Postmark*. Les statistiques obtenues sont également classées par niveau de trace (i.e. bloc d'E/S et VFS).

**Taux de lecture/écriture**

La figure 33 montre le taux d'opérations de lectures et écriture après l'analyse des traces des VM exécutant quatre scénarios de charge d'E/S : 1) Serveur de fichiers, 2) Serveur web, 3) Serveur mail, et 4) Postmark.

Nous remarquons que ces *benchmarks* présentent différents profil d'E/S. Dans les cas du serveur de fichiers et du serveur web, nous pouvons constater le haut taux de lecture, spécialement au niveau VFS (autours de 40%). Ces lecture ont un faible impact au niveau bloc ce qui est expliqué par l'utilisation des caches au niveau VFS.

Ce type d'observation peut guider le choix sur le type de périphérique de stockage. Dans ce cas, le stockage sur HDD et SSD donnent les mêmes performances, car la majorité des E/S se font au niveau VFS.

Pour le serveur mail et *postmark*, nous pouvons observer un taux d'écriture important dans les deux niveaux VFS et bloc ($\simeq$ 100%). Dans ce cas, l'application n'est pas adaptée aux SSD si nous voulons préserver leur durée de vie.

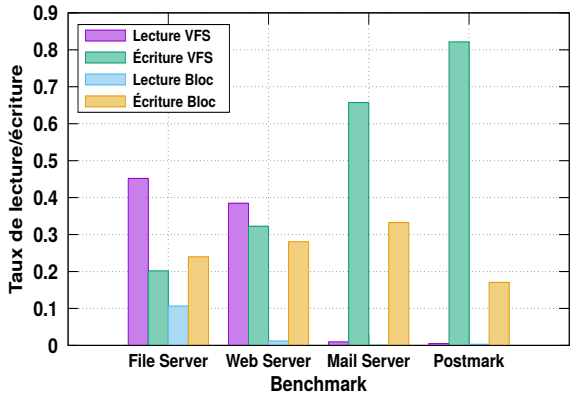


FIGURE 33 : taux de lecture selon les niveaux de traces

### Taux de séquentialité

La figure 34 montre le taux de séquentialité obtenu en analysant les mêmes traces résultant de l'exécution des charges d'E/S.

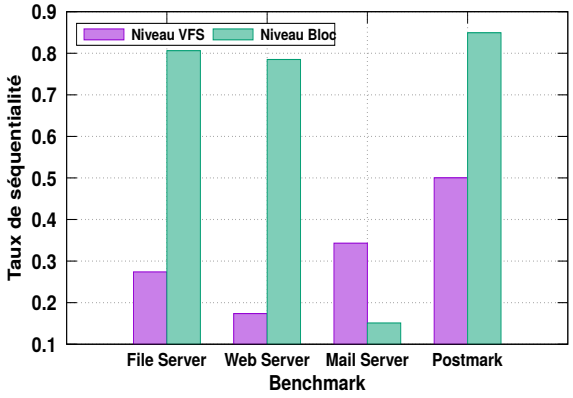


FIGURE 34 : taux de séquentialité selon les niveaux de traces

La figure 34 montre un comportement différent de celui du taux de lecture/écriture, où trois des quatre scénarios présentent un accès séquentiel important au niveau bloc d'E/S, contrairement au niveau VFS.

Le taux de séquentialité élevé et le bas taux de lecture présent dans ces charges d'E/S peuvent favoriser l'utilisation des disques durs magnétiques, dans le but de minimiser les coûts.

### Taille des requêtes d'E/S

La taille des requêtes d'E/S représente un paramètre important dans la caractérisation des charges d'E/S. Les figures 35 et 36 présentent les histogrammes des tailles de requêtes respectivement aux niveaux VFS et bloc.

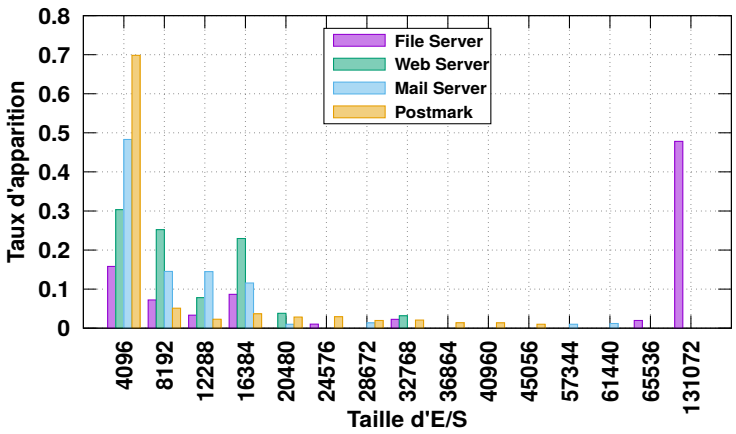


FIGURE 35 : histogrammes de la taille des requêtes au niveau VFS

Nous observons que tous les *benchmarks* utilisés dans nos expérimentations utilisent des requêtes de petite taille, à l'exception du serveur de fichiers. Les tailles

de requêtes dominantes se concentrent sur la première partie de l'histogramme (entre 4Ko et 16Ko).

Pour les scénarios du serveur de fichiers, serveur mail, et *postmark*, nous constatons que plus de 50% des requêtes sont de taille de 4Ko et le reste des requêtes est répartis sur les trois tailles qui suivent (i.e. 8Ko, 12Ko et 16Ko). Ces phénomènes résultent aussi de la configuration des *benchmarks*, ce qui confirme que nous pouvons retrouver les mêmes caractéristiques E/S dans les VM et dans la PM.

Au niveau bloc, nous constatons que les tailles des requêtes restent entre 4Ko et 2Mo, à l'exception du serveur de fichiers (voir figure 36).

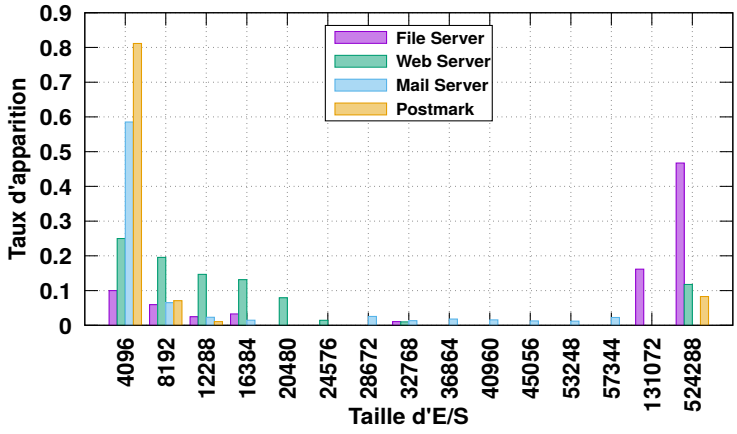


FIGURE 36 : histogrammes de la taille des requêtes au niveau bloc

La taille des requêtes d'E/S dans le cas des scénarios serveur de fichiers, serveur web, et particulièrement *postmark*, explique le taux de séquentialité important observé au niveau bloc (voir figure 34). Dans le cas du serveur mail, la petite taille de requête dominante (65% de taille 4Ko) explique aussi son bas taux de séquentialité de 15%.

#### 4.5.4.2 *Évaluation de l'intrusivité du traceur*

L'évaluation de la version module noyau du traceur d'E/S a été effectuée en suivant le même processus que pour la première version. L'objectif de cette évaluation est la quantification de l'impact du traceur d'E/S sur le système hôte.

Pour ce faire, deux métriques évaluées par le *benchmark filebench* ont été utilisées : 1) le débit de transfert de données, et 2) le temps CPU. La figure 37 montre le surcoût causé par le traceur d'E/S en exécutant les trois scénarios de *filebench*, avec une VM et trois VM.

La figure 37 montre le surcoût du traceur d'E/S, en terme de débit et de temps CPU. Les résultats obtenus montrent un impact sur l'exécution des charges d'E/S par les VM qui ne dépasse pas les 2.5% dans toutes les expérimentations avec une moyenne de ~2% et ~1.2% dans le meilleur cas.

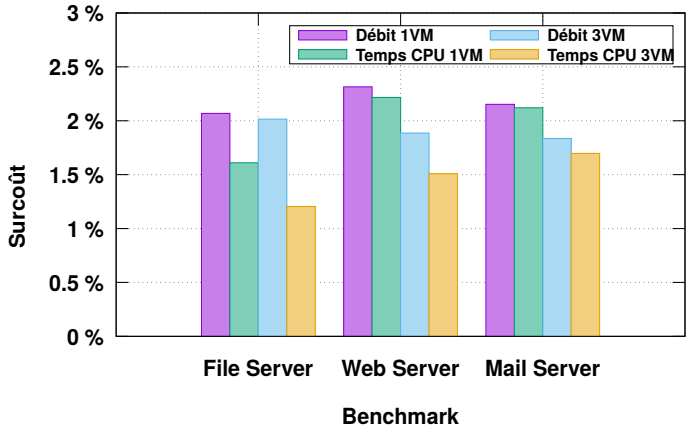


FIGURE 37 : surcoût du traceur sur le débit et le temps CPU

Nous constatons également que ce surcoût diminue dans le cas de plusieurs VM en exécution. Cela représente un résultat encourageant, et montre une réduction d'intrusivité par rapport à la première version du traceur.

## 4.6 CONCLUSION

Dans ce chapitre, nous avons présenté les éléments essentiels constituant l'étape *Monitor* de notre implantation du modèle MAPE-K. Cette première étape est principalement constituée de deux sous-étapes :

1. *monitoring* des opérations d'E/S des VM ;
2. caractérisation des charges E/S des VM.

Premièrement, nous avons présenté l'étape de *monitoring* des E/S, sous la forme d'un outil de proposé pour tracer les accès des VM aux périphériques de stockage. Cet outil a été conçu et développé en deux versions. La première version permet de tracer les accès des VM sur différents niveaux de la pile logicielle d'E/S :

- hyperviseur des VM ;
- système de fichiers virtuel VFS ;
- couche bloc d'E/S.

Cette version combine différents outils de trace et bibliothèques disponibles pour les systèmes Linux. Ces outils et bibliothèques ont été utilisés dans le but de produire des traces d'E/S prêtes à être exploitées dans l'étape de la caractérisation des charges d'E/S.

La deuxième version a été développée sous la forme d'un module noyau Linux. Cette version opère à deux niveaux : 1) niveau VFS, et 2) niveau bloc E/S.

Ensuite, nous avons introduit un modèle de caractérisation des charges d'E/S des VM. Ce modèle permet d'établir les profils d'E/S pour VM en utilisant les traces d'E/S produites par l'outil de *monitoring*.

Les outils proposés ont été évalués à l'aide des *benchmarks* d'E/S. Cette évaluation a deux deux objectifs :

- étudier les traces produites par l'outil de *monitoring*, ensuite, les utiliser pour la caractérisation des charges d'E/S ;
- quantifier le surcoût d'utilisation de l'outil de *monitoring* dans le système hôte.

L'évaluation de la deuxième version de l'outil de *monitoring* a montré un surcoût moyen de 2% en termes de débit et d'utilisation CPU.

Les outils proposés nous permettent d'achever la première étape *M* du modèle MAPE-K, et de produire les éléments nécessaires pour les étapes suivantes.

# Chapitre 5

## ÉTAPE *ANALYZE* : MODÉLISATION DES COÛTS DE STOCKAGE HYBRIDE POUR LES VM

Notre approche s'inscrit dans une démarche suivant le modèle MAPE-K d'auto adaptation appliqué à l'optimisation de placement de données des VM. Comme tout problème d'optimisation, le placement de données des VM s'appuie sur une ou plusieurs fonctions objectifs à minimiser ou à maximiser.

Une fonction objectif a pour but d'évaluer une solution d'optimisation appliquée au problème étudié. Dans le domaine de l'optimisation (plus précisément dans le cas d'un problème de minimisation), la fonction objectif est également appelée fonction-coût. Cette dernière doit regrouper tous les coûts qui peuvent résulter d'une configuration donnée (une solution du problème).

Pour la minimisation des coûts de placement des données des VM, notre fonction de coût doit inclure tous les coûts fixes et variables induits par le stockage des données d'une VM sur un périphérique de stockage. Nous devons déterminer tout les paramètres à prendre en compte et qui peuvent avoir un impact sur les coûts de stockage, et par conséquent sur la décision à prendre dans le problème d'optimisation. Tout d'abord, nous devons modélisons les caractéristiques de notre système de stockage en prenant en considération tous ses paramètres. Deuxièmement, nous intégrons les contraintes liées à l'environnement virtualisé, car il a été prouvé que la virtualisation peut avoir un impact important sur les systèmes de stockage [174]. Enfin, la modélisation des coûts de stockage doit aussi tenir compte des besoins des clients du service *cloud*, en termes de qualité de service spécifiée dans le SLA.

De nombreux modèles de coûts ont été proposés dans l'état de l'art. La partie 3.3 présente une étude des travaux de la littérature avec une classification des approches (voir tableau 2).

Après exploration des coûts liés au stockage des données des VM, nous avons constaté que les coûts dépendent de trois facteurs principaux :

1. *la VM :* le profil d'accès des E/S et stockage virtuel

2. *le système de stockage :* les types et les caractéristiques des périphériques de stockage

3. *l'environnement cloud :* le prix des services *cloud* et les contraintes de qualité de service

Certains coûts fixes ne dépendent pas des 3 facteurs listés ci-dessus (e.g. construction ou location des locaux, ressources humaines). Ce type de coûts n'est pas détaillé dans notre étude.

Dans ce chapitre, nous proposons un modèle de coût évaluant le stockage des VM sur un système de stockage hybride. Le modèle prend en compte les trois facteurs présentés précédemment afin d'évaluer précisément les coûts de stockage des données des VM.

Nous avons évalué la partie traitant la consommation énergétique de notre modèle sur une plateforme réelle avec un système de stockage hybride. Dans le cas d'un périphérique de stockage de type HDD, notre modèle montre un taux d'erreur de 0.04% dans le meilleur cas, et de 8% dans le pire cas. Pour un périphérique de stockage de type SSD, le taux d'erreur est de 2% dans le meilleur cas, et de 15% dans le pire cas.

Notre contribution présentant ce modèle de coût a été publiée dans *"24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)"* [153].

Ce chapitre est organisé comme suit. Premièrement nous donnons un aperçu global sur le modèle de coût et ses principaux paramètres. La deuxième, la troisièmes, et la quatrième partie détailles les différentes les composants du modèle de coûts. Nous présentons l'évaluation de notre modèle dans la cinquième partie avant de conclure ce chapitre.

## Sommaire

## 5.1 APERÇU SUR MODÈLE DE COÛT

Tout d'abord et avant de détailler les différentes parties du modèle de coût, nous allons définir les concepts et les termes utilisés dans notre modélisation.

**Définition 5.1.** *(**stockage de** VM) [95] : le stockage d'une VM désigne le ou les fichiers des images disques représentant les disques virtuels.*

**Définition 5.2.** *(**migration de** VM) [143] : c'est l'opération qui consiste à transférer une VM en cours d'exécution d'une PM (source) à une autre (destination). Ce concept est également appliqué au niveau des systèmes de stockage, entre un périphérique source et un autre destination [97].*

La figure 38 présente un aperçu de notre modèle de coût proposé sous une forme hiérarchique.

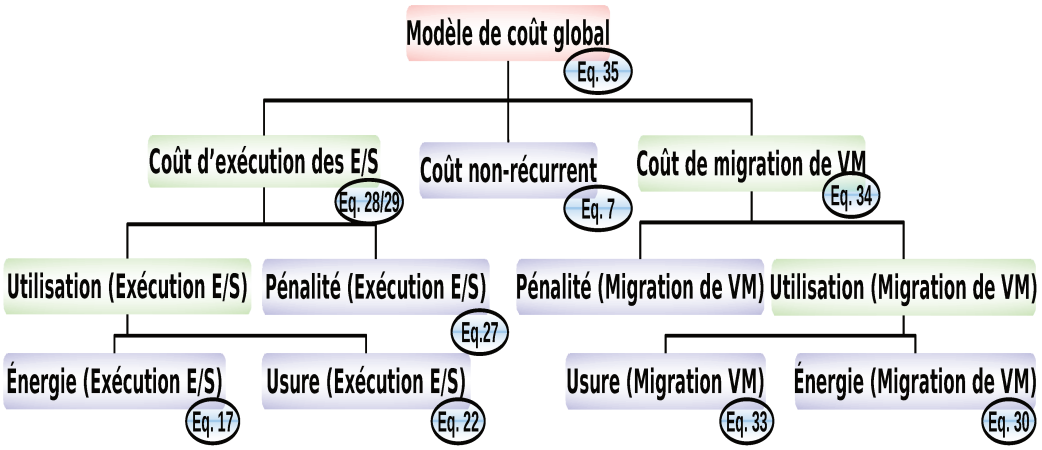


FIGURE 38 : diagramme du modèle de coût global

La figure 38 montre les principaux composants de notre modèle, ainsi que les numéros des équations détaillant le coût correspondant.

L'équation 4 correspond au point d'entrée qui représente le coût global de stockage (noté Cost) d'une VM (noté VM) sur un périphérique de stockage (noté D) :

$$\mathrm{Cost}(\mathrm{VM}, \mathrm{D}) = \mathrm{Cost}_{\mathrm{exe}}(\mathrm{VM}, \mathrm{D}) + \mathrm{Cost}_{\mathrm{mig}}(\mathrm{VM}, \mathrm{D}) + \mathrm{Cost}_{\mathrm{nrc}} \tag{4}$$

où $\mathrm{D} \in \{\mathrm{HDD}, \mathrm{SSD}\}$.

Comme détaillé par l'équation 4, pour une période de *Monitoring* donnée, le stockage d'une VM sur un périphérique de stockage est composé de trois coûts :

1. $\mathrm{Cost}_{\mathrm{nrc}}$ (équation 5)correspond aux coûts non-récurrents incluant les frais qui ne dépendent pas de l'exécution des VM sur l'infrastructure matérielle (location des locaux, ressources humaines, etc) ;

2. $\mathrm{Cost}_{exe}$ (équations 26 et 27) représente le coût de l'exécution de la charge d'E/S d'une VM sur un périphérique de stockage ;

3. $\mathrm{Cost}_{mig}$ (équation 32) est le coût de migration d'une VM à partir d'un périphérique de stockage vers un autre.

Dans les partie suivantes, nous allons détailler chacun des coûts composant le modèle de coût global.

## 5.2 COÛTS NON-RÉCURRENTS

Le coût non-récurrent correspond à tous les coûts fixes qui ne dépendent pas du stockage des données des VM ni de l'environnement *cloud*. Cette partie peut inclure les coûts de l'installation des ressources matérielles, la location des locaux des centres de données, les coûts des ressources humaines, etc.

Dans notre modèle, nous considérons que le coût non-récurrent est constant pour l'ensemble des VM présentes dans le centre de données :

$$\mathrm{Cost}_{nrc} = \mathrm{NRC} \tag{5}$$

où NRC (*Non-Recurring Cost*) est un constant.

## 5.3 COÛTS D'EXÉCUTION DES CHARGES D'E/S DES VM

Le coût d'exécution des VM désigne tous les coûts induits par l'exécution de la charge d'E/S d'une VM donnée, sur le périphérique de stockage contenant son stockage virtuel (i.e. les images disques).

L'équation 6 montre le coût d'exécution d'une VM, du 1) coût total du cycle de vie (noté $\mathrm{Cost}_{e\_tco}$), et du 2) coût de pénalité (noté $\mathrm{Cost}_{e\_pen}$) :

$$\mathrm{Cost}_{exe}(VM, D) = \mathrm{Cost}_{e\_tco}(VM, D) + \mathrm{Cost}_{e\_pen}(VM, D) \tag{6}$$

Dans les parties suivantes, nous allons détailler chaque terme de l'équation 6.

### 5.3.1 *Coût total du cycle de vie d'exécution des charges d'E/S des VM*

Pour un périphérique de stockage, le coût total du cycle de vie (ou TCO pour *Total Cost of Ownership*) inclut l'ensemble des dépenses liées à l'utilisation du périphérique, tout en excluant le coût d'achat des ressources qui est pris en compte dans le coût non-récurrent.

Comme le montre l'équation 7, le coût du cycle de vie comporte le coût énergétique (noté $\mathrm{Cost}_{e\_egy}$) et le coût de l'usure du périphérique de stockage (noté $\mathrm{Cost}_{e\_wo}(VM, D)$).

$$\mathrm{Cost}_{e\_tco}(VM, D) = \mathrm{Cost}_{e\_egy}(VM, D) + \mathrm{Cost}_{e\_wo}(VM, D) \tag{7}$$

Le coût énergétique désigne la facturation de la consommation du périphérique de stockage (noté D), en exécutant la charge d'E/S de la VM (noté VM). Le coût de l'usure donne une indication sur l'état de santé du périphérique de stockage D, après avoir exécuté la charge d'E/S de la VM. En effet, la durée de vie d'un périphérique de stockage dépend essentiellement de la charge d'E/S appliquée à ce dernier [10] (voir partie 5.3.1.2).

#### 5.3.1.1 *Coût énergétique*

Le coût énergétique (noté $Cost_{e\_egy}$) est obtenu à partir : 1) de la quantité d'énergie (notée $E_{tot}$) consommée par le périphérique de stockage D, durant l'exécution de la charge d'E/S de la VM, 2) et du prix unitaire de l'énergie électrique (noté $E_{UP}$) :

$$Cost_{e\_egy}(VM, D) = E_{tot} \cdot E_{UP} \tag{8}$$

Le prix unitaire de l'énergie $E_{UP}$ utilisé dans l'équation 8 peut être obtenu auprès des fournisseurs d'électricité.

Dans notre étude, nous supposons que la quantité d'énergie consommée durant l'exécution de la charge d'E/S (notée $E_{tot}$) est la somme des énergies consommées par tous les composants (parcourus par j) contribuant à l'exécution de cette charge d'E/S (i.e. CPU, RAM, et disques) . Dans le cas de n composants contribuant à l'exécution de la charge d'E/S d'une VM, nous avons :

$$E_{tot} = \sum_{j=1}^{n} E(VM, j) \tag{9}$$

où $E(VM, j)$ est l'énergie consommée par le composant j.

Nous considérons exclusivement la consommation énergétique des opérations d'E/S et du système de stockage. La modélisation de la consommation énergétique du CPU et de la RAM pour les autres types d'opération n'est donc pas prise en compte par notre modèle de coût.

Pour chaque périphérique de stockage, l'énergie consommée durant l'exécution de la charge d'E/S d'une VM est obtenue à partir de la puissance électrique appliquée et le temps d'exécution. Pour un périphérique j contribuant à l'exécution de la charge d'E/S d'une VM, l'énergie consommée $E(VM, j)$ est obtenue en utilisant sa puissance électrique $P_j$ et la période de monitoring $T_{mon}$ de la VM :

$$E(VM, j) = P_j \cdot T_{mon} \tag{10}$$

Afin de simplifier la modélisation, tout au long de la présentation du modèle de coût, nous supposons que l'image d'une VM représentant son stockage est stockée sur un seul périphérique (hypothèse 7 partie 1.4).

Pour un périphérique donné, la puissance électrique dépend de son état durant l'exécution de la charge d'E/S. Par exemple, un HDD peut se trouver dans

l'un des états suivants : 1) *operating mode*, 2) *idle mode*, 3) *standby mode/sleep mode* (voir partie 2.2.2). Un SSD peut avoir des modes de puissance électriques différents de ceux d'un HDD. Pour un SSD [57], les spécifications présentent trois modes avec un *operating mode* et deux *idle mode*.

Si nous prenons le cas d'un HDD , la quantité d'énergie consommée durant la période de *monitoring* (notée $T_{mon}$) est obtenue à l'aide des puissances des différents modes (*operating mode* noté $P_{op}$, *idle mode* noté $P_{idl}$, et *standby mode* noté $P_{stdb}$) et du temps passé dans chaque mode (i.e. $T_{op}$, $T_{idl}$, et $T_{stdb}$ respectivement). Le HDD consomme également de l'énergie durant la transition du *standby mode* à *operating mode*. Cette opération est appelée *spin-up* et l'énergie correspondante est notée $E_{spn}$. L'équation 11 montre comment obtenir la consommation énergétique d'un HDD en fonction de ses états :

$$E(VM, HDD) = (P_{op} \cdot T_{op}) + (P_{idl} \cdot T_{idl}) + (P_{stdb} \cdot T_{stdb}) + (n \cdot E_{spn}) \quad (11)$$

où

$$T_{mon} = T_{op} + T_{idl} + T_{stdb}$$

Le paramètre $n$ utilisé dans l'équation 11, désigne le nombre de transitions *spin-up* compté durant la période de *monitoring*. Dans le cas d'un SSD, il n'y a pas de consommation énergétique liée au *spin-up*, mais il existe toutefois deux *idle mode* (*idel mode 1* et *idle mode* 2).

La quantité d'énergie consommée dans le cas d'un SSD est donnée comme suit :

$$E(VM, SSD) = (P_{op} \cdot T_{op}) + (P_{idl1} \cdot T_{idl1}) + (P_{idl2} \cdot T_{idl2}) \quad (12)$$

Pour un périphérique de stockage donné, la puissance électrique en *operating mode* $P_{op}$ 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).

Les HDD présentent une asymétrie des performances par rapport au motif d'accès séquentiel/aléatoire.

Pour les SSD, l'asymétrie des performances n'est pas seulement par rapport au type d'accès en lecture/écriture, mais aussi au motif d'accès [158]. Par conséquent, la quantité de l'énergie consommée en mode opérationnel $E_{op}$ peut être formulée comme suit :

$$E_{op}(VM, D) = \sum_{i}^{seq,rnd} \sum_{j}^{read,wrt} (P_{i,j} \cdot T_{i,j}) \quad \textbf{où} \begin{cases} i \in \{seq, rnd\} \\ \textbf{et} \\ j \in \{read, wrt\} \end{cases} \quad (13)$$

Les valeurs des paramètres $T_{i,j}$ et $P_{i,j}$ peuvent être obtenues de deux façons : à partir des fiches techniques des périphériques de stockage, ou bien à l'aide de mesures.

Dans notre cas, nous avons analysé les traces d'E/S afin d'extraire les valeurs des temps $T_{i,j}$. En ce qui concerne les valeurs de la puissance $P_{i,j}$, nous les avons mesurées en utilisant un dispositif spécifique appelé PDU (pour *Power Distribution Unit*) [161]. Les valeurs de temps des opérations de lecture/écriture séquentielle/aléatoire sont calculées à partir des taux de lecture/écriture opérationnelle/aléatoire $rate_{i,j}$ obtenus à la sortie du processus de caractérisation de la charge d'E/S (voir partie 4.4). L'équation 14 montre le calcul des différentes valeurs de temps en utilisant $rate_{i,j}$ et le temps passé en mode opérationnel $T_{op}$ :

$$T_{i,j} = \frac{rate_{i,j}}{100} \cdot T_{op} \tag{14}$$

Si nous prenons comme exemple le cas d'une VM stockée sur un HDD noté D, le coût énergétique de l'exécution de la VM est :

$$Cost_{e\_egy}(VM, D) = \left[ \sum_{i}^{seq,rnd} \sum_{j}^{read,wrt} \left( P_{i,j} \cdot \frac{rate_{i,j}}{100} \cdot T_{op} \right) + (P_{idl} \cdot T_{idl}) + (P_{stdb} \cdot T_{stdb}) + (n \cdot E_{spn}) \right] \cdot E_{UP} \tag{15}$$

Nous avons détaillé le coût de la consommation énergétique liée à l'exécution de la charge d'E/S d'une VM sur un périphérique de stockage. La prochaine partie présente le coût de l'usure du périphérique de stockage liée à l'exécution de la VM, ce qui complète le coût total de cycle de vie.

#### 5.3.1.2 *Coût de l'usure des périphériques de stockage*

Les modèles de coûts intègrent de plus en plus le coût de l'usure des dispositifs matériels [199][122]. Pour calculer le coût de l'usure d'un périphérique donné, certain travaux de l'état de l'art se servent d'une estimation de la durée de vie de ce périphérique.

La durée de vie d'un HDD dépend de son temps de service (voir partie 2.2.2). En pratique, le coût de l'usure d'un HDD peut être calculé à partir de sa durée de vie garantie par le constructeur, de son prix d'achat, et du prix du service *cloud*.

L'usure est l'une des contraintes les plus importantes pour une mémoire flash. Pour un SSD, celle-ci dépend du nombre de cycles d'écriture/effacement (voir partie 2.2.3). Ainsi, il est plus simple d'estimer la durée de vie du périphérique en ayant connaissance de la quantité totale de données écrites. Tout comme le HDD , le coût de l'usure d'un SSD est liée à sa durée de vie.

Pour récapituler, le coût de l'usure (noté $WO_{cost}$) d'un périphérique de stockage (noté D) stockant une VM (notée VM) dépend de la charge d'E/S de cette dernière :

$$WO_{cost}(VM, D) = \begin{cases} WO_{cost}(NB_{wrt}) & \text{si D est un SSD} \\ WO_{cost}(MTTF/AFR/\#start-stop) & \text{si D est un HDD} \end{cases} \quad (16)$$

où $NB_{wrt}$ désigne le nombre de requêtes d'écriture exécutées par la VM sur le SSD, MTTF, AFR, et $\#start-stop$ désignent les paramètres de l'état de santé du HDD.

Pour un SSD, nous supposons que le SSD est complètement usé et doit être changé dés 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). Dans notre cas, la quantité de données écrite (notée $NB_{wrt}$) par une VM doit être quantifiée afin de calculer l'usure du SSD. D'une manière générale, la quantité de données écrite peut être obtenue comme suit :

$$write_{tot}(VM) = NB_{wrt} \cdot req_{size}$$

où $write_{tot}$ désigne la fonction calculant la quantité de données écrite par la VM, $NB_{wrt}$ désigne le nombre de requêtes d'écriture, et $req_{size}$ désigne la taille des requêtes.

Notre approche utilise le nombre de requêtes d'écriture exécutées par une VM en analysant ses traces au niveau bloc d'E/S. Ce niveau de trace nous permet de ne considérer que les requêtes d'E/S effectivement traitées par le périphérique de stockage (voir partie 4.3.4).

Il est plus significatif de se servir des taux obtenus à partir de la caractérisation de la charge d'E/S, afin de quantifier la quantité de données écrite par une VM. La quantité de données écrite par une VM comme suit :

$$write_{tot}(VM) = (T_{rnd,wrt} \cdot rate_{IO} \cdot req_{size}) + (T_{seq,wrt} \cdot vm_{DTR}(VM)) \quad (17)$$

où $T_{rnd,wrt}$ et $T_{seq,wrt}$ désignent respectivement les temps d'écriture aléatoire et séquentielle calculés à partir de l'équation 14. Les termes $rate_{IO}$ et $vm_{DTR}$ concernent les performances d'E/S de la VM.

À partir de la quantité de données écrite par une VM donnée à l'aide de l'équation 17 nous pouvons à calculer le coût pour chaque mégaoctet (noté $Cost_{per\_mb}$) écrit sur le SSD.

Pour ce faire, nous utilisons l'équation 17 et les caractéristiques du périphérique de stockage. Le coût de l'usure par mégaoctets (noté $Cost_{per\_mb}$) est calculé ainsi :

$$Cost_{per\_mb}(SSD) = \frac{stg_{UP} \cdot stg_{cap}}{MAX_{wrt} \cdot 1024} \quad (18)$$

où $stg_{UP}$ est le prix unitaire d'un gigaoctet du SSD, $stg_{cap}$ est sa capacité, et $MAX_{wrt}$ est la quantité maximale de données à écrire.

Maintenant, nous pouvons estimer le coût de l'usure (équation 7) lié à l'exécution de la charge d'E/S d'une VM sur un périphérique de stockage SSD qui s'exprime comme suit :

$$Cost_{e\_wo}(VM, SSD) = Cost_{per\_mb}(SSD) \cdot write_{tot}(VM) \tag{19}$$

ou encore :

$$Cost_{e\_wo}(VM, SSD) = \left\lceil \frac{stg_{UP} \cdot stg_{cap}}{MAX_{wrt}} \right\rceil \cdot \Big[ (T_{rnd,wrt} \cdot rate_{IO} \cdot req_{size}) + (T_{seq,wrt} \cdot vm_{DTR}(VM)) \Big] \tag{20}$$

Le coût de l'usure d'un HDD n'est pas aussi prévisible que celui d'un SSD, car les paramètres de la durée de vie d'un HDD (présents dans la fiche technique) sont basés sur des probabilités statistiques (e.g. MTTF, MTBF, AFR, etc). Par conséquent, un calcul s'appuyant sur ces paramètres ne sera pas complètement représentatif de l'état d'usure d'un HDD [170].

Dans le cas d'un HDD, il est préférable d'utiliser le nombre de cycles *start-stop* afin d'estimer sa durée de vie. Ce paramètre reflète le caractère mécanique du HDD. Le nombre de cycles *start-stop* est un compteur incrémenté à chaque passage vers/depuis le mode *standby*. Chaque HDD a un seuil de cycles *start-stop*. Une fois ce seuil atteint, il est préférable de changer le HDD afin d'éviter une éventuelle panne. La valeur du compteur de cycles *start-stop* peut être obtenue à l'aide de la technologie S.M.A.R.T (*Self-Monitoring, Analysis and Reporting Technology*), intégrée dans la plupart des HDD modernes.

En appliquant la méthode suivie pour établir l'usure d'un SSD (voir formule 18), le coût d'un cycle *start-stop* d'un HDD peut être obtenu comme suit :

$$Cost_{str-stp}(HDD) = \frac{stg_{UP} \cdot stg_{cap}}{N_{str-stp}} \tag{21}$$

où $stg_{UP}$ est le prix unitaire d'un gigaoctet du HDD, $stg_{cap}$ est sa capacité, et $N_{str-stp}$ est le nombre total des cycles *start-stop* indiqué dans la fiche des spécification du HDD.

Le coût de l'usure d'un HDD $Cost_{e\_wo}$ peut donc être calculé ainsi :

$$Cost_{e\_wo}(VM, HDD) = Cost_{str-stp}(HDD) \cdot NB_{str-stp}(T_{mon}) \tag{22}$$

où $Cost_{str-stp}$ est le coût d'un cycle *start-stop* du HDD, et $NB_{str-stp}$ est une fonction donnant le nombre de cycles *start-stop* effectués durant la période du *monitoring* $T_{mon}$ de la VM.

Enfin, l'équation du coût de l'usure d'un HDD liée à l'exécution de la charge d'E/S d'une VM est établi comme suit :

$$Cost_{e\_wo}(VM, HDD) = \frac{stg_{UP} \cdot stg_{cap}}{N_{str-stp}} \cdot NB_{str-stp}(T_{mon}) \tag{23}$$

Dès lors, nous avons exprimé en détail toutes les parties du coût total de cycle de vie, lié à l'exécution de la charge d'E/S d'une VM. Dans la prochaine étape, nous détaillons le deuxième terme du coût d'exécution de VM, c'est à dire le coût de pénalité.

### 5.3.2 *Coût de pénalité d'exécution des charges d'E/S des VM*

Le SLA est un concept import du *cloud* [67] (voir partie 2.5.1). Nous y trouvons les différents termes garantissant la qualité de service demandée par les clients du *cloud*. Dans le cas où le fournisseur du service *cloud* offre une qualité de service (notée $QoS_{offered}$) inférieure à celle demandée par le client (notée $QoS_{requested}$), alors le fournisseur doit dédommager le client sous la forme d'une pénalité. La pénalité est un taux (notée $rate_{pen}$) qui réduit la facture du client (notée $Bill$) et qui dépend du prix unitaire du service *cloud*. Le montant de la facture recalculée $Bill_{tot}$ s'exprime comme suit :

$$QoS_{offered} < QoS_{requested} \Rightarrow Bill_{tot} = (Bill(VM, D) \cdot CS_{UP}) - rate_{pen} \quad (24)$$

Trois méthodes existent pour calculer le montant de la pénalité [68] :

1. **Pénalité fixe** : dans ce cas de figure, $rate_{pen}$ est constant et doit être retranché de la facture à chaque violation du SLA ;
2. **Pénalité dépendante du délai** : le montant $rate_{pen}$ 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 $rate_{pen}$ dépend du ratio entre la qualité de service demandée et celle offerte.

Les fournisseurs du *cloud* utilisent une méthode mixant pénalité fixe et pénalité proportionnelle [74][139][172]. Ils définissent des intervalles de qualité de service et fixent les pénalités correspondantes. Le tableau 7 montre un exemple du calcul de la pénalité mixte :

| **Intervalle de $QoS_{offered}$** | **Taux de pénalité** |
|---|---|
| Supérieure à $B_0$ | 0 |
| $[B_i - B_{i+1}]$ | $rate_{pen_i}$ |
| Inférieure à $B_n$ | $rate_{pen_n}$ |

TABLE 7 : exemple d'un modèle de calcul de pénalité utilisé par les fournisseurs *cloud*

Dans le tableau 7, les paramètres $B_0, \ldots, B_n$ représentent les bornes des intervalles de la qualité de service offerte, et $rate_0, \ldots, rate_n$ sont les pénalités fixes correspondantes.

Pour notre modèle, nous choisissons d'appliquer la pénalité proportionnelle car elle est la plus équitable pour le calcul de la pénalité. En effet, le montant de la pénalité dépend du degré de violation des termes du SLA. Cela conduit au calcul du coût de pénalité $Cost_{e\_pen}$ qui s'exprime comme suit :

$$Cost_{e\_pen}(VM, D) = rate_{pen} \cdot Bill(VM, D) \quad \textbf{où} \quad \begin{cases} rate_{pen} \geqslant 0 \\ \textbf{et} \\ rate_{pen} = 1 - \frac{QoS_{offered}}{QoS_{requested}} \end{cases} \tag{25}$$

Les paramètres de la qualité de service $QoS_{offered}$ et $QoS_{requested}$ peuvent être représentés par le débit (en IOPS) ou le taux de transfert de données (bande passante en Mo/s), autrement dit :

- $QoS_{offered}$ : $dev_{IOPS}$ ou $dev_{DTR}$
- $QoS_{requested}$ : $req_{IOPS}$ ou $req_{DTR}$

où $dev_{IOPS}/dev_{DTR}$ sont les paramètres de la QoS offerts par les périphériques de stockage, et $req_{IOPS}/req_{DTR}$ sont les paramètres de la QoS demandés par la VM.

Nous avons détaillé jusqu'ici chaque terme du coût de l'exécution de VM présenté dans l'équation 6.

Pour conclure, dans le cas d'un HDD, le coût d'exécution de la charge d'E/S d'une VM est calculé comme suit :

$$\begin{aligned} Cost_{exe}(VM, D) = &\left[\left[\sum_{i}^{seq,rnd}\sum_{j}^{read,wrt}\left(P_{i,j} \cdot \left(\frac{rate_{i,j}}{100} \cdot T_{op}\right)\right) + (P_{idle} \cdot T_{idle})\right.\right. \\ &\left.\left. + (P_{stdb} \cdot T_{stdb}) + (n \cdot E_{spn})\right] \cdot E_{UP}\right] + \left[\left(1 - \frac{dev_{DTR}(D)}{req_{DTR}(VM)}\right) \cdot \right. \\ &\left. Bill(VM, D)\right] + \left[\frac{stg_{UP} \cdot stg_{cap}}{N_{str-stp}} \cdot NB_{str-stp}(T_{mon})\right] \end{aligned} \tag{26}$$

Dans le cas d'un SSD, le coût est le suivant :

$$\begin{aligned} Cost_{exe}(VM, D) = &\left[\left[\sum_{i}^{seq,rnd}\sum_{j}^{read,wrt}\left(P_{i,j} \cdot \left(\frac{rate_{i,j}}{100} \cdot T_{op}\right)\right) + (P_{idl1} \cdot T_{idl1})\right.\right. \\ &\left.\left. + (P_{idl2} \cdot T_{idl2})\right] \cdot E_{UP}\right] + \left[\left(1 - \frac{dev_{DTR}(D)}{req_{DTR}(VM)}\right) \cdot Bill(VM, D)\right] \\ &+ \left[\frac{stg_{UP} \cdot stg_{cap}}{MAX_{wrt}}\right] \cdot \left[(T_{rnd,wrt} \cdot rate_{IO} \cdot req_{size}) + (T_{seq,wrt} \cdot vm_{DTR}(VM))\right] \end{aligned} \tag{27}$$

Après avoir détaillé la première partie du coût total de stockage de VM, nous allons décrire la seconde partie, qui représente le coût de migration (voir figure 38)

## 5.4 COÛTS DE MIGRATION DE STOCKAGE DES VM

La migration de VM appliquée au niveau du système de stockage entre un périphérique *source* et un autre *destination*. Dans la suite du chapitre, nous notons le périphérique *source* $D_{src}$ et le périphérique *destination* $D_{dst}$.

À l'image du modèle de coût d'exécution de VM, le coût de migration est obtenu à partir du coût de cycle de vie et du coût de pénalité (voir la figure 38).

### 5.4.1 *Coût total du cycle de vie de migration de VM*

Le processus de migration du stockage d'une VM doit garantir l'exécution de deux tâches principales [205] :

1. copie des disques virtuels entre les périphériques de stockage ;
2. traitement des requêtes d'E/S exécutées pendant l'opération de la copie.

Les requêtes d'E/S exécutées pendant la copie peuvent être traitées par le périphérique de stockage *source* $D_{src}$ ou *destination* $D_{dst}$ [129].

Afin de simplifier la modélisation du coût de migration, nous considérons dans un premier temps qu'une VM n'exécute pas de requête d'E/S pendant le processus de migration. Par conséquent, une migration peut être perçue comme une dégradation des performances qui se traduit par un coût de pénalité (voir partie 5.4.2).

Dans ces conditions, le coût de cycle de vie est uniquement calculé pour la tâche de copie des disques virtuels.

#### 5.4.1.1 *Coût énergétique*

D'une manière générale, le coût de la copie d'un disque virtuel est composé de deux coûts :

1. coût de la lecture du disque virtuel à partir du périphérique *source* $D_{src}$ ;
2. coût de l'écriture du disque virtuel sur le périphérique *destination* $D_{dst}$.

Ainsi, le coût énergétique (noté $Cost_{m\_egy}$) est calculé par :

$$Cost_{m\_egy}(VM, D) = (E(VM, D_{src}) + E(VM, D_{dst})) \cdot E_{UP} \quad (28)$$

où $D_{src}$ est le périphérique de stockage dans lequel la VM est actuellement stockée, $D_{dst}$ représente le périphérique de stockage D, $E_{D_{src}}$ est l'énergie consommée par le périphérique *source*, $E_{D_{dst}}$ est l'énergie consommée par le périphérique *destination*, et $E_{UP}$ est le prix unitaire de l'énergie.

Pour chacun des deux périphériques de stockage, la quantité d'énergie consommée pendant le processus de migration est obtenue comme suit :

$$\begin{aligned} E(VM, D_{src}) &= \sum_{i}^{seq,rnd} (P_{i,read} \cdot T_{i,read}) \\ E(VM, D_{dst}) &= \sum_{i}^{seq,rnd} (P_{i,wrt} \cdot T_{i,wrt}) \end{aligned} \tag{29}$$

La prochaine partie détaille la seconde partie du coût de cycle de vie, qui vise à calculer le coût de l'usure des périphériques de stockage (noté $Cost_{m\_wo}$).

#### 5.4.1.2 *Coût de l'usure des périphériques de stockage*

Lors de la migration, la quantité de données à transférer entre les périphériques de stockage est connue à l'avance. Il s'agit de la taille du disque virtuel $IMG_{size}$ de la VM en question. Par conséquent, le temps nécessaire pour copier le disque virtuel (noté $T_{mig}$) est calculé à partir du taux minimum de transfert de données entre les deux périphériques de stockage $D_{src}$ et $D_{dst}$ :

$$T_{mig}(VM, D_{src}, D_{dst}) = \frac{IMG_{size}}{Min\big(dev_{DTR}(D_{src}), dev_{DTR}(D_{dst})\big)} \tag{30}$$

Pour un SSD, le coût de l'usure d'un périphérique de stockage lié à la migration est obtenu à partir du coût par mégaoctet (noté $Cost_{per\_mb}$ et calculé dans l'équation 18), et la taille du disque virtuel.

Pour un HDD, nous utilisons le coût par cycle *start-stop* (noté $Cost_{str-stp}$ et calculé dans l'équation 21), ainsi que le nombre de cycles *start-stop* (noté $NB_{str-stp}$) pendant la période de migration $T_{mig}$ :

$$Cost_{m\_wo}(VM, D_{src}, D_{dst}) = \begin{cases} Cost_{per\_mb}(D_{dst}) \cdot IMG_{size} \\ \quad \textbf{si } D_{dst} = SSD \\ Cost_{str-stp}(D_{dst}) \cdot NB_{str-stp}(T_{mig}(VM, D_{src}, D_{dst})) \\ \quad \textbf{si } D_{dst} = HDD \end{cases} \tag{31}$$

Les fiches techniques des HDD modernes intègrent de plus en plus la quantité maximale de données écrites et lues à partir du disque. Il est donc tout à fait possible d'utiliser le modèle d'usure des SSD pour les HDD. Nous avons opté pour utiliser les deux modèles d'usure qui reflètent les caractéristiques conceptuelles de chaque périphérique de stockage.

La prochaine partie présente le coût de pénalité liée à la migration de VM.

### 5.4.2 *Coût de pénalité de migration de VM*

Pendant la migration de l'image d'une VM, les performances de cette dernière se dégradent considérablement [129]. Cette dégradation des performances peut

conduire à une violation des termes du SLA, et par conséquent à des pénalités. Ces pénalités doivent être ajoutées aux pénalités de l'exécution de VM.

Nous avons décrit tous les termes constituant le coût de la migration de stockage d'une VM. L'équation 32 montre un exemple de coût de la migration du stockage $Cost_{mig}$ d'une VM (notée VM), depuis un HDD (noté $D_{src}$) vers un SSD (noté $D_{dst}$) :

$$\begin{aligned} Cost_{mig}(VM, D_{src}, D_{dst}) = & \left[ \sum_{i}^{seq,rnd} (P_{i,read} \cdot T_{i,read}) \right. \\ & \left. + \sum_{i}^{seq,rnd} (P_{i,wrt} \cdot T_{i,wrt}) \right] \\ & + \left[ \frac{stg_{UP} \cdot stg_{cap}}{MAX_{wrt}} \cdot IMG_{size} \right] \end{aligned} \tag{32}$$

où VM désigne la VM à migrer, $D_{src}$ représente le périphérique *source* HDD, et $D_{dst}$ est le périphérique *destination* SSD.

## 5.5 SYNTHÈSE DU MODÈLE DE COÛT

À ce stade, nous avons modélisé tous les coûts constituant le coût totale de stockage d'une VM sur un périphérique de stockage. En guise d'exemple, l'équation 33 donne le coût de stockage d'une VM sur un SSD migrée par la suite vers un HDD :

$$\begin{aligned} Cost(VM, D) = & \left[ \left[ \sum_{i}^{seq,rnd} \sum_{j}^{read,wrt} \left( P_{i,j} \cdot \left( \frac{rate_{i,j}}{100} \cdot T_{op} \right) \right) + (P_{idl1} \cdot T_{idl1}) \right. \right. \\ & \left. \left. + (P_{idl2} \cdot T_{idl2}) \right] \cdot E_{UP} \right] + \left[ \left[ \frac{stg_{UP} \cdot stg_{cap}}{MAX_{wrt}} \right] \right. \\ & \left. \cdot \left[ (T_{rnd,wrt} \cdot rate_{IO} \cdot req_{size}) + (T_{seq,wrt} \cdot vm_{DTR}(VM)) \right] \right] \\ & + \left[ \left( 1 - \frac{dev_{DTR}(D)}{req_{DTR}(VM)} \right) \cdot Bill(VM, D) \right] \\ & + \left[ \sum_{i}^{seq,rnd} (P_{i,read} \cdot T_{i,read}) + \sum_{i}^{seq,rnd} (P_{i,wrt} \cdot T_{i,wrt}) \right] \\ & + \left[ \left( \frac{stg_{UP} \cdot stg_{cap}}{N_{str-stp}} \right) \cdot NB_{str-stp}(T_{mig}(VM, D_{src}, D_{dst})) \right] + Cost_{nrc} \end{aligned} \tag{33}$$

où VM désigne la VM pour laquelle nous calculons le coût, et D représente le périphérique de stockage sur lequel la VM est/sera stockée.

Dans la suite du chapitre, nous vérifions la pertinence de notre modèle de coût. Nous présentons son évaluation qui comprend la méthode suivie, les expérimentations effectuées, ainsi qu'une analyse et une discussion des résultats obtenus lors de cette évaluation.

Il est maintenant nécessaire de vérifier la pertinence de ce modèle de coût.

## 5.6 ÉVALUATION DU MODÈLE DE COÛT ÉNERGÉTIQUE

Nous avons utilisé une plate-forme physique réelle pour évaluer les coûts de la consommation énergétique pour l'exécution et la migration de VM. L'évaluation des coûts de pénalité et d'usure des périphériques de stockage n'est pas abordée dans notre travail. Ce type d'évaluation nécessite une étude à part entière, dans un environnement de *cloud* réel ou simulé, avec des clients, un fournisseur du service *cloud*, et des contrats du SLA [170]. L'environnement à notre disposition ne nous a pas permis de traiter ce point dans notre étude.

Cette partie présente une évaluation de notre modèle coût, en utilisant des *benchmark* d'E/S. Nous allons d'abord introduire dans la partie 5.6.1 la méthode générale d'évaluation ainsi que la plateforme expérimentale. Nous décrivons par la suite (partie 5.6.2) le processus suivi pour collecter les paramètres nécessaires au calcul du coût énergétique. La partie 5.6.3 montre ensuite les scénarios et les configurations choisies pour nos expérimentations. Finalement, les parties 5.6.4 et 5.6.5 présentent respectivement les résultats d'évaluation obtenus, ainsi qu'une analyse des résultats du modèle de coût à partir des mesures réelles.

### 5.6.1 *Méthode d'évaluation*

Cette évaluation est restreinte au modèle de coût énergétique lié à l'exécution de la charge d'E/S des VM, ainsi qu'à la migration (équations 13 et 29).

Nous considérons également que les VM exécutent des requêtes d'E/S durant toute la période de *monitoring* sans interruption. Cela veut dire qu'il n'y aura pas de changement de modes au niveau des périphériques de stockage : les périphériques seront en mode opérationnel durant toute la durée du *monitoring*. Les paramètres de la puissance électrique $P_{i,j}$ et du temps $T_{i,j}$ utilisés dans le calcul des coûts énergétique sont obtenus respectivement à l'aide d'une étape de calibration et d'analyse des traces E/S des VM. Pour ce faire, nous avons procédé comme suit :

1. **Calibration des paramètres du modèle de coût** : cette étape a pour but de collecter les valeurs des paramètres de la puissance électrique (i.e. $P_{seq,read}$, $P_{rnd,read}$, $P_{seq,wrt}$, $P_{rnd,wrt}$), qui sont utilisés pour le calcul des coûts énergétiques.

2. **Analyse des traces d'E/S et migration des VM** : cette étape comporte deux d'expérimentations. Dans la première, nous exécutons un *benchmark* d'E/S en faisant varier un ensemble de paramètres (voir tableau 8). La deuxième expérimentation consiste à migrer les VM en faisant varier la taille du disque virtuelle correspondant.

   Pendant l'exécution des d'expérimentations, nous traçons les opérations d'E/S, puis nous les analysons en ligne afin d'obtenir les valeurs des paramètres de temps (i.e. $T_{seq,read}$, $T_{rnd,read}$, $T_{seq,wrt}$, $T_{rnd,wrt}$). Pour chaque expérimentation, nous mesurons l'énergie consommée afin de la comparer avec celle calculée à partir de notre modèle de coût en prenant en compte les valeurs des paramètres : $P_{seq/rnd,read/wrt}$, et $T_{seq/rnd,read/wrt}$.

où $P_{seq/rnd,read/wrt}$ et $T_{seq/rnd,read/wrt}$ désignes les différentes valeur de la puissance électrique et de temps d'exécution des opérations de lecture/écriture séquentielle/aléatoire.

#### 5.6.1.1 *Plateforme expérimentale*

Les expérimentations ont été effectuées sur une machine physique dotée d'un processeur Intel® Xeon® E5-1620 cadencé à 3.7GHz, et d'une RAM de 16Go.

Pour le système de stockage, deux partitions de taille égale ont été utilisées. La première est une partition d'un HDD Segate ST1000DM003 d'une taille de 128GB, et la deuxième est une partition d'un SSD Samsung 840 PRO d'une taille de 128GB également. Selon les fiches techniques des deux périphériques de stockage, le HDD consomme ~5.9W en mode opérationnel, ~3.36W en mode *idle*, et ~0.63W en mode *standby*. Le SSD consomme ~0.069W en mode opérationnel, et ~0.054W en mode *idle*. Dans nos expérimentions, ces valeurs ne sont pas utilisées pour valider notre modèle de coût car il n'est pas facile de reproduire le même environnement et les même outils utilisés pour obtenir ces valeurs. De plus, nous ne disposions pas de moyen pour mesurer la puissance électrique de chaque composant séparément (CPU, RAM, HDD, SSD, etc). La figure 39 montre la plate-forme expérimentale utilisée pour l'évaluation de notre modèle de coût énergétique :

Comme le montre la figure 38, nous avons utilisé trois dispositifs principaux pour effectuer les expérimentations :

1. **une machine de contrôle** à partir de laquelle nous lançons les expérimentations et dans laquelle nous effectuons les analyses des traces E/S. Nous utilisons le protocole de communication SSH pour le e lancement des scénarios d'expérimentations ;

2. **la machine d'expérimentations**, qui héberge les VM, exécute leur charge d'E/S, et effectue leur migration ;

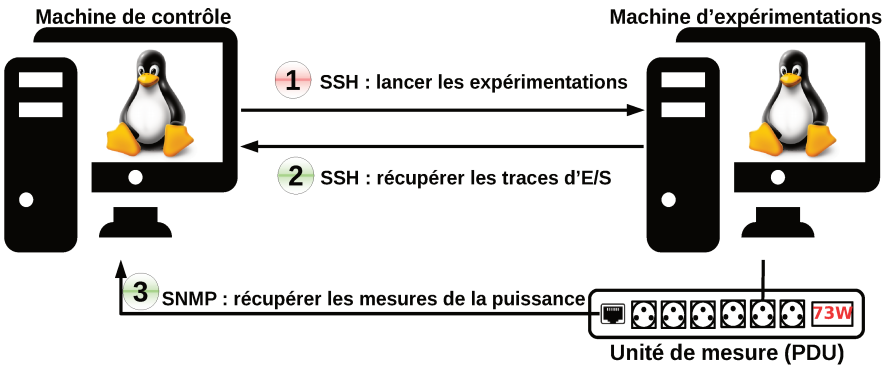


FIGURE 39 : plate-forme des expérimentations

3. **l'unité de mesure (PDU) (*Power Distributionand metering Unit*)** qui se charge de mesurer la consommation de la puissance électrique de la machine d'expérimentations pendant l'exécution des différents scénarios. Nous utilisons un PDU Raritan PX3 5190R qui permet de mesurer le voltage et le courant, avec une fréquence d'une mesure par seconde en utilisant le protocole SNMP.

La figure 40 détaille l'utilisation de la plate-forme expérimentale.

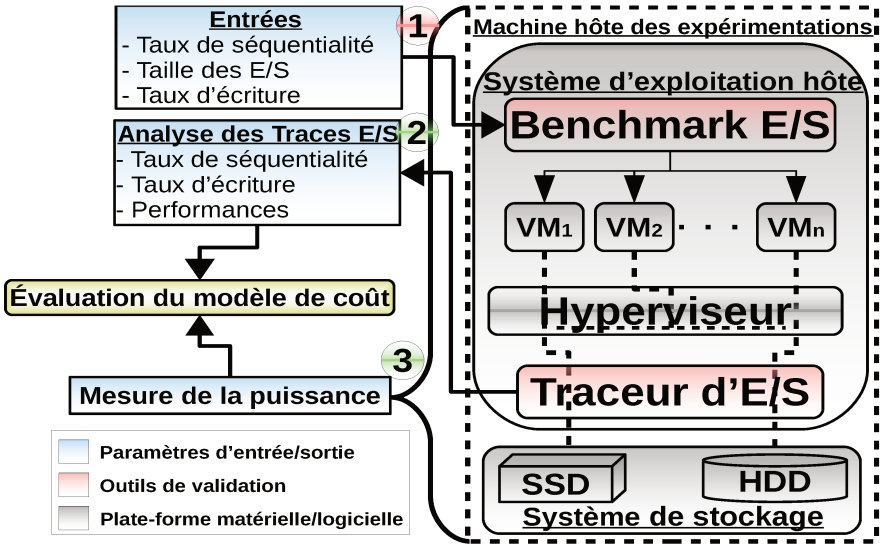


FIGURE 40 : architecture détaillée de la plate-forme expérimentale

Comme le montre la figure 40, la première étape consiste à exécuter le *benchmark* d'E/S en faisant varier : 1) le taux d'écriture, 2) le taux de séquentialité, 3) et la taille des requêtes. Après l'exécution des expérimentations, nous analysons dans un second temps les traces d'E/S des VM obtenu à l'aide de notre outil de *monitoring* (voir partie 4.3). Cette analyse utilise le modèle de caractérisation présenté dans la partie 4.4. La puissance électrique est également mesurée durant l'exécution des expérimentations. Les résultats récupérés depuis la ma-

chine expérimentale (i.e. traces d'E/S et puissance électrique) sont utilisés pour la validation de notre modèle de coût.

En terme de configuration logicielle, les machines physiques et virtuelles utilisent Debian 8 GNU/Linux comme système d'exploitation. La version du noyau utilisé est 3.2.0 et le système de fichiers utilisé est EXT4. Le *benchmark* d'E/S utilisé dans les VM est *fio* [43] (voir partie 2.4). Les VM s'exécutent sur l'hyperviseur KVM/Qemu. La version module noyau du traceur d'E/S a été utilisée afin de tracer les opérations d'E/S des VM (voir partie 4.3).

### 5.6.2 *Calibration des paramètres du modèle de coût*

Nous considérons que la puissance consommée par un périphérique de stockage est la différence entre la puissance consommée pendant l'exécution des opérations d'E/S, et celle consommée en mode *idle* du système. Cette puissance inclut la puissance consommée par le CPU durant le traitement des E/S, ainsi que celle consommée par la RAM durant les opérations de copies.

Pour pouvoir collecter les valeurs des paramètres recherchés d'un périphérique de stockage donné, nous avons développé un *micro-benchmark* effectuant des opération d'E/S élémentaires. Les paramètres ciblés sont les puissances électriques des périphériques de stockage en exécutant les différents types d'E/S : puissance des lectures séquentielles $P_{seq,read}$, puissance des lectures aléatoires $P_{rnd,read}$, puissance des écritures séquentielles $P_{seq,wrt}$, et puissance des écritures aléatoires $P_{rnd,wrt}$.

Dans le but de minimiser l'impact des différents caches du système hôte, notre *micro-benchmark* d'E/S exécute exclusivement des lectures/écritures séquentielles/aléatoires synchrones. De cette façon, nous limitons l'intervention de la RAM, et ainsi nous isolons les activités E/S du reste du système.

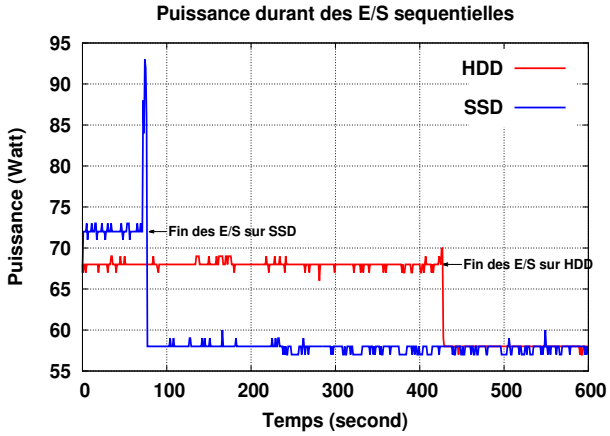


FIGURE 41 : calibration de la puissance des E/S séquentielles

La figure 41 montre la puissance totale (CPU, RAM, et périphériques de stockage) consommée durant l'exécution des opérations d'E/S séquentielles sur HDD (HDD), et SSD. À première vue, nous pouvons constater que dans le cas d'un SSD, le système consomme plus de puissance (~72W) par rapport au cas d'un HDD (~68W). Néanmoins, le SSD a fini le traitement des E/S beaucoup plus tôt (~340 secondes plus tôt) que le HDD . Comme l'énergie consommée est l'intégrale de la puissance durant le temps de l'exécution, le système consomme 4 fois plus d'énergie dans le cas du HDD par rapport au SSD. L'autre observation concerne la puissance en mode *idle* du système qui est de ~58W. Pour ce qui concerne les opérations d'E/S aléatoires, la puissance est de ~60W dans le cas d'un HDD , et ~71W dans le cas d'un SSD.

### 5.6.3 *Expérimentations de l'exécution et de la migration des VM*

Après avoir collecté les valeurs des paramètres de puissance, nous avons exécuté les expérimentations décrites précédemment afin de collecter les paramètres $T_{seq,read}$, $T_{rnd,read}$, $T_{seq,wrt}$, $T_{rnd,wrt}$ nécessaires pour la validation du modèle de coût (équations 13 et 29). Cette partie a pour but de décrire le processus expérimental suivi pour la validation des coûts énergétiques.

Les expérimentions ont été effectuées en utilisant trois VM. Toutes les VM disposent de la mème configuration, à savoir 1 vCPU et 1Go de RAM. Les VM ont différentes capacités de stockage : 20, 30, et 50Go de vDisk. En terme de *benchmark* d'E/S, toutes les VM exécutent *fio* avec les mêmes paramètres d'entrée. Les caches sur les différents niveaux (systèmes d'exploitation hôte et VM) sont vidés avant et après chaque expérimentation, afin de minimiser leur effet et de garder le même état initial. De même pour conserver un état initial cohérent sur les périphériques de stockage, les partitions dédiées aux expérimentations sont formatées avant chaque expérimentation. Chaque expérimentation a été exécutée 5 fois, et seule la moyenne de chaque paramètre de sortie est utilisée pour la validation. Le tableau 8 donne les différents scénarios de validation du modèle de coût :

| **Coût évalué** | **Paramètre** | **Entrée** | **Sortie** |
|---|---|---|---|
| **Exécution de VM** | Taux d'écriture (%) | [0-100] | $T_{i,j}$, $P_{i,j}$ |
| | Motif d'accès | $seq, rnd$ | |
| | $req_{size}$(KB) | [4-128] | |
| **Migration de VM** | $IMG_{size}$(GB) | 10,30,50 | $T_{mig}$, $P_{i,j}$ |

TABLE 8 : scénarios d'évaluation du modèle de coût

Pour évaluer le coût énergétique lié à l'exécution de VM, trois paramètres du *benchmark* d'E/S ont été variés :

1. le taux d'écriture (de 0% à 100% par pas de 20%)
2. le motif d'accès (E/S aléatoires ou séquentielles)
3. la taille des requêtes d'E/S (de 4KB à 128KB suivant une suite géométrique de raison 2)

L'évaluation du coût énergétique de migration a été effectuée en faisant varier la taille du stockage de la VM. Nous mesurons la puissance totale consommée par le système durant l'exécution des expérimentations, puis nous calculons l'énergie totale consommée pour l'exécution des E/S et la migration des VM.

### 5.6.4 *Analyses des résultats de validation*

Cette partie présente une analyse des résultats obtenus à partir des expérimentations. L'analyse traite séparément l'évaluation du modèle de coût pour HDD et l'évaluation du modèle pour SSD. Pour chaque périphérique de stockage, nous présentons une comparaison entre les résultats sur l'énergie consommée obtenue à partir du modèle de coût, et celle mesurée durant l'exécution des expérimentations.

#### 5.6.4.1 *Validation du modèle de coût pour HDD*

Dans ce scénario de validation, les VM sont initialement stockées sur le HDD (pour le coût énergétique d'exécution de VM), puis migrées vers le SSD (pour le coût énergétique de migration de VM).

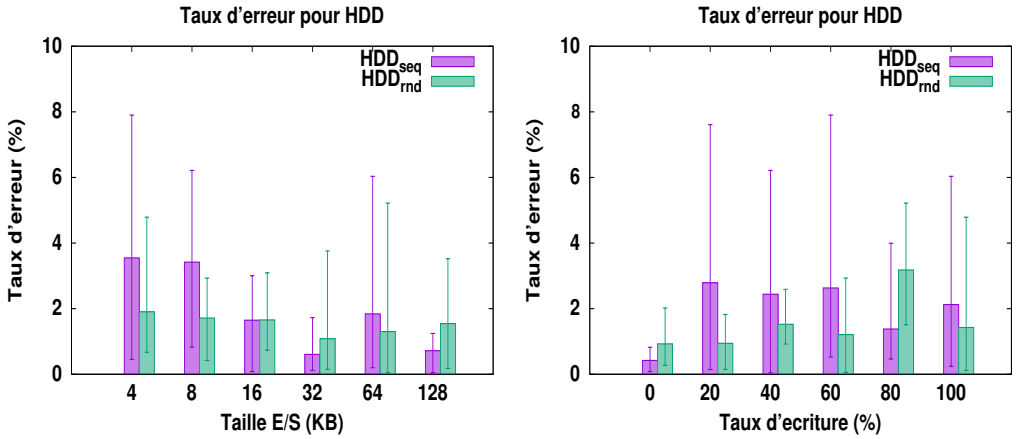


FIGURE 42 : taux d'erreur du modèle énergétique sur HDD

La figure 42 montre la moyenne (en histogramme), le minimum et le maximum (en barres d'erreur) du taux d'erreur entre les résultats obtenus à l'aide de notre

modèle et ceux obtenus avec les mesures réelles. L'axe des abscisses représente la taille des requêtes E/S ou le taux d'écriture, et l'axe des ordonnées le taux d'erreurs. Nous constatons que notre modèle présente un taux d'erreur inférieur à 8% dans le pire cas, et inférieur à 1% dans le meilleur cas. La deuxième observation est que le taux d'erreur moyen ne dépasse pas 4% dans tous les cas de figures. La dernière observation concerne l'instabilité des résultats dans le cas des accès séquentielles. Cela est principalement due à l'effet de cache en exécutant des accès séquentiels, car même si l'effet de cache est réduit par les opérations d'E/S synchrones, le mécanisme de préchargement de Linux (i.e. *read-ahead*) reste toujours activé. Cet effet induit une consommation énergétique supplémentaire du CPU et de RAM, ce qui n'est pas intégré dans notre modèle.

#### 5.6.4.2 *Validation du modèle de coût pour SSD*

Dans ce scénario de validation, les VM sont initialement stockées sur le SSD (pour le coût énergétique d'exécution de VM), puis migrées vers le HDD (pour le coût énergétique de migration de VM).

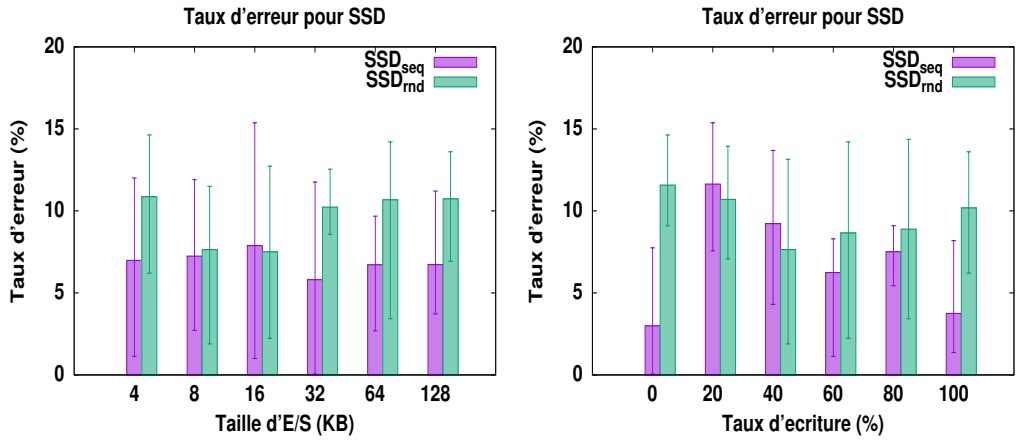


FIGURE 43 : taux d'erreur du modèle énergétique sur SSD

La figure 43 montre la moyenne, le minimum et le maximum du taux d'erreur entre notre modèle de coût énergétique, et les mesures réelles. Dans ce cas d'étude, notre modèle de coût donne des résultats moins précises que ceux obtenus dans le cas du HDD . Cependant, dans la plupart des cas le taux d'erreur moyen reste en dessous de 10%, et dans tous les cas inférieur à 15% . Cette différence de résultats entre les deux types de périphérique de stockage est principalement due à l'instabilité des activités internes du SSD (effet de cache, ramasse miettes, nivellement de l'usure, etc). Nous notons également que ces aspects ne sont pas pris en compte dans la version actuelle de notre modèle de coût.

### 5.6.5 *Discussion sur notre modèle et son évaluation*

Les expérimentations effectuées pour la validation du modèle de coût énergétique ont permis de déterminer les paramètres à prendre en compte afin d'avoir un modèle pertinent. À notre connaissance, ces paramètres n'ont pas été pris en compte dans les travaux de la littérature. De nos expérimentations, nous en déduisons :

1. la consommation énergétique liée au stockage des VM ne peut pas être ignorée, et doit être prise comte en même temps que l'énergie liée au CPU et à la RAM. L'énergie liée au stockage dépend fortement des caractéristiques de la charge d'E/S des VM. Nous avons constaté une chute de 50% de la consommation énergétique quand la taille des requêtes d'E/S est doublée. La calibration nous a révélé qu'il n'y a pas de différence significative entre la puissance consommée en exécutant des opérations de lecture et des opérations d'écriture. Les différences principales apparaissent au niveau des performances (i.e. des temps) en fonction des opérations de lectures et d'écritures, et au niveau de la puissance des opérations séquentielles/aléatoires.

2. la modélisation détaillée et réaliste des périphériques de stockage, en passant par une étape de calibration nous a conduit à des résultats inférieurs à 10% dans notre modèle de coût, par rapport aux mesures réelles.

3. les coûts liés à l'environnement du *cloud* et à la migration de VM doivent être considérés, car ils peuvent avoir un impact sur la qualité du service et sur la consommation énergétique [97].

L'étape de validation de notre modèle de coût a prouvé que les coûts de la migration de VM sont correctement modélisés, car le taux moyen d'erreur reste en dessous de 10%.

De plus, même si les coûts de pénalité n'ont pas été validés, notre modèle de coût prend en compte une méthode de calcul de pénalité proportionnelle fréquemment utilisée qui nous semble la plus équitable parmi les méthode.

Nous avons proposé un modèle de coût précis pour les systèmes de stockages, qui peut être facilement étendu et appliqué à d'autres composants. Ce modèle peut également être utilisé dans l'optimisation de stockage à l'aide des algorithmes de placement de données, des approches de consolidation, ou même dans perspectives de tarification du service *cloud*.

## 5.7 CONCLUSION

Dans l'optique de maximiser leurs revenues et de proposer des prix compétitifs pour les clients, les fournisseurs des services *cloud* cherchent à minimiser le coût total de leurs infrastructures.

Pour atteindre cet objectif, les fournisseurs ont besoin de modéliser avec précision les coûts de leur infrastructure virtualisée. Malheureusement, les modèles de coût proposés dans l'état de l'art ne prennent pas en compte simultanément les contraintes liées à l'environnement *cloud* (i.e. SLA et pénalité, migration de VM), et aux périphériques de stockage (i.e. consommation énergétique, usure du périphérique).

Nous avons proposé un modèle de coût pour le stockage des VM dans le contexte du *cloud*. Le modèle prend en considération deux types de périphérique de stockage (HDD et SSD) avec différentes contraintes liées à ces périphériques (performance, énergie, et usure du périphérique). Il considère également les contraintes liées à l'environnement du *cloud* (SLA et migration des VM).

Nous avons validé ce modèle à l'aide d'une plate-forme expérimentale. Pour un périphérique de stockage de type HDD, notre modèle conduit à un taux d'erreur 8% dans le pire cas, et 0.04% dans le meilleur cas. Pour un SSD, notre modèle montre un taux d'erreur de 2% dans le meilleur cas, et 5% dans le pire cas.

La suite de ce manuscrit est organisée comme suit. La prochaine étape consiste à montrer l'intégration de notre modèle de coût dans un processus d'optimisation de stockage de VM. Nous allons détailler comment ce modèle a été intégré dans l'outil de simulation de *cloud CloudSim* [154], pour être utilisé par la suite dans un le placement de VM. Ensuite, nous présentons nos approches de placement utilisant le modèle de coût et son intégration dans le but d'optimiser les coûts d'infrastructure. La dernière partie conclut ce document.

# Chapitre 6

# ÉTAPE *PLAN* : INTÉGRATION ET SIMULATION DES ENTRÉES/SORTIES ET DES SYSTÈMES DE STOCKAGE HYBRIDES DANS LE CLOUD

Dans le modèle MAPE-K, la deuxième étape, *Analyze*, est immédiatement suivie de l'étape *Plan*. Dans notre étude, cette dernière se charge de mettre en œuvre le modèle de coût dans le but de construire des approches d'optimisation de placement de VM prenant en compte les coûts des E/S et des systèmes de stockage. L'étape *Plan* comporte deux parties. La première partie représente l'intégration de notre modèle de coût dans un environnement permettant d'expérimenter des approches d'optimisation de placement de VM. La deuxième partie comporte les approches d'optimisation utilisant cette intégration. Ce chapitre présente nos contributions pour la première partie de l'étape PLAN.

Nous avons mis en œuvre notre modèle de coût sous forme d'une extension du simulateur de *cloud CloudSim* [40]. En effet, ce simulateur, bien que largement utilisé pour l'implantation et l'expérimentation des approches d'optimisation de placement de VM, s'appuie sur une modélisation partielle du stockage et des E/S disque :

- l'exécution des charges d'E/S des VM n'est pas considérée. Par conséquent, le temps, la charge CPU, et la consommation énergétique engendrés par ce processus ne sont pas intégrés ;
- *CloudSim* ne prend pas en compte les systèmes de stockage hybrides utilisées pour stocker les données des VM.

Ce chapitre présente nos extensions de *CloudSim* afin de prendre en compte :

- l'exécution des charges d'E/S des VM et ce qui en résulte en termes de temps d'exécution, d'utilisation CPU, et de consommation énergétique ;
- le support des systèmes de stockages hybrides.

L'évaluation de notre extension a montré que l'exécution des E/S peut impacter considérablement la consommation énergétique d'un centre de données. Nos tests sur des traces réelles présentent une consommation d'énergie liée à l'exécution des E/S et aux supports de stockage avoisinant les 25%.

Ce chapitre est organisé comme suit. Dans un premier temps, nous définissons des concepts fondamentaux utilisés tout au long du chapitre. La deuxième partie décrit la gestion native du stockage dans *CloudSim*. Nous détaillons nos contributions sur le traitement des E/S et le support du stockage hybride dans la troisième et la quatrième parties. L'implantation et l'évaluation de notre extension sur *CloudSim* sont présentés dans le cinquième et la sixième parties. La septième partie conclut ce chapitre.

## Sommaire

## 6.1 DÉFINITIONS

Avant de rentrer dans les détails de notre extension sur *CloudSim*, nous définissons d'abords les concepts fondamentaux.

**Définition 6.1.** *(**Cloudlet**) [40] : dans CloudSim, une Cloudlet modélise une application ou un service exécuté par une VM. Une Cloudlet est modélisée par ses besoins en terme de capacité de calcul exprimée en MI (Millions d'Instructions).*

**Définition 6.2.** *(**Régression linéaire multiple**) [37] : une méthode de régression linaire permet de prédire les valeurs d'une variable $y$ en fonction de $k$ paramètres $x_1, x_2, \ldots, x_k$ en utilisant un modèle linéaire comme suit :*

$$y = b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + \cdots + b_k \cdot x_k + e$$

*où $\{b_0, b_1, \ldots, b_k\}$ sont $k+1$ des constantes, et $e$ est un terme représentant l'erreur du modèle.*

## 6.2 CLOUDSIM ET LA GESTION DU STOCKAGE

Avant de décrire notre mise en œuvre sur *CloudSim*, nous introduisons quelques principes élémentaires de sa gestion du stockage.

### 6.2.1 *CloudSim : principes de base*

L'architecture de *CloudSim* est composée de quatre entités : 1) `Datacenter`, 2) `DatacenterBroker`, 3) `CloudInformationService`, et 4) `CloudsimShutdown`. L'entité `Datacenter` modélise un centre de données composé d'un ensemble de serveurs (i.e. PM). L'entité `DatacenterBroker` modélise le comportement de l'administrateur du *cloud*. Son rôle est de servir d'intermédiaire entre le fournisseur des services *cloud* et les clients (i.e. recevoir et soumettre leurs demandes). `CloudInformationService` est l'entité modélisant le fournisseur du service *cloud* IaaS. L'entité `CloudsimShutdown` se charge de vérifier la terminaison de l'exécution de toutes les VM présentes dans le centre de données, puis d'éteindre les PM et de terminer leur simulation.

Les entités de *CloudSim* communiquent entre elles via des messages. Ces messages sont générés en réponse à un ensemble d'évènements prédéfinis. Les évènements dans *CloudSim* peuvent être divisés en deux catégories :

- ***les évènements externes*** : s'effectuent entre les différentes entités ;
- ***les évènements internes*** : s'effectuent au sein d'une même entité.

La génération et l'envoi d'un message par une instance d'une entité a lieu suite au déclenchement d'un évènement. L'entité réceptrice doit disposer d'une fonction associée à l'évènement que nous appelons *handler*. Cette fonction exécute un ensemble d'opérations en réponse à l'évènement associé, avant de renvoyer un accusé de réception à l'entité expéditrice (e.g. `VM_CREATE_ACK`, `VM_MIGRATE_ACK`, `VM_DESTROY_ACK`, etc).

D'un point de vu conceptuel, *CloudSim* modélise un centre de données sous la forme d'un ensemble de PM hébergeant des VM, qui à leur tour exécutent des tâches appelées *Cloudlet*. Un scénario de simulation *CloudSim* est basé sur l'exécution des *Cloudlet* présentes dans le centre de données.

*CloudSim* permet la simulation des scénarios d'utilisation d'un *cloud* IaaS, avec différentes architectures d'infrastructure matérielle. *CloudSim* est couramment utilisé dans l'objectif d'expérimenter des politiques d'optimisation de placement de VM. Ces politiques ont différents buts, notamment la consolidation pour la minimisation de la consommation énergétique d'un centre de données, ou l'équilibrage de charges entres PM.

### 6.2.2 *Motivations et contributions*

Du point de vue des E/S, *CloudSim* présente deux problèmes majeurs, notamment : 1) dans le traitement des E/S et 2) au niveau du système de stockage.

- **traitement des E/S** : *CloudSim* ne considère que le temps d'enregistrer le fichier modélisant les données d'entrée d'une *Cloudlet* lors de sa première mise en exécution sur le centre de données. Ce temps est calculé en fonction des performances des HDD (i.e. latence et débit) et du débit de transfert du réseau LAN. La consommation énergétique et l'utilisation CPU liées à cette opération ne sont pas non plus prises en compte. Ainsi, le stockage de l'image d'une VM et l'exécution de sa charge d'E/S ne sont pas considérés.

- **système de stockage** : la version actuelle de *CloudSim* (i.e. version 4.0) utilise une seule classe de stockage (i.e. HDD) sous forme d'un stockage partagé en SAN. Cette architecture représente un SAN comme un ensemble de HDD du même type [69], interconnectés par un réseau LAN. Cette architecture peut conduire à un goulet d'étranglement, tandis que l'utilisation exclusive des HDD restreint l'utilisation de *CloudSim*. Cela ne reflète pas la tendance actuelle des centre de données [142].

L'objectif principal de notre extension est de prendre en compte l'exécution des charges d'E/S des VM, dans le contexte d'un *cloud* IaaS. L'exécution des charges d'E/S est composée des deux éléments suivants :

1. l'exécution des charges d'E/S sur le système de stockage, et tout ce qu'il en résulte en termes de temps de traitement, de consommation énergétique, et d'utilisation CPU ;

2. le support des systèmes de stockage hybride comprenant divers classes de périphériques de stockage utilisé en local (i.e. en DAS) ou partagé (i.e. en SAN).

Nos contributions à *CloudSim* peuvent être résumés en deux points :

1. ***modélisation du temps de traitement, de l'énergie, et d'utilisation CPU pour les E/S*** : nous avons d'abord étendu les modèles existants de calcul de temps et de consommation énergétique afin de prendre en compte l'exécution des E/S liées aux VM. Notre extension définit les charges d'E/S au niveau des VM. Ce choix est motivé par notre objectif d'optimisation de placement des VM (voir partie 4.4). D'autre part, le niveau VM est le plus utilisé dans les plates-formes réelles de virtualisation [11]. Nous nous sommes basés sur des mesures réelles de performances des E/S pour établir un modèle de corrélation empirique permettant de prédire l'utilisation CPU en fonction des E/S synchrones. Le modèle obtenu dépend principalement des caractéristiques des charges d'E/S et celles des périphériques de stockage.

2. ***mise en œuvre du support de stockage hybride*** : notre contribution intervient également au niveau de la modélisation des périphériques de stockage. Nous avons étendu le modèle du système de stockage utilisé dans *CloudSim*. Notre extension vise les performances et la puissance électrique des périphériques de stockage. Notre contribution permet également d'utiliser un stockage local (i.e. DAS) ou partagé (i.e SAN) .

La prochaine partie présente en détail chacun des points abordés ci-dessus. Nous y exposons également la méthode utilisée pour leur évaluation, ainsi que les résultats obtenus.

## 6.3 MODÉLISATION DU TEMPS DE TRAITEMENT, D'ÉNERGIE, ET D'UTILISATION CPU POUR LES E/S

Dans cette partie, nous présentons les trois points constituant notre première contribution sur *CloudSim*. Premièrement, nous détaillons notre modèle du calcul de temps de traitement des E/S. Nous décrivons dans un second temps le modèle de la consommation énergétique liée au traitement des E/S. La troisième partie concerne notre modèle de corrélation entre la charge d'E/S des VM et le CPU.

### 6.3.1 *Modélisation du temps de traitement des E/S*

Le temps dans *CloudSim* est modélisé d'évènements successifs. Parmi les évènements affectant le temps logique dans *CloudSim*, on distingue les évènements de traitement des *Cloudlet* et des VM (e.g. création, migration, pause, etc). La figure 44 montre un scénario simple d'exécution d'une *Cloudlet* par une VM et la composition du temps d'exécution avant notre extension.

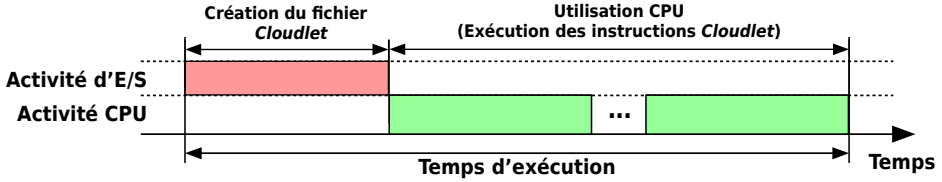


FIGURE 44 : composition du temps d'exécution avant notre extension

Comme illustré dans la figure 44, le temps d'exécution est composé de deux parties. La première est la création du fichier modélisant les données d'entrée du *Cloudlet* sur le périphérique de stockage. Cette opération est le résultat du déclenchement de l'évènement `CLOUDLET_SUBMIT`. Cet évènement se produit une seule fois pour chaque *Cloudlet*, au tout début de la simulation. Sa durée dépend de la taille du fichier *Cloudlet* et des performances du périphérique de stockage.

La deuxième partie est l'exécution des instructions de la *Cloudlet*. Son temps simulé dépend de la charge CPU de la *Cloudlet* (exprimée en nombre d'instructions), ainsi que des performances CPU de la VM et de la PM.

Comme nous l'avons annoncé précédemment, notre extension modifie le temps total simulé. La figure 45 présente la séquence d'évènements affectant le temps simulé après l'intégration de notre extension.

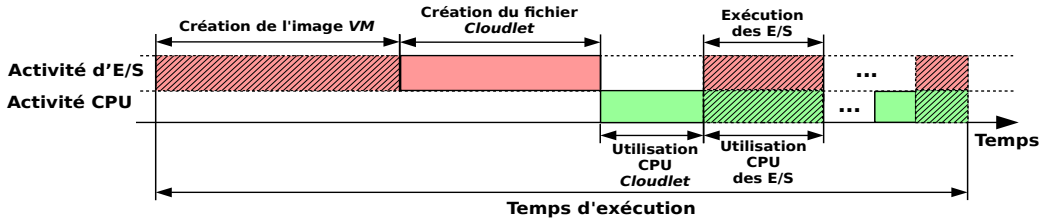


FIGURE 45 : composition du temps d'exécution après notre extension

Notre modélisations du temps d'exécution des E/S dans *CloudSim* se résume en deux points :

- ***la création de l'image de la VM*** : la création d'une VM est un évènement interne du centre de données. Nous avons étendu le temps de création d'une VM en ajoutant le temps de création de son image sur le système de stockage. Ce temps varie en fonction de la taille de l'image de la VM et des performances du périphérique de stockage sur lequel l'image est stockée.

- ***l'exécution des E/S*** : nous avons introduit le concept de temps d'exécution des E/S dans le but de simuler l'exécution des charges d'E/S. Cela permet également de quantifier la consommation énergétique correspondante. Le temps d'exécution des E/S dépend des caractéristiques de la charge d'E/S des VM et des performances des périphériques de stockage. Afin de caractériser la charge d'E/S d'une VM, nous avons utilisé les mêmes paramètres que ceux présentés dans la partie 4.4.

L'intégration du calcul de temps des E/S nous permet de calculer la consommation énergétique liée aux E/S. La prochaine partie présente notre contribution à la quantification de la consommation énergétique des systèmes de stockage et des E/S.

### 6.3.2 *Modélisation de la consommation énergétique du traitement des E/S*

Le temps d'exécution n'est pas le seul paramètre affecté par l'exécution des charges d'E/S ; La consommation énergétique l'est aussi. Dans les versions précédentes de *CloudSim*, la consommation énergétique dépend exclusivement de la charge CPU des PM. La formule 34 donne le modèle de calcul de la puissance électrique $P(u)$ d'une PM, en fonction de sa charge CPU $u$ [26] :

$$P(u) = P_{ratio} \cdot P_{max} + (1 - P_{ratio}) \cdot P_{max} \cdot u \tag{34}$$

À titre d'exemple, les auteurs de [26] considèrent qu'une PM consomme au repos 70% de la puissance maximale, ce qui donne $P_{ratio} = 0.7$.

Dans l'équation 34, le paramètre d'entrée $u$ de la fonction de puissance $P$ représente l'utilisation CPU de la PM. Cette utilisation dépend de la charge CPU des VM et des performances CPU de la PM. La charge CPU d'une VM est la somme de toutes les charges CPU de ses *Cloudlet*. La charge CPU d'une *Cloudlet* est exprimée en MI (Millions d'Instructions). *CloudSim* utilise les MIPS (Millions d'Instructions Par Seconde) comme seule métrique de mesure de performance CPU. À titre d'illustration, la figure 46 montre la puissance électrique consommée par une PM sans prendre en compte le système de stockage.

La figure 46 est obtenue en utilisant le modèle de puissance initialement utilisé par *CloudSim* basé exclusivement sur l'utilisation CPU pour le calcul de la puissance de la PM (i.e. CPU et RAM). Les valeurs de $P_{max}$ et $P_{ratio}$ sont issues des expérimentations présentées dans [26]. La figure 46 présentent trois phases. La première et la dernière phase correspondent à la puissance en mode repos avec 0% de charge CPU. Durant la deuxième phase, la PM exécute les instructions des VM qui est représenté par une activité CPU dans la figure 46.

Nous avons complété le modèle de puissance initialement utilisé par *CloudSim*, en considérant la puissance relative à l'exécution des charges d'E/S des VM. La figure 47 illustre le résultat de notre modèle de puissance utilisé par *CloudSim*.

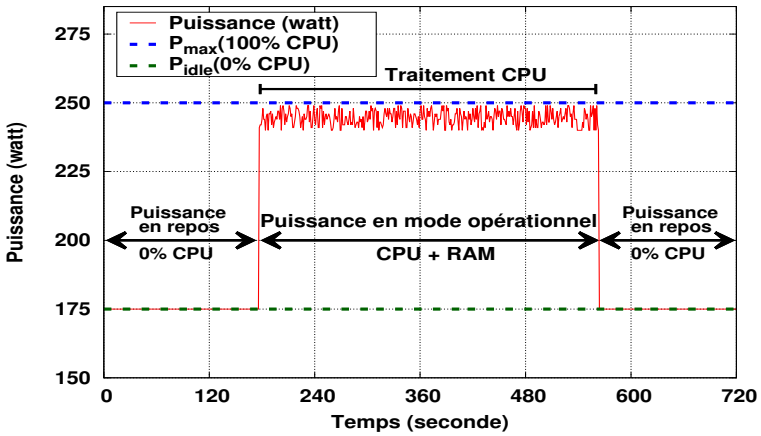


FIGURE 46 : modèle de puissance de *CloudSim* avant notre extension

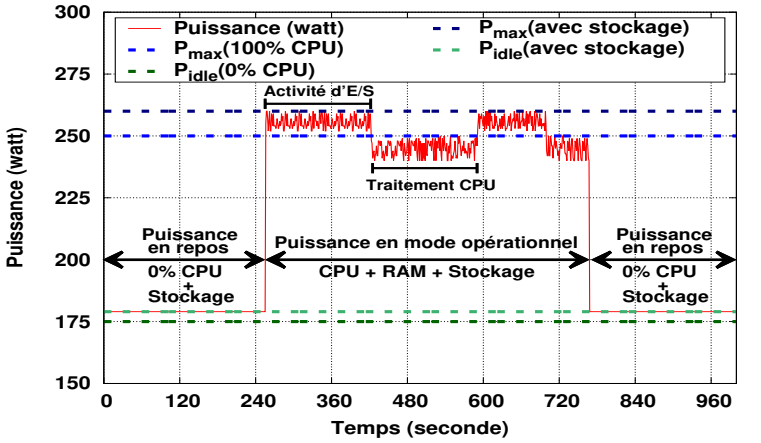


FIGURE 47 : modèle de puissance de *CloudSim* après notre extension

Un périphérique de stockage consomme de l'énérgie en mode repos (e.g. jusqu'à 60% de la puissance en mode opérationnel [171]) et par conséquent la puissance électrique de la PM en mode repos augmente également. Durant la deuxième période, la PM alterne entre l'exécution des charges d'E/S (représenté par l'activité d'E/S dans la figure 47), et l'activité CPU des VM. On constate également que le système de stockage et l'exécution des charges d'E/S ont un impact sur la consommation énergétique, en terme de puissance électrique et de temps d'exécution.

### 6.3.3 *Modélisation de la charge CPU du traitement des E/S*

La prise en compte de l'exécution des charges d'E/S dans les simulations n'implique pas seulement de considérer les activités du système de stockage, mais nécessite aussi de comptabiliser la charge CPU due à l'exécution de la pile logicielle des E/S.

Dans le but de quantifier l'implication du CPU lors du traitement des E/S. Nous avons établi empiriquement un modèle de corrélation pour la charge CPU du traitement des E/S. Avant de renter dans les détails de cette modélisation, nous délimitons d'abord le cadre de ce travail.

#### 6.3.3.1 *Cadre de travail*

Ce travail a pour objectif principal d'établir un modèle qui de prédit la charge CPU pour une plateforme donnée. Ce modèle est constitué de facteurs variables et fixes. La facteurs fixes sont :

- ***paramètres matériels*** : le CPU, la RAM, les modèles des périphériques de stockage ;
- ***paramètres logiciels*** : la version du noyau Linux, la version et configuration du *benchmark*.

Pour les facteurs que nous avons fait varier, nous avons :

- ***la charge d'E/S*** : les scénarios exécutés par le *benchmark* ;
- ***le système de stockage*** : le type de périphérique de stockage.

Ces paramètres sont présentés en détail par la suite (voir partie 6.3.3.2). La figure 48 présente les étapes suivies pour la modélisation de la charge CPU.

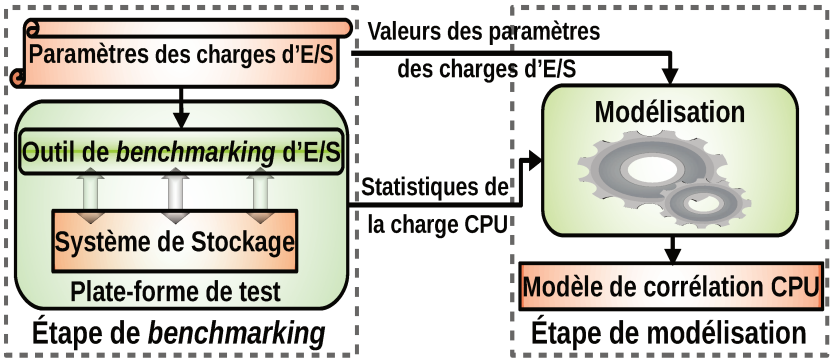


FIGURE 48 : étapes de la modélisation de la charge CPU des traitements des E/S

Comme le montre la figure 48, nous avons proposé notre modèle en suivant deux étapes :

- une **étape de *benchmarking*** ;
- puis une **étape de modélisation**.

Nous allons détailler chacune de ces étapes dans les parties suivantes.

#### 6.3.3.2 *Étape de benchmarking*

Comme le montre la figure 48, l'étape de *benchmarking* consiste à exécuter différents scénarios d'un *benchmark* d'E/S. Chaque scénario dépend de la combinaison des paramètres d'entrée du *benchmark*. Cette étape a pour objectif de mesurer la charge CPU du système en fonction des paramètres variés. Ces paramètres sont liés 1) à la charge d'E/S, et 2) au périphérique de stockage.

***1) Paramètres des charges d'E/S*** : nous avons configuré le *benchmark* pour exécuter des E/S synchrones. Cette configuration nous permet d'une part, éviter la modélisation complexe du comportement des différents caches systèmes. D'autre part, cela nous permet d'isoler l'implication du CPU dans l'exécution des E/S. La charge d'E/S d'une VM est modélisée par les paramètres suivants (voir partie 4.4) :

- le motif d'accès exprimé en taux de séquentialité/aléatoire $\mathtt{rate}_{\mathtt{rnd}}$ ;
- le type de requêtes d'E/S exprimé en taux de lecture/écriture $\mathtt{rate}_{\mathtt{wrt}}$ ;
- la taille des requêtes d'E/S $\mathtt{req}_{\mathtt{size}}$ ;
- le taux d'arrivée des requêtes d'E/S $\mathtt{rate}_{\mathtt{IO}}$ ;
- le volume total des données traitées $\mathtt{data}_{\mathtt{amount}}$.

Nous avons fait varier les quatre premiers paramètres, afin d'observer le changement de la charge CPU et identifier les paramètres les plus importants. Le dernier paramètre (i.e. le volume total des données traitées) est une constance, car nous avons constaté qu'il n'a pas d'impact sur la charge CPU, mais seulement sur le temps d'exécution du *benchmark*. Les intervalles de valeurs des paramètres sont présentés dans la partie 6.5.1.

***2) Paramètres du système de stockage*** : comme nous l'avons mentionné auparavant, notre contribution à *CloudSim* consiste en l'ajout du support de deux classes de périphérique de stockage : 1) HDD, 2) SSD. Ainsi, un paramètre permet de faire varier le type de périphérique de stockage (i.e. HDD ou SSD). Lors de l'évaluation, nous prenons deux classes de stockage avec différentes caractéristiques de performance afin de montrer la variation de la charge CPU en fonction du type de périphérique de stockage.

#### 6.3.3.3 *Étape de modélisation*

Comme le montre la figure 48, l'étape de modélisation utilise les paramètres des charges d'E/S et les statistiques de la charge CPU obtenus de l'étape de *benchmarking*, afin d'établir le modèle de corrélation.

Notre modèle de corrélation CPU peut être formalisée comme suit :

$$cpu_{load}(D, W)$$

où $cpu_{load}$ représente la fonction du modèle de corrélation, D désigne un périphérique de stockage, et $W$ est la charge d'E/S. La fonction $cpu_{load}$ calcule la charge CPU engendrée par l'exécution de la charge d'E/S.

Nous rappelons que notre modèle est conçu pour deux classes de périphérique de stockage qui sont les HDD et les SSD :

$$D \in \{HDD, SSD\}$$

Un deuxième paramètre d'entrée du modèle représente la charge d'E/S d'une VM. Comme cela a été mentionné précédemment, nous utilisons la caractérisation des charges d'E/S présentée dans la partie 4.4) :

$$W \text{ définit par } < rate_{rnd}, rate_{wrt}, rate_{IO}, req_{size} >$$

Cette multitude de variables nous a poussé à utiliser deux approches d'apprentissage automatique (*Machine Learning*) pour la conception d'un modèle de corrélation qui permet de prédire la charge CPU.

Les deux approches proposées sont : 1) la méthode *Ordinary Least Squares (OLS)* (ou méthode de *moindres carrés*), et 2) la méthode *Multivariate Adaptive Regression Splines (MARS)*. Ces deux méthodes sont des approches de régression linéaire.

Le choix de ces méthodes est basé sur les motivations suivantes :

- OLS est un modèle simple et fréquemment utilisé. C'est le modèle de regression utilisé par défaut par de nombreux outils d'analyse et de traitement de données (e.g. Matlab Ref, R ref, `scikit-learn` [6], MS Excel [3], Libreoffice Calc [1]) ;
- contrairement à OLS, le modèle MARS permet de sélectionner les paramètres les plus importants, et modélise la charge CPU avec ces paramètres.

Nous allons présenter brièvement chacune des méthodes utilisées dans notre étude.

***1) Modélisation avec la méthode OLS*** [61] : OLS est une méthode de régression utilisée pour la régression multivariée. Pour une expérimentation donnée, OLS cherche à identifier une relation entre les paramètres d'entrée et les résultats observés, dans le but de produire un modèle de prédiction sur les paramètres d'entrée. Plus précisément, cette méthode cherche à minimiser le taux d'erreur entre les résultats réellement observés, et ceux obtenus à partir du modèle de prédiction produit.

Dans notre cas, pour un périphérique de stockage donné, la charge CPU est modélisée en se basant sur les paramètres de la charge d'E/S (voir partie 6.1) :

$$cpu_{load}(D, W) = b_0 + b_1 \cdot rate_{rnd} + b_2 \cdot rate_{wrt} + b_3 \cdot rate_{IO} + b_4 \cdot req_{size} + e \tag{35}$$

où $rate_{rnd}$ est le taux des requêtes E/S aléatoires, $rate_{wrt}$ est le taux d'écriture, $rate_{IO}$ est le taux d'arrivée des requêtes E/S, $req_{size}$ est la taille des requêtes d'E/S, et $e$ représente l'erreur produite par le modèle.

À partir de l'équation 35, le modèle de régression OLS doit trouver les valeurs des paramètres $b_0$, $b_1$, $b_2$, $b_3$, et $b_4$, tout en minimisant le taux d'erreur (i.e. $e$) entre la charge CPU calculée, et celle réellement observée.

***2) Modélisation avec la méthode MARS*** : ce modèle a été choisi afin de faire face à la multitude des paramètres d'entrée de notre modèle [64].

Les modèles MARS sont essentiellement basés sur les fonctions *charnières* (*hinge loss function*) [167]. Elles représentent un type de fonctions fréquemment utilisées dans les algorithmes de *Machine Learning*. L'équation 36 présente une définition formelle d'une fonction charnière :

$$h(x, t) = [x - t]_+ = \begin{cases} x - t & \text{if } x > t \\ 0 & \text{if } x \leqslant t \end{cases} \tag{36}$$

où $h()$ est la fonction *charnière*, $x$ est un paramètre d'entrée du modèle, et $t$ est une constante définie par l'algorithme MARS.

Cet algorithme établit le modèle suivant deux étapes essentielles : 1) passe en avant (*forward pass*), 2) puis passe en arrière ou passe d'élagage (*backward/pruning pass*). Durant la passe en avant (i.e. *forward pass*), MARS cherche à trouver la valeur du premier coefficient où toutes les variables valent zéro (i.e. $b_0$ dans l'équation 35). Avec ce premier coefficient, la passe en avant ajoute des fonctions charnières pour les paramètres d'entrée, tout en gardant un taux d'erreur minimal entre le modèle et les valeurs réelles (obtenues à partir des expérimentations). Dans notre cas, un exemple simple du résultat de la passe en avant peut être donné comme suit :

$$\begin{aligned} cpu_{load}(D, W) = & b_0 + b_1 \cdot h(rate_{rnd} - t_1) + b_2 \cdot h(rate_{wrt} - t_2) \\ & + b_3 \cdot h(rate_{IO} - t_3) + b_4 \cdot (req_{size} - t_4) + e \end{aligned} \tag{37}$$

où $rate_{rnd}$, $rate_{wrt}$, $rate_{IO}$, et $req_{size}$ représentent les paramètres de la charge d'E/S, tandis que $t_1$, $t_2$, $t_3$, et $t_4$ sont des constantes définies par l'algorithme MARS.

La passe d'élagage prend la sortie de la passe en avant, et essaye de le raffiner en éliminant les paramètres les moins importants en utilisant la méthode GCV (*Generalized Cross Validation*) [73]. Le résultat de la passe d'élagage peut donner

lieu à une combinaison de fonctions charnières pour les paramètres d'entrée retenus. Certains paramètres peuvent ne pas avoir de fonction charnière.

La prochaine partie détaille l'implantation des différentes contributions présentées précédemment.

## 6.4 SUPPORT DES SYSTÈMES DE STOCKAGE HYBRIDES : IMPLANTATION DANS *cloudsim*

Nous avons étendu le système de stockage de *CloudSim* en modélisant deux types de périphériques de stockage : HDD et SSD. Du point de vue implantation, nous avons ajouté plusieurs classes afin de supporter cette extension (voir figure 49).

Les méthodes et les attributs ajoutés sont classés en deux catégories : ds attributs et des méthodes pour 1) les performances, et pour 2) la puissance électrique.

Pour chaque périphérique de stockage, la puissance électrique dépend des différents états (i.e. repos, opérationnel, et prêt), tandis que les performances dépendent de deux facteurs principaux :

1. les caractéristiques des charges d'E/S (voir partie 4.4) ;
2. les caractéristiques des périphériques de stockage (e.g. latence, temps moyen de recherche, IOPS, etc).

Concernant la consommation énergétique des périphériques de stockage, nous avons implanté le modèle de consommation énergétique que nous avons présenté dans la partie 5.3.

La figure 49 montre le diagramme de classes UML de notre extension de CloudSim. Ce diagramme présente les principales classes ajoutées complétant le modèle du système de stockage et d'exécution des E/S. Nous avons ajouté toutes les classes et interfaces en couleur foncée.

Comme nous l'avons décrit dans la partie 6.3, le traitement des E/S modifie le temps total d'exécution. Nous avons implanté ce délai par un nouveau évènement interne dans le centre de données.

L'envoi, la manipulation, et la réception des messages déclenchés par cet évènement sont implantés par la classe `IoDatacenter`. Le temps de traitement des E/S dépend essentiellement des périphériques (implantés par les `HardDriveStorage` et `SolidSateStorage`) stockant les fichiers des VM (implanté par la classe `IoVm`) et de la charge d'E/S des VM (implanté par la classe `IoWorkloadModel`).

La consommation énergétique due à l'exécution des charges d'E/S est calculée en utilisant notre modèle de coût (voir partie 5.3.1.1). Ce modèle est implanté par les classes `StorageEnergyModel`, `StorageWearOutModel` et `StorageSLAModel`.

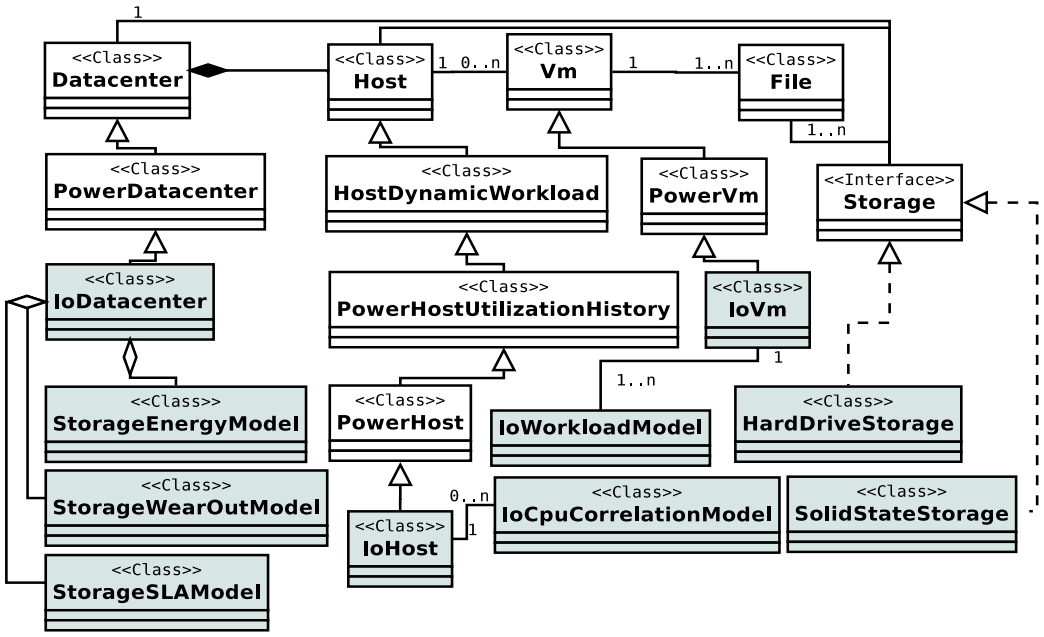


FIGURE 49 : diagramme de classes de notre extension de *CloudSim*

L'énergie liée à la charge CPU engendrée par le traitement des E/S est calculée par le modèle nativement implanté dans *CloudSim*.

Notre extension de *CloudSim* ne vise pas seulement le calcul de temps et de la consommation énergétique, mais aussi la prise en compte de la charge CPU générée par le processus d'exécution des charges d'E/S des VM. Le modèle de corrélation CPU que nous avons présenté dans la partie 6.3.3 a été implanté par la classe `IoCpuCorrelationModel`. Ce modèle prend en compte les charges d'E/S des VM, les performances CPU et les périphériques de stockage utilisés par chaque PM. Comme nous l'avons précisé dans la partie 6.3.3.1, ce modèle est donc lié aux caractéristiques des PM exécutant les charges d'E/S des VM (implanté par la classe `IoHost`).

Après avoir détaillé la modélisation et l'implantation de notre extension de *CloudSim*, nous allons maintenant présenter la méthode d'évaluation de cette extension, ainsi que les résultats obtenus.

## 6.5 ÉVALUATION DU MODÈLE DE CORRÉLATION CPU ET DE NOTRE EXTENSION DE *cloudsim*

Cette évaluation est divisée en deux parties. La première partie décrit la méthode et les résultats d'évaluation du modèle de corrélation CPU. La deuxième partie valide notre extension de *CloudSim* en utilisant différents scénarios de simulation.

### 6.5.1 *Évaluation du modèle de corrélation CPU*

L'évaluation de notre modèle de corrélation comporte deux étapes. La première consiste à exécuter un *benchmark* d'E/S en faisant varier les paramètres de la charge d'E/S. L'objectif de cette étape est de mesurer la charge CPU en fonction de la charge d'E/S.

La deuxième étape vise à établir un modèle de prédiction de la charge CPU, en nous basant sur les mesures de la charge CPU et les paramètres de la charges d'E/S variés lors de la première partie. Les mesures de la charges CPU sont divisées aléatoirement en deux sous-ensembles. Le premier sous-ensemble représente 70% de l'ensemble des résultats obtenus. Ce groupe est utilisé pour la conception du modèle de corrélation CPU. Dans le domaine de l'apprentissage automatique, ces données sont appelées données d'entraînement (*training features*). Le second sous-ensemble de 30% de données restantes est utilisé pour valider le modèle obtenu. Cet ensemble est appelé données de validation (*validation features*).

Pour l'évaluation de la pertinence des modèles obtenus, nous comparons les modèles OLS et MARS en utilisant le coefficient de détermination appelé $R^2$.

#### 6.5.1.1 *Description des scénarios de benchmarking*

L'étape de *benchmarking* consiste à exécuter `fio` [43], tout en faisant varier un ensemble de paramètres d'entrée. Le tableau 9 présente les paramètres d'entrée et leur intervalle de variation.

| **Paramètre** | **Valeurs & Intervalles** |
|---|---|
| D | {HDD, SSD} |
| $\mathtt{rate}_{\mathtt{rnd}}$ | [0%,...,100%] avec un pas de 20% |
| $\mathtt{rate}_{\mathtt{wrt}}$ | [0%,...,100%] avec un pas de 20% |
| $\mathtt{rate}_{\mathtt{IO}}$ | Performance disponible du périphérique de stockage |
| $\mathtt{req}_{\mathtt{size}}$ | [2KB,...,1024KB] avec une échelle logarithmique |

TABLE 9 : valeurs et intervalles des paramètres du modèle

Le premier paramètre D est le périphérique de stockage. Les autres paramètres sont liés aux charges d'E/S : le taux d'écriture ($\mathtt{rate}_{\mathtt{wrt}}$), le taux de requêtes aléatoires ($\mathtt{rate}_{\mathtt{rnd}}$), le taux d'arrivé des requêtes ($\mathtt{rate}_{\mathtt{IO}}$), et la tailles des requêtes ($\mathtt{req}_{\mathtt{size}}$).

Pour le taux d'arrivée des requêtes d'E/S ($\mathtt{rate}_{\mathtt{IO}}$), `fio` utilise tout le débit de transfert de données disponible vers le périphérique de stockage. Avant d'être utilisés, ces paramètres ont été normalisés (i.e. prennent des valeurs entre 0 et 1). Pour ce faire, le paramètre D prend la valeur 0 dans le cas d'un HDD et 1 dans

le cas d'un SSD. Pour $rate_{rnd}$ et $rate_{wrt}$, la normalisation est immédiate car ce sont des pourcentages. Les performances des périphériques de stockage sont représentés par le temps de traitement d'une requête d'E/S (i.e. $\frac{1}{rate_{IO}}$). Comme la taille maximale d'une requête d'E/S est de 1024Ko, nous avons donc choisi d'exprimer la taille des requêtes en mégaoctets afin de garder sa valeur entre 0 et 1. Ce processus nous permet d'identifier des paramètres représentatifs lors de la construction de nos modèles.

Comme nous l'avons mentionné dans la partie 6.3.3.1, `fio` est configuré pour exécuter des requêtes d'E/S synchrones et contourner les différents caches du système. L'objectif de cette configuration est de minimiser l'interférence avec la RAM, et ainsi d'isoler les activités CPU.

Les périphériques de stockage utilisés pour ces tests ont été formatés avant chaque expérimentation. Cela nous permet de garder le même état initial avant chaque expérimentation.

Chaque scénario a été exécuté cinq fois, et seules les moyennes des valeurs de sortie ont été retenues. Nous avons constaté que les résultats présentent un faible écart types (e.g. ~0.6% d'écart type pour une moyenne de 25% d'utilisation CPU).

#### 6.5.1.2 *Outils et configuration pour l'évaluation*

Les expérimentations ont été réalisées sur une PM dotée d'un microprocesseur Intel(R) Xeon E5-1620 3.70GHz, et d'une mémoire principale d'une capacité de 16Go. En ce qui concerne le système de stockage, nous avons utilisé deux partitions de tailles identiques de 128Go. Deux périphériques de stockage ont été employés : un HDD Seagate ST1000DM003 [171] et un SSD Samsung 840 PRO [57].

Pour la configuration logicielle, la PM tourne sous la distribution Debian 8 avec un noyau Linux 3.2.0. Les partitions de test ont été formatées et montées avec le système de fichiers EXT4. La version du *benchmark* `fio` est 2.10-2 [43].

Pour la modélisation et l'analyse des résultats, nous avons utilisé l'implantation Python de la suite d'outils pour l'apprentissage automatique `scikit-learn` [6], et l'implantation *open-source* de MARS [2].

#### 6.5.1.3 *Résultats d'évaluation du modèle de corrélation*

Après avoir obtenu les résultats de la charge CPU en fonction des paramètres de la charge d'E/S et des systèmes de stockage, nous les utilisons afin d'établir le modèle de corrélation CPU.

Comme mentionné dans la partie 6.3.3, nous avons fait appel à deux approches de régression pour modéliser la charge CPU. Les résultats des modèles OLS et MARS sont comparés en utilisant le coefficient de détermination $R^2$.

***a) Résultats du modèle OLS :*** le modèle de corrélation CPU obtenu en appliquant la méthode des moindres carrés OLS est donné par l'équation suivante :

$$\begin{aligned} cpu_{load}(D,W) =& 0.0353 + (-0.0232 \cdot rate_{rnd}) + (0.0331 \cdot rate_{wrt}) + (-0.0689 \cdot req_{size}) \\ &+ \left(-0.4346 \cdot \frac{1}{rate_{IO}}\right) + (0.1378 \cdot D) \end{aligned} \tag{38}$$

L'équation 38 du modèle OLS utilise tous les paramètres d'entrée pour modéliser la corrélation CPU. Nous pouvons remarquer que le taux d'arrivée des requêtes ($rate_{IO}$) représente le paramètre le plus important car il dispose d'un coefficient plus important comparé aux autres paramètres. Le coefficient de détermination $R^2$ est égal à 0.596, ce qui signifie que ce modèle reste peu précis et loin des résultats réels.

***b) Résultats du modèle MARS :*** le modèle de corrélation CPU obtenu à l'aide de la méthode MARS est donné par l'équation suivante :

$$\begin{aligned} cpu_{load}(D,W) =& \left[328.027 \cdot h\left(\frac{1}{rate_{IO}}, 0.00024\right)\right] + \left[9.4753 \cdot h\left(0.0038, \frac{1}{rate_{IO}}\right)\right] \\ &+ \left[-1154.85 \cdot h\left(\frac{1}{rate_{IO}}, 7.1882 \cdot 10^{-5}\right)\right] + \left[3771.77 \cdot h\left(7.1882 \cdot 10^{-5}, \frac{1}{rate_{IO}}\right)\right] \\ &+ \left[100.137 \cdot h\left(\frac{1}{rate_{IO}}, 0.00063\right)\right] + \left[726.126 \cdot h\left(\frac{1}{rate_{IO}}, 0.00012\right)\right] \\ &+ \left(0.0091 \cdot rate_{wrt}\right) + \left(0.0102 \cdot req_{size}\right) + \left(0.02397 \cdot D\right) + 0.1526 \end{aligned} \tag{39}$$

Contrairement au modèle OLS de l'équation 38, le modèle MARS obtenu par l'équation 39 choisit les paramètres d'entrée les plus significatifs. Nous pouvons constater que le paramètre le plus important est le taux d'arrivée des requêtes, qui est utilisé par toutes les fonctions charnières. La deuxième remarque concerne le paramètre du taux de requêtes aléatoires qui n'a pas été choisit par l'algorithme MARS. Le coefficient de détermination $R^2 = 0.98$ qui est plus proche de 1, ce qui signifie que ce modèle est plus précis que celui d'OLS. Nous rappelons que ce modèle dépend de la plateforme de test utilisée pour l'étape de *benchmarking*.

Les figures 50, 51, 52, et 53 montrent des exemples de la charge CPU obtenue en utilisant les modèles de régression OLS et MARS. Les exemples donnés concernent des opérations de lecture et d'écriture, aléatoires et séquentielles, pour HDD et SSD.

Nous constatons que la charge CPU est la plus haute pour les opérations d'E/S avec des requêtes de petite taille. La charge CPU baisse quand la taille des requêtes d'E/S augmente. Cette tendance est due au traitement d'un plus grand nombre de requêtes, ce qui est traduit par plus d'activité CPU dans le cas de petites requêtes.

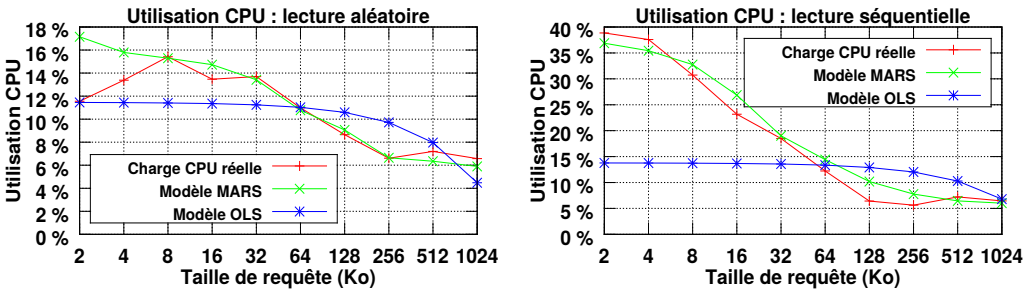


FIGURE 50 : SSD charge CPU pour lecture : réelle vs modèle

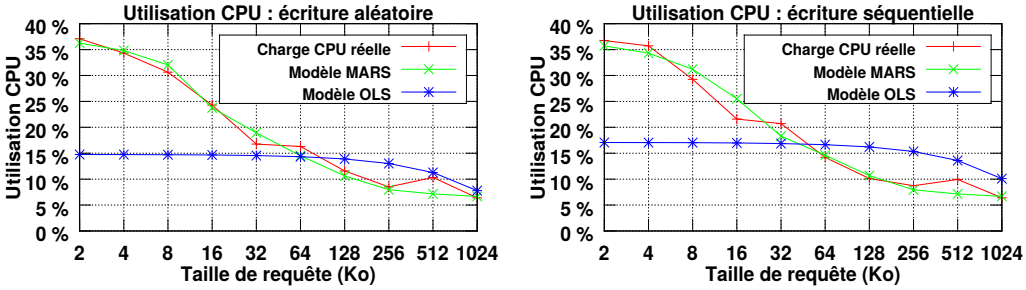


FIGURE 51 : SSD charge CPU pour écriture : réelle vs modèle

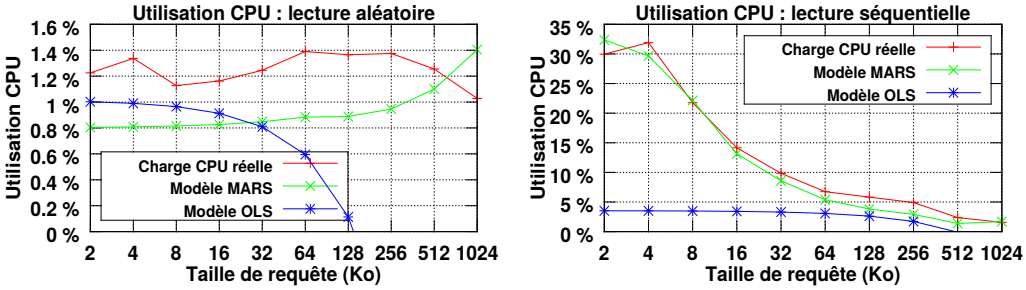


FIGURE 52 : HDD charge CPU pour lecture : réelle vs modèle

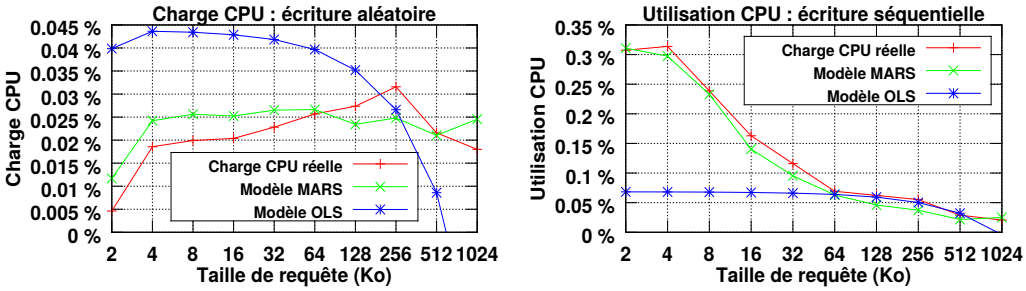


FIGURE 53 : HDD charge CPU pour écriture : réelle vs modèle

Nous remarquons également que la charge CPU est plus importante dans le cas des opérations séquentielles par rapport aux opérations aléatoires, particu-

lièrement pour HDD. De plus, la charge CPU observée dans le cas des opérations séquentielles est plus régulière. Cela est due aux latences causées par les performances du HDD dans le cas des E/S aléatoires.

Pour l'ensemble des cas étudiés, le modèle CPU obtenu à l'aide de la méthode MARS est celui qui s'adapte le mieux avec la charge CPU réelle. Le coefficient de détermination $R^2$ des deux modèles confirme cette observation ($R^2 = 0.98$ pour MARS et $R^2 = 0.596$ pour OLS).

Le modèle obtenu en utilisant la méthode MARS peut donc résoudre notre problème de corrélation CPU évoqué dans la partie 6.3.3. Compte tenu de sa précision, nous utilisons ce modèle dans notre extension de *CloudSim*.

### 6.5.2 *Évaluation de notre extension de CloudSim*

L'évaluation de notre extension a été effectuée en deux parties. La première partie vise à valider nos contributions sur le temps de traitement des E/S et la consommation énergétique dans *CloudSim*. La seconde partie concerne l'intégration du modèle de corrélation CPU obtenu dans la partie 6.5.1.

La figure 54 montre les étapes du processus de validation.

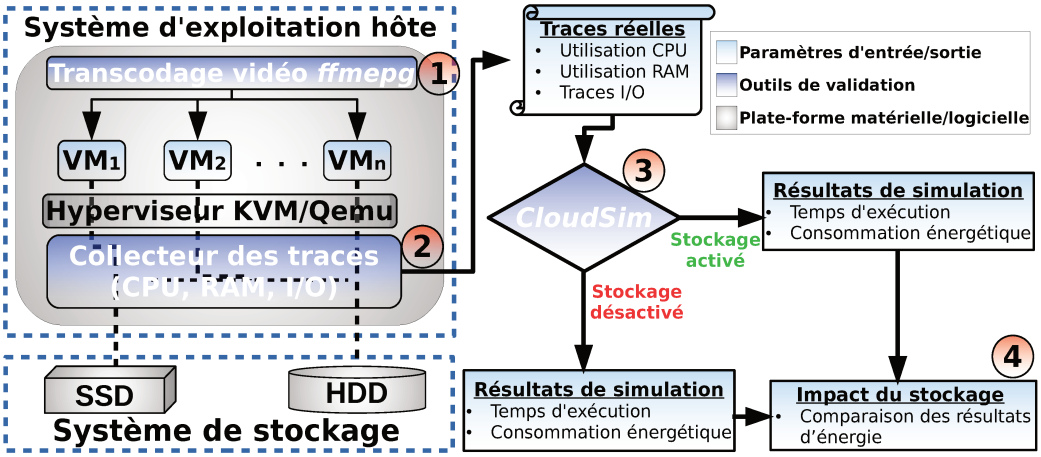


FIGURE 54 : méthode d'évaluation

Comme le montre la figure 54, l'évaluation de notre extension comporte quatre partie : 1) exécution des *benchmarks*, 2) collection des mesures, 3) simulation et collection des résultats, avec/sans stockage et traitement des E/S, 4) comparaison des résultats. Nous détaillons chacune de ces partie dans la partie suivante.

#### 6.5.2.1 *Description et configuration de l'évaluation de notre extension de CloudSim*

L'évaluation de notre extension de *CloudSim* s'effectue en deux phases :

1. l'exécution des charges dans un environnement réel (numéro 1 figure 54), ainsi que la collection des mesures de l'utilisation CPU, de l'utilisation mé-

moire, et des traces E/S (numéro 2 figure 54). L'utilisation CPU comprend le traitement des E/S et d'autres activités. Notre modèle de corrélation CPU étudié auparavant est utilisé pour quantifier la charge CPU qui est exclusivement liée au traitement des E/S.

2. la simulation des charges en intégrant les mesures et les traces collectées précédemment (numéro 3 figure 54), en activant/désactivant le stockage et le traitement des E/S, puis la quantification de l'impact des E/S (numéro 4 figure 54).

Nous détaillons ci-dessous les différentes étapes d'évaluation décrites dans la figure 54.

***1) Exécution des charges*** : cette étape consiste à exécuter une charge de travail dans les VM (étape 1 dans la figure 54). Comme étude de cas, nous avons choisi un *benchmark* de transcodage vidéo dans un environnement *cloud*. Les VM transcodent une vidéo du format `mov`, en plusieurs vidéos au format TS (*Transport Stream*). Ce processus de transcodage est utilisé pour le streaming vidéo fonctionnant avec le protocole HLS (*HTTP Live Streaming*). Pour le processus de transcodage, nous avons choisi les normes H264 pour la vidéo et HE-AAC pour l'audio. Huit VM ont été exécutées : quatre VM stockées sur HDD et les quatre autres sur SSD. Nous avons exécuté quatre expérimentations en faisant varier le nombre de VM par périphérique de stockage (HDD, SSD) comme suit : (1,1), (2,2), (3,3), et (4,4). Toutes les VM ont les mêmes caractéristiques : une image disque de 20Go, 1 vCPU et 1Go de RAM.

***2) Collection des traces*** : l'utilisation CPU et mémoire ainsi que les opérations d'E/S ont été tracées durant l'exécution des charges de travail. Les traces CPU et mémoire ont été agrégées en une valeur moyenne qui correspond à une unité de temps fixe (définie à 5 minutes dans *CloudSim*). Concernant les E/S, chaque VM possède un fichier de traces d'E/S capturées au niveau bloc (voir partie 4.3.4).

***3) Simulation des traces réelles*** : les traces obtenues dans l'étape numéro 2 sont rejouées dans cette phase. L'utilisation CPU et mémoire dépendent de la PM, alors que les traces d'E/S sont liées aux VM. Les simulations de cette étape ont été effectuées en faisant varier les paramètres suivants :

- le nombre de VM exécutée par PM ;
- l'activation et la désactivation du système de stockage ;
- le stockage utilisé (HDD ou SSD), si le système de stockage est activé.

Concernant le nombre de VM par PM, nous avons gardé la même configuration utilisée lors de l'exécution des charges réelles. Suivant le paramétrage de la va-

lidation effectuée dans [26], le nombre total de VM a été fixé à 290 VM. Ainsi, le nombre de PM actives dépend du nombre de VM par hôte, car les PM inactives sont éteintes.

***4) Comparaison des résultats de simulation*** : nous avons utilisé la consommation énergétique générée par le stockage et les E/S, afin de montrer et de quantifier l'impact du traitement des charges d'E/S et du système de stockage sur la consommation totale du système.

Dans la partie suivante, nous montrons les résultats obtenus à partir de cette évaluation.

#### 6.5.2.2 *Résultats d'évaluation de notre extension de CloudSim*

Les résultats obtenus après l'intégration de notre extension sont présentés en deux parties. La première partie des résultats montre la différence entre les simulations avec et sans prise en compte des E/S. L'objectif de cette évaluation est de confirmer l'impact du système de stockage et des types de périphérique de stockage sur la consommation énergétique totale.

La seconde partie évalue l'impact de chaque composant contribuant à l'exécution des charges d'E/S (i.e. système de stockage et CPU). La figure 55 montre l'énergie consommée en désactivant puis en activant le système de stockage et le traitement des E/S.

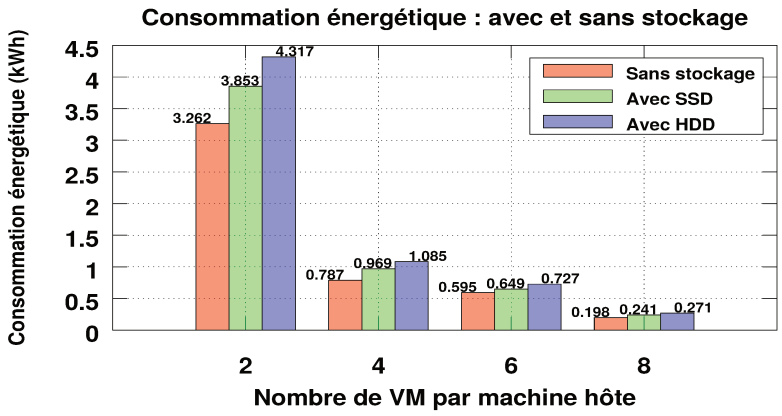


FIGURE 55 : consommation énergétique avec et sans E/S

Le premier constat concerne la tendance de la consommation énergétique qui baisse en augmentant le nombre de VM par PM. Cette tendance est due au nombre de PM actives qui diminue en augmentant le nombre de VM exécutées par PM (i.e. consolidation des serveurs). La deuxième observation est la différence entre l'énergie consommée sans et avec système de stockage et traitement des E/S. On constate que le système de stockage et le traitement des E/S peut

consommer une part significative de l'énergie totale (~25% dans le cas d'un HDD et ~17% dans le cas d'un SSD).

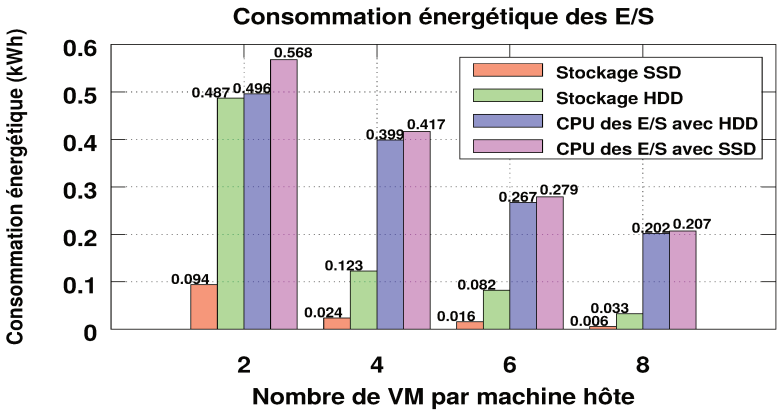


FIGURE 56 : consommation énergétique liée aux E/S

La figure 56 montre l'énergie consommée par les périphériques de stockage (i.e. HDD et SSD), ainsi que celle consommée par le CPU lors de l'exécution des E/S sur chaque périphérique de stockage.

Nous pouvons constater que le CPU est le composant le plus énergivore dans tous les cas présentés. Ces résultats confirment l'importante de la charge CPU observée dans l'évaluation du modèle de corrélation (voir partie 6.5.1 partie III). On remarque que la quantité d'énergie consommée par le CPU pour les deux types de périphérique de stockage (i.e. HDD et SSD) est presque la même (avec ~5% de différence). Cela est du au profil d'accès exhibé par les charges d'E/S des VM (i.e. ~88% de taux de séquentialité et ~82% taux de lecture), qui donne lieu à une charge CPU quasi-identique pour SSD et HDD (voir figures 50 et 52). Même si le CPU consomme la part la plus importante d'énergie, les périphériques de stockage en consomment également.

## 6.6 CONCLUSION

En s'appuyant sur le modèle MAPE-K, notre étude a pour objectif de considérer les systèmes de stockage dans le processus d'optimisation de placement de VM. Après avoir modélisé les coûts de stockage et d'exécution des E/S des VM dans l'étape ANALYZE, nous avons mis en œuvre cette modélisation dans le but de l'utiliser par la suite dans les approches d'optimisation de placement de VM dans l'étape PLAN.

Nous avons étendu le simulateur *CloudSim* afin de supporter des systèmes de stockage hybrides et ainsi permettre de simuler des charges d'E/S pouvant être exécutées par les VM.

Nous avons proposé des modèles pour le calcul du temps de traitement des E/S, pour l'utilisation CPU, et pour la consommation énergétique liées à l'exécution des E/S des VM et aux systèmes de stockage. Pour la prédiction de l'utilisation CPU due à l'exécution des E/S, nous avons proposé et évalué un modèle de régression basé sur une approche d'apprentissage automatique.

L'évaluation de notre modèle d'utilisation CPU liée aux E/S a montré une précision de l'ordre de 0.98 exprimée en coefficient de corrélation $R^2$. La validation de notre extension de *CloudSim* a révélé que les systèmes de stockage et l'exécution des opérations d'E/S peuvent consommer jusqu'à 25% de la consommation totale d'un centre de données. Ces résultats confirment l'importance de la prise en compte des systèmes de stockage et des E/S dans les approches d'optimisation de placement de VM.

Le chapitre suivant présente l'impact de la prise en compte du traitement des E/S et des systèmes de stockage, lors de la conception des approches d'optimisation de placement de VM.

# Chapitre 7

## ÉTAPE *PLAN* : OPTIMISATION DES COÛTS EN CONSIDÉRANT LE STOCKAGE ET LES ENTRÉES/SORTIES DES VM

Les approches pour le placement de VM proposées dans l'état de l'art ne considèrent pas les coûts engendrés par les charges d'E/S des VM.

Nous avons donc proposé un modèle de coût qui permet l'estimation des coûts de stockage et d'exécution des E/S des VM au chapitre 5.

La mis en œuvre de notre modèle sur le simulateur de *cloud* IaaS *CloudSim* est présentée dans le chapitre 6. L'objectif du travail décrit dans ce chapitre est d'utiliser ce modèle pour le placement de VM, afin de montrer l'impact des E/S dans une telle approche.

Ce chapitre présente nos contributions représentant l'étape *Plan* du modèle MAPE-K. Nous avons proposé et comparé deux types d'approche d'optimisation : 1) une méthode exacte, et 2) trois heuristiques. La méthode exacte effectue une recherche exhaustive de la totalité des solutions de placement de VM possibles, puis choisit celle donnant le coût minimum. Comme heuristiques, nous proposons une approches gloutonne et deux approches modélisant le placement de VM comme un problème de *Bin Packing* [52]. L'approche gloutonne parcourt l'ensemble de VM et l'ensemble de PM et périphériques de stockage dans le but de trouver un meilleur placement (i.e. moins coûteux) que celui actuel. Les deux approches heuristiques s'inspirent de l'algorithme *Best Fit Decreasing* [98]. Nous avons évalué l'ensemble de ces approches avec le simulateur *CloudSim*.

Comparés aux approches de placement implantées dans *CloudSim* [25], nos approches trouvent des solutions réduisant jusqu'à 5 fois le coût d'utilisation de l'infrastructure *cloud*.

Ce chapitre est organisé comme suit. Premièrement, nous définissons quelques principes utilisés dans les différentes parties du chapitre. Ensuite, nous présentons un aperçu sur le placement de VM pour la consolidation. La troisième partie détaille nos contributions sur le placement de VM avec la prise en compte des

E/S. La quatrième partie présente l'évaluation de nos contributions. La cinquième partie conclut le chapitre.

## Sommaire

## 7.1 DÉFINITIONS

**Définition 7.1.** *(**sur-utilisation**) [26] : une PM est dite sur-utilisée lorsque son utilisation CPU dépasse un seuil maximum préalablement défini ou dynamiquement calculé.*

**Définition 7.2.** *(**sous-utilisation**) [26] : une PM est dite sous-utilisée lorsque son utilisation CPU est en dessous d'un seuil minimum préalablement défini ou dynamiquement calculé.*

## 7.2 ÉTAT DE L'ART : CONSOLIDATION ET OPTIMISATION DU PLACEMENT DE VM

La consolidation pour le placement de VM peut être divisée en quatre étapes principales [25] :

1. désigner l'ensemble de PM considérées comme sur-utilisées, puis faire migrer certaines VM depuis ces PM pour baisser leur consommation énergétique et éviter d'éventuelles pénalités ;
2. désigner l'ensemble de PM considérées comme sous-utilisées, puis faire migrer toutes les VM depuis ces PM, puis mettre ces PM en mode veille pour baisser leur consommation énergétique ;
3. dans le cas d'une sur-utilisation de PM, sélectionner quelles VM doivent être migrées depuis ces PM ;
4. une fois les VM à migrer désignées, trouver le nouveau plan de placement de ces VM.

Le pseudo-code 1 donne un aperçu sur la consolidation pour le placement de VM proposée dans [25].

Le pseudo-code 1 montre que la consolidation requiert la détection des PM sous-utilisées et sur-utilisées. D'abord, l'approche de placement parcourt l'ensemble de PM (ligne 1) à la recherche des PM sur-utilisées (ligne 2). Pour chaque PM sur-utilisée, l'algorithme de sélection choisit les VM à migrer (ligne 3). Afin d'éviter une sur-utilisation ultérieure, la PM sur-utilisée est exclue du calcul du nouveau plan de placement de VM (ligne 4). L'ensemble de PM est à nouveau parcouru (ligne numéro 7) afin de désigner les PM sous-utilisées (ligne numéro 8). Toutes les VM des PM sous-utilisées sont alors sélectionnées pour être migrées (ligne numéro 9). Les PM sous-utilisées sont mises en mode veille et écartées du calcul du nouveau plan (lignes 10 et 11 respectivement). À présent, le calcul d'un nouveau plan de placement de VM peut être effectué (ligne numéro 14).

Nous allons décrire les différentes étapes illustrées par le pseudo-code 1 dans *CloudSim*.

**Algorithme 1 :** le placement de VM dans *CloudSim*

**Données :** Plan de placement actuel (liste des PM, liste des VM)
**Résultat :** Plan de placement de VM

```
1  pour chaque PM dans la liste des PM faire
2      si PM est sur-utilisée alors
3          Sélectionner les VM à migrer depuis PM
           /* pour éviter une future sur-utilisation                    */
4          Exclure PM du calcul du nouveau plan de placement de VM
5      fin
6  fin
7  pour chaque PM dans la liste des PM faire
8      si PM est sous-utilisée alors
9          Migrer toutes les VM depuis PM
10         Mettre PM en mode veille
11         Exclure PM du calcul du nouveau plan de placement de VM
12     fin
13 fin
14 Calculer le nouveau plan de placement pour les VM sélectionnées
15 retourner Nouveau plan de placement de VM
```

### 7.2.1 *Algorithmes de détection de sur-utilisation des PM*

*CloudSim* se base sur l'utilisation CPU pour le placement de VM et la détection de sous-utilisation et de sur-utilisation des PM.

Pour la détection des PM sur-utilisée, il existe deux approches implantées dans *CloudSim* : 1) détection avec seuil statique, 2) détection avec seuil dynamique. Nous allons présenter brièvement chacune de ces approches.

#### 7.2.1.1 *Détection avec seuil statique*

Cette approche a été initialement proposée par les developpeurs de *CloudSim* dans [26]. L'approche consiste à définir des seuils statiques minimum et maximum d'utilisation CPU. L'objectif est de garder l'utilisation CPU de toutes les PM entre ces deux seuils. Si l'utilisation CPU d'une PM dépasse le seuil supérieur, elle est considérée comme sur-utilisée. Dans le cas où l'utilisation CPU d'une PM tombe en dessous du seuil inférieur, la PM est considérée comme sous-utilisée.

#### 7.2.1.2 *Détection avec seuil dynamique*

Pour faire face à l'imprédictibilité de la charge CPU des VM, les auteurs de [26] ont proposé des approches utilisant des seuils dynamiques (i.e. *Adaptive Utilization Threshold*) [25]. Le principe consiste à ajuster automatiquement le seuil supérieur

d'utilisation CPU, en se basant sur l'historique d'utilisation CPU des VM, ce qui peut être fait de plusieurs façons différentes.

***a) Écart absolu médian (Median Absolute Deviation)*** [25] : cette approche utilise la déviation de l'utilisation CPU pour ajuster le seuil supérieur de l'utilisation CPU. La valeur du seuil supérieur baisse avec l'augmentation de la déviation. En effet, une déviation importante peut facilement conduire à 100% d'utilisation CPU. Cette surcharge ne permet pas de satisfaire tous les besoins des VM ce qui donne lieu à des pénalités.

La métrique de calcul de la déviation de l'utilisation CPU proposée dans [25] est l'écart absolu médian (*Median Absolute Deviation* ou MAD). Pour une PM donnée, l'historique de l'utilisation CPU est représenté sous la forme $u_1, u_2, ..., u_n$ où $u_i$ est l'utilisation CPU mesurée à l'instant $t_i$. Le MAD est donné comme suit :

$$MAD = median(|u_i - median(u)|) \tag{40}$$

où $median()$ est la fonction qui calcule la valeur médiane des écarts absolus (calculées par $|u_i - median(u)|$), et $u_i$ est la valeur médiane de l'historique d'utilisation. Le seuil supérieur d'utilisation CPU (noté $T_u$) est donc obtenu à partir de la valeur du MAD comme suit :

$$T_u = 1 - s \cdot MAD \tag{41}$$

où le paramètre $s$ est un nombre réel positif définissant l'agressivité de la consolidation. Plus la valeur de $s$ est grande, plus petite sera la consommation énergétique avec une probabilité plus importante de violation de SLA.

***b) Écart inter-quartiles (Interquartile Range)*** [25] : cette approche est basée sur l'écart inter-quartiles (*Interquartile Range* ou IQR). C'est une autre métrique statistique de calcul de dispersion des valeurs d'une variable statistique. Pour une distribution donnée, l'IQR est la différence entre le premier quartile (noté $Q_1$) et le troisième quartile (noté $Q_3$) :

$$IQR = Q_3 - Q_1 \tag{42}$$

Le seuil supérieur d'utilisation CPU (noté $T_u$) est calculé comme suit :

$$T_u = 1 - s \cdot IQR \tag{43}$$

où le paramètre $s$ définit l'agressivité du processus de consolidation.

***c) Régression locale (Local Regression)*** [25] : la troisième approche utilise un modèle de prédiction de l'utilisation CPU basé sur une régression locale (*Local Regression* ou LOESS) [50]. Cette approche n'est pas utilisée pour définir un seuil supérieur mais plutôt pour prédire l'utilisation CPU et décider si une sur-utilisation

pourrait avoir lieu. Dans ce cas, une PM est considérée comme sur-utilisée si, et seulement si, les conditions suivantes sont satisfaites :

$$s \cdot \widehat{g}(u_{k+1}) \geqslant 1 \quad \text{et} \quad u_{k+1} - u_k \leqslant t_m \tag{44}$$

où le paramètre $s$ est utilisé pour définir l'agressivité de la consolidation, $\widehat{g}(u_{k+1})$ est la fonction de prédiction de l'utilisation future du CPU $u_{k+1}$, $u_k$ est l'utilisation CPU actuelle, et $t_m$ est le temps maximum de migration d'une VM depuis la PM en question. Cette approche est nommée LR (*Local Regression*) dans *CloudSim*. Il existe une deuxième version de la régression locale proposée dans [25] et appelée regression locale robuste (*Local Regression Robust* ou LRR) [51].

### 7.2.2 *Politiques de sélection de VM*

Comme le décrit le pseudo-code 1, une fois qu'une PM est désignée comme sur-utilisée (ligne numéro 2), la politique de sélection est appelée (ligne numéro 2) afin de choisir les VM à migrer depuis cette PM. Trois politiques de sélection de VM ont été proposées et implantées dans *CloudSim* [25] [26]. Cette partie présente ces politiques.

#### 7.2.2.1 *Choix aléatoire (RC)*

Parmi l'ensemble de VM présentes dans une PM sur-utilisée, cette politique choisit aléatoirement la VM à migrer. Le choix de la VM suit une variable aléatoire discrète uniformément distribuée.

#### 7.2.2.2 *Temps minimum de migration (MMT)*

La politique MMT (*Minimum Migration Time*) choisit la VM qui prend le moins de temps pour être migrée depuis la PM sur-utilisée. Le temps de migration $T_{mig}(VM)$ d'une VM (noté VM) est calculé en fonction de la taille de la mémoire $RAM_u(VM)$ utilisée par la VM et la bande passante réseau $pm_{net}$ allouée à la PM.

$$T_{mig}(VM) = \frac{RAM_u(VM)}{pm_{net}} \tag{45}$$

#### 7.2.2.3 *Minimum d'utilisation CPU (MU)*

La politique MU (*Minimum Utilization*) est basée sur le rapport (noté $CPU_{ratio}(VM)$) entre la capacité CPU réellement utilisée par une VM (notée $CPU_{current}(VM)$) et celle demandée au départ (notée $CPU_{requested}(VM)$).

$$CPU_{ratio}(VM) = \frac{CPU_{current}(VM)}{CPU_{requested}(VM)} \tag{46}$$

Une VM avec un écart important entre sa demande et son utilisation réelle en terme de capacité CPU a plus de chances de causer une sur-utilisation de la PM. Elle sera alors une bonne candidate pour la migration. Pour une PM sur-utilisée, les VM à migrer sont celles qui présentent un ratio minimum d'utilisation CPU.

#### 7.2.2.4 *Maximum de corrélation (MC)*

L'idée de base de la politique MC (*Maximum Correlation*) vient de l'étude présentée dans [189]. Ce travail propose une approche basée sur la corrélation entre les applications partageant les mêmes ressources. Plus cette corrélation est importante, plus il est probable que les ressources soient sur-utilisées. Sur *CloudSim*, les applications sont représentées par les VM et la ressource partagée qui est le CPU. Le principe est de prédire l'utilisation CPU à l'aide de l'historique d'utilisation CPU de toutes les VM. Parmi l'ensemble de VM, cette politique choisit celle qui a le coefficient de corrélation le plus élevé par rapport à la prédiction obtenue.

### 7.2.3 *Algorithme de placement de VM*

Cette partie présente l'algorithme de placement de VM (lignes numéros 6 et 12 pseudo-code 1). Le placement de VM dans *CloudSim* est modélisé comme un problème de *bin packing*. Les PM représentent les boites avec une capacité limitée en terme d'utilisation CPU, les VM représentent les objets avec une taille définie en terme de demande CPU, et la fonction coût est représentée par la consommation énergétique engendrant le placement d'une VM dans une PM.

Comme le problème de *bin packing* est un problème NP-complet, une heuristique basée sur l'approche *Best Fit Decreasing* (BFD) à été proposée et implantée dans *CloudSim* [26]. L'approche PABFD (*Power Aware Best Fit Decreasing*) proposée dans [26] trie d'abord les VM en ordre décroissant selon leur utilisation CPU. Ensuite, pour chaque VM, l'heuristique parcourt l'ensemble des PM susceptibles de l'accueillir afin de choisir celle qui présente la consommation énergétique minimale.

Nous avons présenté un aperçu du placement de VM sur *CloudSim*. Les parties suivantes sont organisées comme suit. La prochaine partie présente nos contributions sur le placement de VM avec prise en compte des E/S et des système de stockage. Ensuite, nous détaillons la méthode et les résultats d'évaluation de nos approches de placement. Enfin, nous concluons ce chapitre.

## 7.3 PRISE EN COMPTE DES COÛTS DES E/S DANS LE PLACEMENT DE VM

Comme nous l'avons présenté dans les parties précédentes, les travaux de l'état de l'art sont majoritairement basés sur l'utilisation CPU. Nous avons démontré que le traitement des E/S peut avoir un effet important sur les coûts des infrastructures *cloud* (voir chapitres 5 et 6).

L'objectif de notre étude est de combler ces faiblesses en prenant compte les coûts des E/S dans le placement de VM. Cette partie détaille nos approches de placement de VM, implanté et évalué dans *CloudSim* (voir partie 7.2.3). Nous donnons d'abord une formulation mathématique du problème. Nous présentons ensuite nos contributions avant de les évaluer.

### 7.3.1 *Formulation du problème*

Avant de détailler nos approches pour résoudre ce problème, nous rappelons l'architecture générale du système faisant l'objet de notre étude.

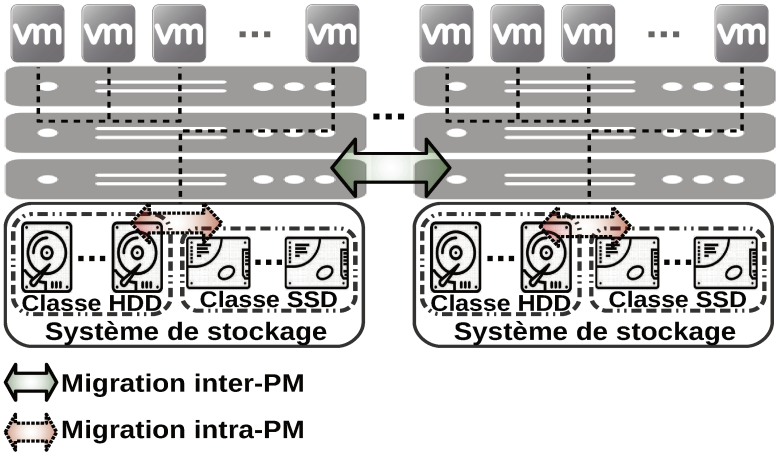


FIGURE 57 : architecture du système à optimiser

Comme le montre la figure 57, notre système représente un centre de données constitué d'un ensemble de PM, qui a leur tour hébergent un ensemble de VM. Les PM sont dotées d'un système de stockage hybride regroupant HDD et SSD. Ce système de stockage est utilisé par les VM afin d'exécuter leur charges d'E/S. Nous cherchons à trouver le meilleur plan de placement de VM qui permette de minimiser les coûts de l'infrastructure, tout en prenant en compte les coûts des E/S. Pour ce faire, nous utilisons la migration des VM appliquée à deux niveaux :

- au niveau du système de stockage d'une PM ;
- entre les PM constituant le centre de données.

Le problème de placement de VM prend en entrée un centre de données avec un ensemble de $n$ VM à placer sur $m$ PM. À chaque PM, un ensemble de $l$ périphériques de stockage est attaché.

Pour formaliser le problème de placement, nous avons donc l'ensemble de VM :

$$VM = \{VM_1, VM_2, \ldots, VM_n\}$$

Chaque VM demande une configuration particulière en terme de ressources matérielles CPU, RAM, et stockage :

$$\forall\, VM_i \in VM, VM_i \text{ est défini par } < vm_{CPU_i}, vm_{RAM_i}, vm_{size_i}, vm_{IOPS_i} >$$

où $vm_{CPU_i}$, $vm_{RAM_i}$, $vm_{size_i}$, et $vm_{IOPS_i}$ représentent respectivement les demandes de la VM $VM_i$ en termes de capacité CPU, de capacité mémoire, de capacité de stockage, et de performances de stockage. La demande en capacité CPU $vm_{CPU_i}$ peut être exprimée en nombre de cœurs CPU virtuels (i.e. vCPU) avec une fréquence définie, tandis que la capacité mémoire $vm_{RAM_i}$ est exprimée en mégaoctets.

L'interaction entre chaque VM et le système de stockage est représentée par une charge d'E/S. Nous avons donc un ensemble $W$ de charges d'E/S du même cardinal $n$ que l'ensemble $VM$ :

$$W = \{W_1, W_2, \ldots, W_n\}$$

où $w_i$ est la charge d'E/S de la VM $VM_i$.

Comme nous l'avons détaillé dans la partie 4.4, chaque charge d'E/S est modélisée par un ensemble de paramètres :

$$\forall\, W_i \in W, W_i \text{ est défini par } < rate_{rnd_i}, rate_{wrt_i}, rate_{IO_i}, req_{size_i}, data_{amount_i} >$$

Les VM sont hébergées dans le centre de données qui est constitué d'un ensemble de $m$ PM :

$$PM = \{PM_1, PM_2, \ldots, PM_m\}$$

Chaque PM dispose d'une configuration matérielle :

$$\forall\, PM_j \in PM, PM_j \text{ est définie par } < pm_{CPU_j}, pm_{RAM_j}, pm_{net_j}, D_{1_j}, D_{2_j}, \ldots, D_{l_j} >$$

où $pm_{CPU_j}$, $pm_{RAM_j}$, et $pm_{net_j}$ représentent respectivement les capacités de la PM $PM_j$ en termes de capacité CPU, de mémoire, et de réseaux, tandis que $D_{k_j}$ désigne le périphérique de stockage $k$ attaché à la PM $PM_j$ avec $k \in \{1, \ldots, l\}$.

Chaque périphérique de stockage a une capacité de stockage, et des performances en IOPS :

$$\forall\, PM_j \in PM, \forall\, D_{k_j} \in PM_j, D_{k_j} \text{ est définie par } < d_{cap_{k_j}}, d_{IOPS_{k_j}} >$$

où $d_{cap_{k_j}}$ et $d_{IOPS_{k_j}}$ désignent respectivement la capacité et les performances du périphérique de stockage $D_{k_j}$. Dans notre étude, nous prenons en compte un système de stockage hybride composé de deux classes de stockage (i.e SSD et HDD) :

$$\forall\, PM_j \in PM, \forall\, D_{k_j} \in PM_j, D_{k_j} \in \{SSD, HDD\}$$

Après avoir défini toutes les variables nécessaires pour la formulation du problème, nous donnons la forme linéaire du problème d'optimisation :

$$\begin{cases} Min\Big(\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{l} Cost(VM_i, D_{k_j})\Big), VM_i \in VM, D_{k_j} \in PM_j \text{ tel que } PM_j \in PM & (47a) \\ \forall\, PM_j \in PM, \sum_{i=1}^{n} vm_{CPU_i} \leqslant pm_{CPU_j}, VM_i \text{ s'exécute sur } PM_j & (47b) \\ \forall\, PM_j \in PM, \sum_{i=1}^{n} vm_{RAM_i} \leqslant pm_{RAM_j}, VM_i \text{ s'exécute sur } PM_j & (47c) \\ \forall\, PM_j \in PM, \forall\, D_{k_j} \in PM_j, \sum_{i=1}^{n} vm_{size_i} \leqslant d_{cap_{k_j}}, VM_i \text{ est stockée sur } D_{k_j} & (47d) \\ \forall\, PM_j \in PM, \forall\, D_{k_j} \in PM_j, \sum_{i=1}^{n} vm_{IOPS_i} \leqslant d_{IOPS_{k_j}}, VM_i \text{ est stockée sur } D_{k_j} & (47e) \end{cases}$$

Notre problème de placement de VM a pour fonction objectif (équation 47a) la minimisation du coût de placement de l'ensemble de VM sur l'ensemble des PM en prenant en compte leur coût des charges d'E/S exécutées sur les périphériques de stockage. Nous utilisons notre modèle de coût présenté dans le chapitre 5 comme fonction objectif.

La première contrainte (équation 47b) assure pour une PM donnée, que la demande de capacité CPU des VM qui y sont hébergées ne dépasse pas la capacité totale de la PM.

De la même manière, la deuxième contrainte (ligne 47c) vérifie que la capacité totale de la mémoire d'une PM ne peut pas être dépassée par les demandes des VM. La troisième et la quatrième contraintes concernent les demandes des VM en terme de capacité et de performance du stockage. Le système de stockage d'une PM doit avoir une capacité de stockage (ligne 47d) et des performances (ligne 47e) suffisantes pour répondre aux besoins des VM qui y sont stockées.

Après avoir formulé notre problème de placement de VM sous forme d'un problème d'optimisation, nous allons décrire nos approches pour la résolution de ce système linéaire. Nous proposons quatre approches. Les prochaines parties présentent nos solutions sous forme d'une méthode exacte et trois approches heuristiques.

### 7.3.2 *Méthode exacte (recherche exhaustive)*

Avant d'étudier les heuristiques, nous avons d'abord estimé les limites d'une résolution exacte du placement de $n$ VM sur $l$ périphériques de stockage. Nous

avons donc implanté et évalué une approche exacte d'énumération des solutions de placement de VM. Pour ce faire, nous énumérons exhaustivement toutes les possibilités de placement de VM, puis nous choisissons le plan de placement le moins coûteux.

Dans le cas d'un système contenant $m$ périphériques de stockage et $n$ VM, la méthode exacte est d'une complexité de $\mathcal{O}(m^n)$ en temps de calcul.

### 7.3.3 *Approches heuristiques*

Pour faire face à la complexité exponentielle de la méthode exacte, nous proposons dans cette partie trois heuristiques. Nous présentons une méthode gloutonne et deux méthodes basées sur l'algorithme *Best Fit Decreasing*.

#### 7.3.3.1 *Approche 1 : méthode gloutonne*

Cette partie présente une méthode gloutonne de placement de VM avec prise en compte des coûts des E/S. Le principe consiste à minimiser le coût d'un placement déjà en place en déplaçant certaines VM. Le pseudo-code 2 décrit les différentes étapes de notre méthode gloutonne.

Le pseudo-code 2 parcourt l'ensemble de VM dans le centre de données (ligne numéro 1). Puis, nous initialisons le coût de placement ainsi que la Physical Machine (PM) et le périphérique de stockage alloué à la VM courante (lignes numéros 2, 3, et 4). Ensuite, les PM sont parcourues (ligne numéro 5) et la possibilité d'un placement est évaluée (ligne numéro 6). Cette évaluation vérifie les contraintes 47b et 47c de notre programme linéaire (voir partie 7.3.1). Les périphériques de stockage de la PM sont parcourus (ligne numéro 7) et la possibilité d'accueillir la VM est vérifiée (ligne numéro 8). Cette vérification porte sur les contraintes 47d et 47e de notre programme linéaire (voir partie 7.3.1).

Si toutes les contraintes sont vérifiées, le nouveau coût de placement de la VM est calculé (ligne numéro 9) en utilisant la fonction 47c du programme linéaire. Si le coût du nouveau placement de la VM est inférieur au coût du placement actuel (ligne numéro 10), alors nous mettons à jour les valeurs du coût de placement, la PM et le périphérique de stockage alloués à cette VM (lignes numéros 11, 12, et 13). Enfin, si une VM est déplacée (ligne numéro 19), nous sauvegardons le nouveau placement obtenu (ligne numéro 20).

Dans le cas d'un centre de données avec $n$ VM, $m$ PM, et $l$ périphériques de stockage, la méthode gloutonne présente une complexité de l'ordre de $\mathcal{O}(n \cdot m \cdot k)$.

**Algorithme 2 :** méthode gloutonne de placement de VM

**Données :** Placement(VM, PM)
**Résultat :** Placement(VM, PM)

```
/* parcourir les VM présentes dans le centre de données                    */
pour chaque VM_i dans VM faire
    Cost_min ← MAX_VALUE
    PM_allocated ← NULL
    D_allocated ← NULL
    pour chaque PM_j dans PM faire
        /* cette PM_j peut-elle accueillir VM_i ?                          */
        si PM_j peut héberger VM_i alors
            /* parcourir les périphériques de stockage de PM_j             */
            pour chaque D_kj dans PM_j faire
                /* le périphérique D_kj peut-il accueillir VM_i ?          */
                si D_kj peut stocker VM_i alors
                    /* évaluer le coût de placement de VM_i sur D_kj       */
                    Cost_after_alloc ← Cost(VM_i, D_kj)
                    /* ce placement est-il meilleur que l'actuel ?         */
                    si Cost_after_alloc < Cost_min alors
                        Cost_min ← Cost_after_alloc
                        PM_allocated ← PM_j
                        D_allocated ← D_kj
                    fin
                fin
            fin
        fin
    fin
    /* sauvegarder le nouveau placement pour la vm                         */
    si PM_allocated ≠ NULL et D_allocated ≠ NULL alors
        Mettre à jour Placement avec (VM_i, PM_allocated, D_allocated)
    fin
fin
retourner Placement
```

#### 7.3.3.2 *Approches 2 et 3 : Best Fit Decreasing (BFD) modifié*

Nos heuristiques s'inspirent de l'algorithme *Best Fit Decreasing* et de l'algorithme de placement présenté dans [26]. Nous rappelons que les approches proposées dans [26] se basent sur l'utilisation CPU dans le but de minimiser la consommation énergétique.

Nous proposons deux versions de l'algorithme *Best Fit Decreasing*. La première version est basée sur le principe de la consolidation au niveau des périphériques de stockage. L'idée consiste à regrouper le maximum de VM sur un minimum de périphériques de stockage tout en profitant des caractéristiques de ces derniers.

La seconde version applique la consolidation à un niveau plus haut, notamment au niveau des PM. Son principe consiste à regrouper le maximum de VM sur un minimum de PM.

Dans cette catégorie d'approche, nous exploitons les caractéristiques des périphériques de stockage afin d'améliorer le placement de VM. Comme nous l'avons présenté en 2.2, les SSD présentent de meilleures performances pour les opérations de lectures aléatoires comparées aux HDD. De plus, l'une des faiblesses des SSD est leur usure qui est corrélée avec les opérations d'écriture. En revanche, à cause de leur conception mécanique, les HDD sont moins performants et consomment plus d'énergie dans le cas des E/S aléatoires. Nous avons utilisé ces caractéristiques afin de construire nos heuristique de placement de VM.

Dans ce qui suit, nous détaillons nos deux heuristiques de placement de VM.

***1) HPSD : Heuristic with Packing into Storage Devices*** : la première approche HPSD applique la consolidation en favorisant le placement de VM sur les périphériques de stockage de type SSD. Le pseudo-code 3 décrit le principe de l'approche HPSD.

L'idée fondamentale de cette heuristique est de sélectionner les VM exposant des charges d'E/S avec un fort taux de lectures aléatoires pour être placées sur SSD. Les VM exécutant plus d'opérations d'écritures séquentielles seront sélectionnées pour être placées sur HDD.

Nous trions par ordre décroissant l'ensemble de VM en fonction du rapport taux d'aléatoire/taux d'écriture (ligne numéro 2). Cette opération permet de mettre les VM avec un fort taux de lectures aléatoires au début de la liste.

Ensuite, nous regroupons l'ensemble des périphériques de stockage de toutes les PM par type HDD ou SSD (lignes numéro 3 et 4 respectivement). Les listes des périphériques de stockage sont triées par ordre décroissant en fonction du produit de leur capacité de stockage par leur performance (lignes numéro 5 et 6). L'objectif de cette opération est de mettre en têtes des listes les périphériques de stockage pouvant accueillir le maximum de VM (i.e. en terme de capacité et/ou de performance), et ainsi minimiser le nombre de périphériques utilisés (i.e. principe de la consolidation au niveau des périphériques de stockage).

**Algorithme 3 :** Heuristic with Packing into Storage Devices (HPSD)

**Données :** Placement(VM, PM)

**Résultat :** Placement(VM, PM)

**initialization :**

*Ordonner l'ensemble* VM par ordre décroissant selon le rapport $\left(\frac{rate_{rnd}}{rate_{wrt}}\right)$

$HDD_{all} \leftarrow D_{k_j}$, *où* $D_{k_j}$ *est de type* HDD, $D_{k_j} \in PM_j$ *et* $PM_j \in PM$

$SSD_{all} \leftarrow D_{k_j}$, *où* $D_{k_j}$ *est de type* SSD, $D_{k_j} \in PM_j$ *et* $PM_j \in PM$

*Ordonner* $HDD_{all}$ par ordre décroissant selon le rapport $(capacity \cdot IOPS)$

*Ordonner* $SDD_{all}$ par ordre décroissant selon le rapport $(capacity \cdot IOPS)$

**packIntoSSD :**

**pour** *chaque* $D_{k_j}$ *dans* $SSD_{all}$ **faire**

  **pour** *chaque* $VM_i$ *dans* VM **faire**

    **si** $PM_j$ *peut héberger à* $VM_i$ ***et*** $D_{k_j}$ *peut stocker à* $VM_i$ **alors**

      $D_{current} \leftarrow$ *périphérique de stockage actuel de* $VM_i$

      **si** $Cost(VM_i, D_{k_j}) < Cost(VM_i, D_{current})$ **alors**

        *Mettre à jour* Placement *avec* $(VM_i, PM_j, D_{k_j})$

        *enlever* $VM_i$ *de l'ensemble* VM

      **fin**

    **fin**

  **fin**

**fin**

**packIntoHDD :**

**pour** *chaque* $D_{k_j}$ *dans* $HDD_{all}$ **faire**

  **pour** *chaque* $VM_i$ *dans* VM **faire**

    **si** $PM_j$ *peut héberger* $VM_i$ ***et*** $D_{k_j}$ *peut stocker* $VM_i$ **alors**

      $D_{current} \leftarrow$ *périphérique de stockage actuel de* $VM_i$

      **si** $Cost(VM_i, D_{k_j}) < Cost(VM_i, D_{current})$ **alors**

        *Mettre à jour* Placement *avec* $(VM_i, PM_j, D_{k_j})$

        *enlever* $VM_i$ *de l'ensemble* VM

      **fin**

    **fin**

  **fin**

**fin**

**retourner** Placement

Mise à part l'étape d'initialisation, cette approche est composée de deux fonctions principales : **packIntoSSD** et **packIntoHDD**. Les premières fonctions parcourent la liste des SSD et la liste des VM (lignes numéro 8 et 9), puis vérifient les contraintes du programme linéaire (ligne numéro 10). Si toutes les contraintes sont vérifiées, nous comparons le coût actuelle de placement de la VM avec le coût d'un déploiement sur SSD (ligne numéro 12). Si le nouveau placement est moins coûteux que le placement actuel, alors la VM est déplacée vers le nouveau périphérique (ligne numéro 13) puis retirée de la liste des VM à migrer (ligne numéro 14). La fonction packIntoHDD comporte les mêmes étapes que la fonction packIntoSSD, mais appliquées sur la liste des HDD.

Pour l'étude de sa complexité, cette approche peut être divisée en trois parties. La première partie concerne l'évaluation et sa complexité dépend des opérations de tri et donc varie en fonction du nombre de VM et de périphériques de stockage. La deuxième et la troisième parties correspondent aux fonctions packIntoSSD et packIntoHDD. Dans chacune de ces fonctions, nous parcourons l'ensemble de périphériques de stockage du type correspondant ainsi que la liste des VM. Dans le cas d'un centre de données avec $n$ VM, et $k$ périphériques de stockage (HDD + SSD), ces fonctions ont une complexité de l'ordre de $\mathcal{O}(n \cdot k)$.

***2) HPPM : Heuristic with Packing into Physical Machine*** : l'approche HPPM applique le principe de la consolidation au niveau des PM. Le pseudo-code 4 décrit le principe de l'approche HPPM.

Le principe de l'approche HPPM est de regrouper le maximum de VM en un minimum de PM. Au sein d'une même PM, les SSD sont favorisés pour les charges d'E/S avec un fort taux de lecture aléatoire. Toujours dans le but de favoriser les SSD pour les VM avec un fort taux de lecture aléatoire (et les HDD pour les VM avec un fort taux d'écritures séquentielles), nous trions les VM selon leur taux de lectures aléatoires (ligne numéro 2). Nous initialisons deux variables *head* et *tail* (lignes numéros 3 et 4) qui servent à parcourir la liste des VM par le début et par la fin respectivement.

Pour appliquer le principe de consolidation au niveau des PM, nous parcourons la liste des PM (ligne numéro 6). Les périphériques de stockage associés à chaque PM sont par la suite parcourus (ligne numéro 7) afin de distinguer leur type (lignes numéro 8 et 20). Si le périphérique de stockage est de type SSD, nous sélectionnons la VM qui se trouve au début de la liste de VM car elle présente le plus haut taux de lectures aléatoires (ligne numéro 10). Dans le cas d'un périphérique de stockage de type HDD, la VM se trouvant à la fin de la liste est choisie (ligne numéro 22) car elle présente le plus faible taux de lecture aléatoire (i.e. le plus haut taux d'écritures séquentielles). Ensuite, nous vérifions les contraintes d'allocation de la PM et le périphérique de stockage (lignes numéro 11 et 23) avant de comparer le coût actuel et le nouveau coût en cas d'allocation (lignes

**Algorithme 4 :** Heuristic with Packing into Physical Machine (HPPM)

**Données :** Placement(VM, PM)
**Résultat :** Placement(VM, PM)

**initialization :**
*Ordonner l'ensemble* VM par ordre décroissant selon le rapport $(\frac{rate_{rnd}}{rate_{wrt}})$
$head \leftarrow 0$
$tail \leftarrow n$, *où* $n$ est le nombre de VM
**packIntoPM :**
**pour** *chaque* $PM_j$ *dans* PM **faire**
  **pour** *chaque* $D_{k_j}$ *dans* $PM_j$ **faire**
    **si** $D_{k_j}$ *est de type* SSD **alors**
      **tant que** $head < tail$ **faire**
        $VM_{current} \leftarrow VM_{head}$
        **si** $PM_j$ *peut héberger* $VM_{current}$ ***et*** $D_{k_j}$ *peut stocker* $VM_{current}$ **alors**
          $D_{current} \leftarrow$ *périphérique de stockage actuel de* $VM_{current}$
          **si** $Cost(VM_{current}, D_{k_j}) < Cost(VM_{current}, D_{current})$ **alors**
            *Mettre à jour* Placement *avec* ($VM_{current}$, $PM_j$, $D_{k_j}$)
            $head \leftarrow head + 1$
          **fin**
        **fin**
      **fin**
    **sinon**
      **si** $D_{k_j}$ *est de type* HDD **alors**
        **tant que** $tail > head$ **faire**
          $VM_{current} \leftarrow VM_{tail}$
          **si** $PM_j$ *peut héberger* $VM_{current}$ ***et*** $D_{k_j}$ *peut stocker* $VM_{current}$ **alors**
            $D_{current} \leftarrow$ *périphérique de stockage actuel de* $VM_{current}$
            **si** $Cost(VM_{current}, D_{k_j}) < Cost(VM_{current}, D_{current})$ **alors**
              *Mettre à jour* Placement *avec* ($VM_{current}$, $PM_j$, $D_{k_j}$)
              $tail \leftarrow tail - 1$
            **fin**
          **fin**
        **fin**
      **fin**
  **fin**
**fin**
**retourner** Placement

numéro 13 et 25). Le plan de placement de VM est mis à jour avec le nouveau placement de la VM (lignes numéro 14 et 26) et également les variables *head* et *tail* (lignes numéros 15 et 27).

En terme de complexité en temps de calcul, l'approche HPPM peut être divisée en deux parties. La complexité de la première partie concerne la fonction d'initialisation qui dépend de l'opération de tri de la liste des VM. La deuxième partie concerne la fonction *packIntoPM*. La complexité de cette fonction dépend du nombre de PM, de VM, et des périphériques de stockage $\mathcal{O}(n \cdot m \cdot l)$.

La prochaine partie décrit la méthode et les résultats d'évaluation des approches présentées.

## 7.4 ÉVALUATION DE NOS APPROCHES

Cette partie présente l'évaluation des approches de placement de VM proposées dans ce chapitre. Nous avons évalué ces approches en deux parties.

La première partie présente l'évaluation de la méthode exacte ainsi que les heuristiques en utilisant des petites instances (i.e. petit de nombre de VM et de PM). Cette partie a pour objectif d'évaluer les performances des approches de placement approchées, en les comparant à la méthode exacte. Nous montrons également les limites de la méthode exacte avec des petites instances (i.e. nombre de VM et de PM). Nous analysons la précision des méthodes approchées en comparant leur résultats avec celles de la méthode exacte. Comme nous l'avons précisé auparavant, le temps d'exécution de cette approche dépend du nombre de périphériques de stockage et le nombre total de VM. Afin de définir les limites de cette méthode exacte, nous fixons le nombre total de périphériques de stockage et nous faisons varier le nombre de VM hébergées dans le centre de données. Les résultats obtenus sont présenté dans la partie 7.4.4.

La seconde partie évalue les approches heuristiques avec des grandes instances. Cette partie a pour objectif d'évaluer les performances des approches heuristiques avec des grandes instances (i.e. nombre de VM et de PM). Nous présentons une étude comparative entre nos approches de placement de VM, et un exemple d'une approche proposée dans [26] et implantée dans *CloudSim*. Pour l'évaluation des approches heuristiques, nous fixons le nombre de VM et PM et nous faisons varier le seuil d'utilisation pour la détection des PM sur-utilisées. Cela nous permet d'analyser l'agressivité des approches de placement pour chaque seuil d'utilisation (i.e. impact sur les pénalités, migration, et la consommation énergétique). Les résultats de cette partie d'évaluation sont présentés dans la section 7.4.5

Avant de présenter les différents résultats issus de chaque partie d'évaluation, nous allons d'abord décrire la méthode d'évaluation suivie, et la plateforme expérimentale utilisée tout au long de l'évaluation.

Aussi, nous allons d'abord présenter les différents paramètres utilisée dans le placement de VM (voir pseudo-code 1).

### 7.4.1 *Détection de sur-utilisation et sous-utilisation des PM*

Pour la détection des PM sous-utilisées, nous avons adapté l'algorithme déjà proposé dans *CloudSim* en intégrant l'utilisation des périphériques de stockage. Nous avons choisi d'utiliser l'algorithme de détection de sur-utilisation avec un seuil d'utilisation statique (voir partie 7.2.1.1). Cet algorithme de détection est simple et permet d'isoler l'impact des algorithmes de placement, de celui des autres politiques (i.e. détection de sur-utilisation et sélection de VM).

### 7.4.2 *Sélection des VM à migrer*

La sélection des VM à migrer depuis les PM sous-utilisées représente une étape primordiale dans le placement de VM (voir sou-partie 7.2.2). Pour évaluer nos approches, nous avons choisi d'utiliser deux politiques de sélection de VM : 1) temps minimum de migration, 2) minimum d'utilisation. Ces politiques ont été choisies pour leur simplicité d'adaptation aux paramètres du stockage. Nous avons intégré les caractéristiques de stockage et des E/S des VM dans la politique du temps minimum de migration (voir partie 7.2.2.2). Avec cette adaptation, le temps de migration d'une VM comprend la taille de son image et son taux d'arrivée des requêtes. En effet, le temps de migration d'une VM ne dépend pas seulement de la taille de son stockage mais aussi de l'activité d'E/S de la VM exprimée en taux d'arrivée des requêtes [205].

$$\begin{aligned} &\forall\, VM_i \in VM, \forall\, PM_j \in PM, \forall\, D_{k_j} \in PM_j, \\ &T_{mig}(VM_i, D_{k_j}) = MAX\Big(\frac{vm_{size_i}}{pm_{net_j}}, \frac{vm_{size_i}}{perf(D_{k_j})}\Big) \cdot \frac{1}{rate_{IO_i}} \end{aligned} \tag{48}$$

où $vm_{size_i}$ est la taille d'image disque de la VM $VM_i$, $pm_{net_j}$ est la bande passante allouée à la PM $PM_j$, $perf(D_{k_j})$ est une fonction donnant les performances du périphérique de stockage $D_{k_j}$ attaché à la PM $PM_j$, et $rate_{IO_i}$ et le taux d'arrivée des requêtes de la VM $VM_i$.

La deuxième politique de sélection choisie est celle du minimum d'utilisation CPU (voir partie 7.2.2.3). Nous avons également adapté cette politique pour prendre en compte les caractéristiques E/S des VM. En plus de l'utilisation CPU, notre adaptation considère l'utilisation des périphériques de stockage pour chaque $VM_i$ :

$$\forall\, VM_i \in VM, perf_{ratio}(VM_i) = \frac{rate_{IO_i}}{vm_{IOPS_i}} \tag{49}$$

où $perf_{ratio}$ est une fonction qui calcule le ratio des performances de stockage de la VM $VM_i$, $rate_{IO_i}$ est le taux d'arrivée des requêtes d'E/S de la VM $VM_i$, et $vm_{IOPS_i}$ est le nombre d'IOPS demandées par la VM $VM_i$.

La sélection des VM se fait sur la base des deux utilisations (i.e. CPU et stockage) :

$$\forall\, VM_i \in VM, Min\Big(\frac{1}{CPU_{ratio}(VM_i)} \cdot \frac{1}{perf_{ratio}(VM_i)}\Big)$$

où Min est une fonction choisissant la VM qui présente le plus petit produit entre son ratio de capacité CPU et son ratio de performances du stockage.

La prochaine partie donne la configuration utilisée lors de l'évaluation des approches du placement de VM.

### 7.4.3 *Configuration des simulations*

Pour la configuration des simulations, nous avons repris les configurations utilisées dans [26].

| Paramètre | Valeur |
|---:|---|
| Types | 4 |
| Nombre de CPU | 1 |
| Performance CPU (MIPS) | [250, 500, 750, 1000] |
| Capacité RAM (Mo) | 128 |
| Capacité réseau (Mbits/s) | 100000 |
| Taille d'image (Mo) | 1024 |
| IOPS demandé | [10000, 20000, 30000, 40000] |
| Nombre de *Cloudlet* | 1 |
| Taille de chaque *Cloudlets* (MI) | 150000 |

TABLE 10 : paramètres des VM

| Paramètre | Valeur |
|---:|---|
| Types de PM | 3 |
| Nombre de CPU | 1 |
| Performance CPU (MIPS) | [1000, 2000, 3000] |
| Capacité RAM (Mo) | 8192 |
| Capacité réseau (Mbits/s) | 1000000 |
| Système de stockage | 1 HDD + 1 SSD |

TABLE 11 : paramètres des PM

Concernant le système de stockage, nous avons simulé un système de stockage hybride, avec des SSD [58] et des HDD [171].

| Paramètre | Valeur |
|---|---|
| Modèle | Seagate ST1000DM003 Barracuda [171] |
| Capacité (Go) | 1000 |
| Prix $/Go ($) | 0.075 |

TABLE 12 : paramètres des HDD

| Paramètre | Valeur |
|---|---|
| Modèle | Samsung SSD 850 EVO PRO Series [58] |
| Capacité (Go) | 256 |
| Prix $/Go ($) | 0.354 |

TABLE 13 : paramètres des SSD

Pour les paramètres du modèle de coût, nous avons choisi de prendre le prix utilisé par Amazon pour les VM [173] avec une configuration semblable à celle utilisée dans les simulations de [26]. Pour le prix de l'énergie électrique, nous avons considéré le prix utilisé dans les centres de données d'*Amazon* en Californie [14].

| Paramètre | Valeur |
|---|---|
| Prix du service cloud ($/heure) | 0.012 |
| Prix de l'énergie ($/kWh) | 0.1763 |

TABLE 14 : paramètres du modèle de coût

Les prochaines parties présentent les résultats d'évaluation en commençant par ceux de la méthode exacte de recherche exhaustive.

### 7.4.4 *Résultats de l'évaluation avec des petites instances*

Dans cette partie, nous présentons les résultats de l'évaluation de nos approches en utilisant des petites instances (i.e. 20 PM, et de 5 à 50 PM). L'objectif de cette étape est de valider la précision des heuristiques en les comparant aux résultats de la méthode exacte. Les paramètres des simulations utilisés dans cette étape sont donnés par le tableau 15.

La prochaine partie présente les résultats de cette étape en terme de coût du placement de VM.

| Paramètre | Valeur |
|---|---|
| Nombre de PM | 20 |
| Périphériques de stockage par PM | 2 (HDD + SSD) |
| Nombre de VM | [5, . . . , 50] |
| Seuil d'utilisation Max. | 90% |

TABLE 15 : paramètres des simulations

#### 7.4.4.1 *Coût de placement*

Les figures 58 et 59 présentent le coût de placement en utilisant les différentes approches, normalisé sur le coût obtenu avec la méthode exacte.

La figure 58 montre le coût du plan de placement en fonction du nombre de VM avec l'algorithme de sélection MMT.

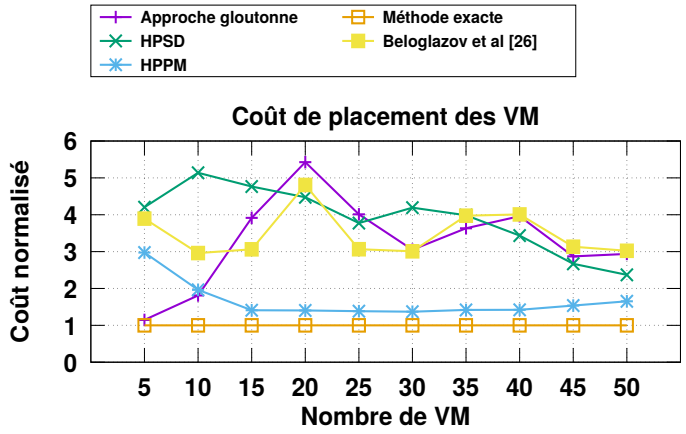


FIGURE 58 : coût de placement avec la politique MMT

Comme le montre la figure 58, l'approche HPPM présente le meilleur coût comparé aux autres approches. L'approche HPSD conduit à un écart allant jusqu'à six fois le coût optimal. La méthode gloutonne et celle de Beloglazov et *al.* présentent des tendances moins stables que celles de la méthode exacte, de l'approche HPPM, et de l'approche HPSD.

La figure 59 montre le coût de placement en utilisant la politique de sélection MU avec la prise en compte des IOPS. Les approches HPPM et HPSD donnent des solutions proches de la solution optimale entre un et deux fois le coûts optimal dans les pires cas, tandis que la méthode gloutonne donne un coût décroissant lorsque le nombre de VM augmente. L'approche de placement de Beloglazov et *al*. [26] donne des résultats éloignés du coût optimal (i.e. allant jusqu'à six fois le coût optimal).

Des deux figures 58 et 59, nous pouvons constater que la politique de sélection a également un impact sur le choix du placement optimal de VM.

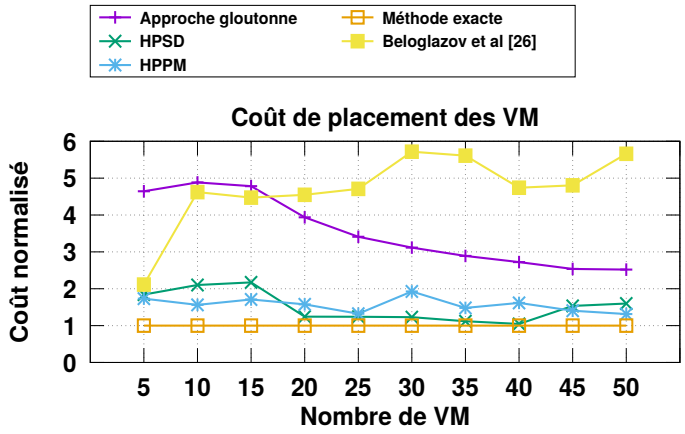


FIGURE 59 : coût de placement avec la politique MU

### 7.4.4.2 *Temps d'exécution*

Les deux figures 60 et 59 présentent le temps d'exécution normalisé par rapport au temps d'exécution de la méthode exacte, pour les différentes approches de placement. La figure 60 montre le temps d'exécution en utilisant la politique de sélection de VM MMT (voir partie 7.2.2.2), tandis que la figure 61 présente le temps d'exécution en utilisant la politique de sélection MU (voir partie 7.2.2.3) avec la prise en compte des IOPS.

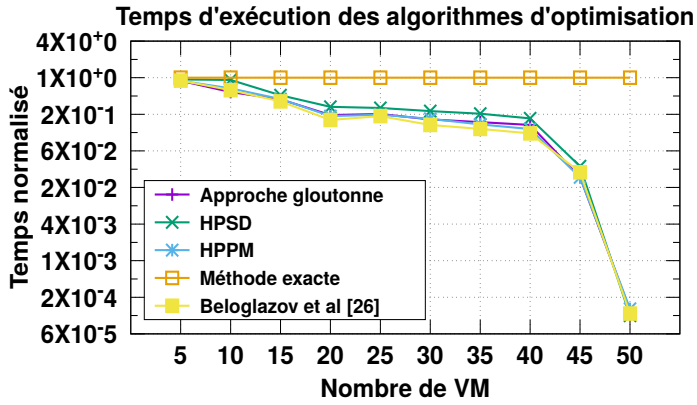


FIGURE 60 : temps d'exécution des différentes approches

Les deux figures montrent que le temps d'exécution des heuristiques ont une tendance décroissante par rapport à la méthode exacte. Par exemple, dans le cas de la politique de sélection MU, la méthode exacte nécessite ~15 minutes de calcul avec seulement 50 VM, tandis que HPSD s'exécute en 0.04 secondes.

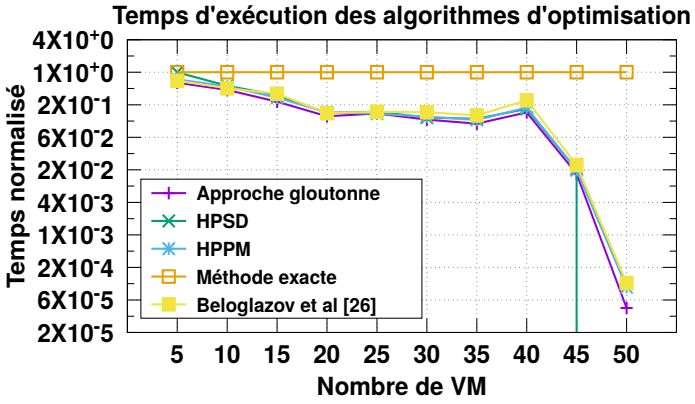


FIGURE 61 : temps d'exécution des différentes approches

### 7.4.5 *Résultats de l'évaluation avec des grandes instances*

Cette partie présente les résultats d'évaluation des algorithmes approchés de placement de VM. Dans ce cas d'étude, nous utilisons des instances de grandes tailles comparées à la partie 7.4.4.

Dans le but de comparer nos contributions avec un travail de reférence dans l'état de l'art, nous utilisons les configurations des simulations effectuées dans [26] et présentées dans le tableau 16.

| Paramètre | Valeur |
|---|---|
| Nombre de PM | 100 |
| Périphériques de stockage par PM | 2 (HDD + SSD) |
| Nombre de VM | 290 |

TABLE 16 : paramètres des simulations

Nous faisons varier le seuil d'utilisation afin d'analyser son impact sur les différents approches, et ainsi évaluer ses effets sur le coût de placement des VM.

Nous prenons en compte deux métriques de comparaison des approches, à savoir le coût de placement et le temps d'exécution.

#### 7.4.5.1 *Coût de placement*

Les figures 62 et 63 présentent le coût de placement en fonction du seuil maximal d'utilisation. La figure 62 correspond au coût de placement en utilisant la politique de sélection MMT (voir partie 7.2.2.2), tandis que la figure 63 correspond au coût de placement en utilisant la politique MU (voir partie 7.2.2.3).

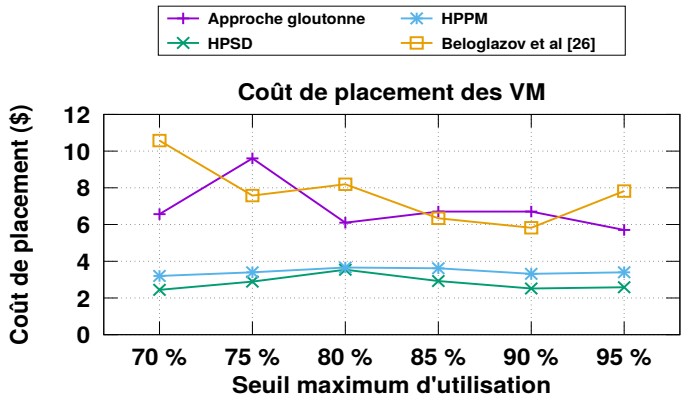


FIGURE 62 : coût de placement avec la politique de sélection MMT

Comme le montre la figure 62, les deux heuristiques sont insensibles au changement du seuil maximal d'utilisation. Ce phénomène est logique puisque la politique MMT ne dépend pas de l'utilisation CPU.

L'approche de placement de Beloglazov et *al.* [26] présente une tendance décroissante avec l'augmentation du seuil d'utilisation. Cela est du au nombre de PM actives qui diminue en augmentant le seuil maximal d'utilisation, ce qui diminue la consommation énergétique totale. La méthode gloutonne présente une trajectoire instable à cause de la non prise en compte des seuils d'utilisation CPU et des performances de stockage.

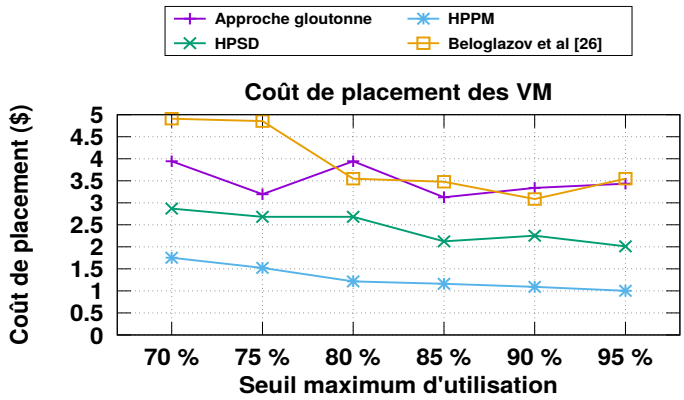


FIGURE 63 : coût de placement avec la politique de sélection MU

La figure 63 montre que le coût de placement diminue en augmentant le seuil d'utilisation avec la politique MU. Ceci s'explique par le fait que cette politique est basée sur l'utilisation CPU. Mise à part la méthode gloutonne, toutes les approches ont une tendance décroissante en fonction du seuil d'utilisation. Cette tendance est expliquée par la diminution de la consommation énergétique causée par la réduction du nombre de PM actives. L'approche HPPM présente les meilleurs résultats avec un coût de placement jusqu'à trois fois moins élevé que celui de l'approche proposée dans [26].

### 7.4.5.2 *Temps d'exécution*

Les deux figures 64 et 63 présentent le temps d'exécution des approches de placement de VM en fonction du seuil d'utilisation et des politiques de sélection MMT et MU. La figure 64 montre que le temps d'exécution des approches de pla-

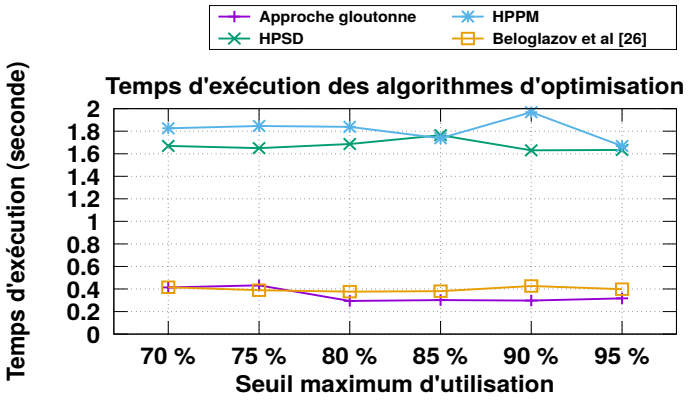


FIGURE 64 : temps d'exécution avec la politique de sélection MMT

cement ne dépend pas du seuil maximal d'utilisation. En effet, la complexité en temps de calcul de toutes les approches dépend du nombre de périphériques de stockage et non du seuil d'utilisation (voir partie 7.4.4.2). Le second constat concerne l'écart entre le temps d'exécution des heuristiques HPPM et HPSD, et celui de la méthode gloutonne et de celle présentée par Béloglazov et *al.* [26]. Les approches HPPM et HPSD parcourent et trient toutes les VM, et les périphériques de stockage, ce qui représentent des tâches chronophages. Comme constaté dans

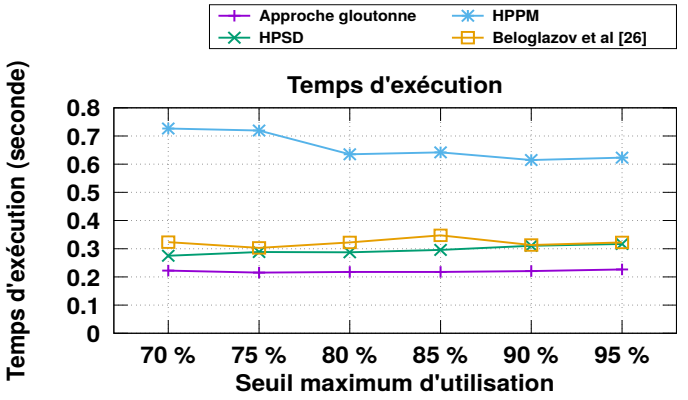


FIGURE 65 : temps d'exécution avec la politique de sélection MU

le cas de la politique MMT, la figure 65 montre que le temps d'exécution des approches de placement ne dépend pas du seuil maximal d'utilisation. À cause de sa dépendance à l'utilisation CPU et aux des performances du stockage, l'heuristique HPPM a un temps de réponse plus long que les autres algorithmes pour la politique MU. En effet, l'utilisation des VM est vérifiée au niveau des PM et

des périphériques de stockage, ce qui rajoute un temps de calcul comparé à la politique MMT. Nous constatons également que la complexité en temps de calcul de toutes les approches ne dépend pas du seuil d'utilisation.

Après avoir présenté et évalué nos contributions sur le placement de VM, la prochaine partie conclut ce chapitre.

## 7.5 CONCLUSION

L'étape *Plan* de notre étude basée sur le modèle MAPE-K, est conduite en deux étapes. Dans la première, nous proposons une mise en œuvre de notre modèle de coût pour le stockage des VM. À l'issue de cette étape, nous disposons d'un ensemble d'outils pour entamer la deuxième phase consistant au placement de VM avec la prise en compte des systèmes de stockage et des E/S. Nous avons présenté dans ce chapitre nos approches de placement de VM.

Les approches de placement proposées prennent en compte les coûts de stockage et d'exécution des charges d'E/S des VM.

Nous avons proposé une méthode exacte et trois heuristiques pour le placement de VM prenant en compte les coûts de stockage des VM. La méthode exacte énumère toutes les solutions possible puis sélectionne le placement de VM le moins coûteux. La première heuristique est une approche gloutonne qui cherche à optimiser un placement de VM déjà mis en place. La deuxième et la troisième heuristiques visent 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).

Les différentes approches ont été implantées et évaluées dans le simulateur *CloudSim*. La méthode exacte a été utilisée comme une référence afin d'analyser la l'exactitude et le temps d'exécution des approches approchées.

Dans les meilleurs cas, nos heuristiques ont donné des solutions jusqu'à six fois moins coûteuses que les approches de placement présentées dans [26]. En terme de précision, nos heuristiques ont conduit à des solutions proches du placement optimal, tandis que l'approche de Beloglazov et *al.* [26] donne des placements de VM beaucoup plus coûteux que la solution optimale. En terme de temps d'exécution, nos approches donnent des solutions en un temps ne dépassant pas deux secondes dans les pires cas.

# Troisième partie

# CONCLUSION

# Chapitre 8

# CONCLUSION

Minimiser les coûts d'utilisation des infrastructures représente un grand défi pour les fournisseurs de services *cloud* en général et particulièrement pour le *cloud* IaaS. Lorsqu'il est question de modéliser et d'optimiser ces coûts, différents composants du système informatique entrent en jeux.

Le stockage représente un élément indissociable de ces systèmes. Malheureusement, les travaux de l'état de l'art proposant des approches de modélisation et d'optimisation des coûts des centres de données ne considèrent pas les aspects liés aux E/S disques et aux systèmes de stockage [22, 24, 26, 54, 81, 91, 107, 111, 177, 181]. De plus, l'arrivée des nouvelles technologies de stockage telles que les disques SSD rend cette tache plus complexe d'une part, et d'autre part offre plus d'opportunités pour amortir les coûts des centres de données.

Dans cette thèse, nous proposons un ensemble de contributions qui visent à optimiser l'utilisation des infrastructures d'un *cloud* IaaS. Les modèles proposés prennent en compte les E/S des VM ainsi que les systèmes de stockage hybrides. Nous avons proposé une approche suivant le modèle MAPE-K pour le placement autonomique des VM sur un système de stockage hybride. Ce chapitre rappelle l'ensemble de nos contributions, ainsi que les perspectives de travaux futurs de cette thèse.

**Sommaire**

## 8.1 CONTRIBUTIONS

Ce travail a donné lieu à différentes contributions organisées selon le modèle MAPE-K.

### 8.1.1 *Monitor*

Notre première contribution consiste en la proposition d'outils de *monitoring* et de caractérisation des charges d'E/S des VM.

Notre contribution sur le *monitoring* est un outil de trace d'E/S des VM. Cet outil a pour objectif de tracer les accès des VM au système de stockage sur différents niveaux de la pile d'E/S. Cet outil a été conçu et développé en deux versions.

La première version combine différents outils et bibliothèques disponibles sous Linux et permet de tracer les E/S des VM sur les niveaux suivants :

- hyperviseur des VM en utilisant la bibliothèque *libvirt* [162] ;
- système de fichiers virtuel (VFS) en utilisant l'outil *strace* [8] ;
- système de fichiers physique (qui est plutôt un niveau de liaison) en utilisant la bibliothèque *libext2fs* [184] ;
- couche bloc d'E/S en utilisant l'outil *blktrace* [17].

La deuxième version a été développée sous la forme d'un module noyau Linux dans le but de rendre l'outil plus générique et indépendant des outils précédemment utilisés. Cette version opère sur deux niveaux : 1) niveau VFS, et 2) niveau bloc d'E/S.

L'évaluation des deux versions du traceur a montrée que ces outils sont peu intrusifs (e.g. un surcoût de ~2%).

Nous avons également proposé un modèle de caractérisation des charges d'E/S des VM. Ce modèle prend en entrée les traces obtenues à l'aide de l'outil de *monitoring* et produit un ensemble de paramètres caractérisant la charge d'E/S de chaque VM.

Ces outils et modèles nous ont permis d'accomplir l'étape *Monitor* du modèle MAPE-K, et ainsi produire les éléments nécessaires pour la gestion autonomique de placement de VM.

### 8.1.2 *Analyze*

Pour l'étape *Anlayze*, nous avons proposé un modèle de coût pour le stockage et l'exécution des charges d'E/S des VM. Le modèle proposé prend en considération deux classes de stockage (disque dur magnétique et disque SSD). Notre modèle

considère également des contraintes liées à l'environnement du *cloud* (SLA et migration de VM).

L'évaluation de la partie traitant les coûts de la consommation énergétique a montré un écart de 0.04% dans le meilleur cas, et de 15% dans le pire cas par rapport à la consommation énergétique réelle.

une précision de notre modèle entre 0.04% dans le meilleur cas, et 15% dans le pire cas.

Notre modèle de coût constitue la fonction objectif utilisée par nos approches de placement de VM, et qui sont incorporées dans l'étape *Plan* du modèle MAPE-K.

### 8.1.3 *Plan*

Nous proposons deux contributions représentant l'étape *Plan*. La première est une extension du simulateur de *cloud CloudSim*. Le deuxième est la conception et l'implantation de plusieurs approches de placement de VM en prenant en compte les E/S et le stockage des VM.

***1) intégration et simulation des E/S et des systèmes de stockage hybrides dans le cloud***

Après avoir modélisé les coûts de stockage et d'exécution des E/S des VM dans l'étape *Analyze*, nous avons proposé une mise en œuvre de cette modélisation dans le simulateur *CloudSim*. Pour cela, nous avons étendu ce dernier par la prise en compte des charges d'E/S et du stockage hybride. Pour ce faire, nous avons intégré des modèles pour le temps de calcul, l'utilisation CPU, et la consommation énergétique liés à l'exécution des E/S des VM.

Pour la prédiction de l'utilisation CPU due à l'exécution des E/S, nous avons proposé un modèle de régression basé sur une approche d'apprentissage automatique. Son évaluation a montré une précision de l'ordre de 0.98 exprimée en coefficient de corrélation $R^2$.

L'évaluation de notre extension a révélé que l'exécution des E/S et les systèmes de stockage peuvent consommer jusqu'à 25% de la consommation totale d'un centre de données. Ces résultats confirment l'importance de la prise en compte des E/S dans les approches du placement de VM.

***2) optimisation des coûts en considérant le stockage et les E/S des VM***

Pour la deuxième contribution de l'étape *Plan*, nous avons proposé quatre approches de placement de VM prenant en compte le stockage et les E/S des VM : une approche exacte et trois heuristiques.

Premièrement, nous avons proposé une approche exacte qui énumère tous les plans de placement de VM puis choisit le moins coûteux. La première heuristique est une approche gloutonne cherchant à optimiser un plans de placement déjà en

place. La deuxième et la troisième heuristiques visent à exploiter les avantages du système de stockage hybride dans le but de minimiser les coûts en fonctions des caractéristiques des charges d'E/S des VM.

Les approches proposées ont été implantées et évaluées dans le simulateur *CloudSim*. L'algorithme exact permet de générer des résultats de référence afin d'analyser l'exactitude et le temps d'exécution des heuristiques. Nos approches ont été comparées à une approche qui ne prend pas en compte les E/S et le stockage des VM.

En terme de precision, nos heuristiques ont donné des solutions proches du plan de placement optimal, tandis que la méthode d'optimisation sans prise en compte des coûts de stockage donne des plans de placement de VM jusqu'à six fois plus coûteux que la solution optimale. En terme de temps d'exécution, nos approches donnent des solutions en un temps ne dépassant pas deux secondes dans les pires cas. Ceci est important pour la mise en œuvre *en ligne* de ces solutions.

## 8.2 PERSPECTIVES

Notre travail s'est appliqué au problème du placement de VM dans un système de stockage hybride. Nous envisageons de profiter des connaissances acquises lors de la réalisation de ce travail afin de traiter les limites de chacune de nos contributions. Cette partie résume nos perspectives pour les travaux futurs de cette thèse.

### 8.2.1 *Monitor*

Les outils et modèles proposés dans l'étape *Monitor* ne permettent pas la prédiction des charges d'E/S des VM. En effet, nous considérons actuellement qu'une VM ne peut avoir qu'un seul profil d'E/S, et qui celui-ci 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 VM. 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 [104], algorithmes d'apprentissage automatique [131, 160], réseaux de neurones [124]).

### 8.2.2 *Analyze*

Notre modèle de coût actuel a été conçu pour les périphériques des stockage sans prise en compte des caches du système d'exploitation. D'autre part, l'interférence entre les VM n'a pas été considérée.

Nous envisageons d'étudier l'intégration d'autres contraintes dans notre modèle (e.g. mécanismes de cache, interférences entre VM, etc), et de modéliser l'impact des E/S sur l'ensemble du système (CPU et RAM).

### 8.2.3 *Plan*

Pour le placement de VM, 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 trois heuristiques dont une gloutonne et deux basées sur l'algorithme *Best Fit Decreasing* pour la minimisation des coûts. Nous envisageons d'élargir notre étude sur d'autres approches multi-objectifs (e.g. algorithmes génétiques, recuit simulé, colonies de fourmis) pour la minimisation des coûts et la maximisation des performances (i.e. équilibrage de charge).

### 8.2.4 *Execute*

L'étapes *Execute* n'a pas été intégrée dans nos approches de placement autonomique de VM.

Afin de compléter notre version du modèle MAPE-K, nous allons étudier l'étape *Execute*. Cette étape doit se charger d'appliquer les plans de placement de VM sur l'infrastructure du *cloud*. Pour ce faire, une étude sur l'ordonnancement des migrations de VM doit être menée [180].

# Quatrième partie

# ANNEXE

# Annexe A

# NOTATIONS

## A.1 NOTATION UTILISÉE DANS LE CHAPITRE 4

Le tableau 17 donne la notation utilisée lors de la caractérisation des charges d'E/S des VM présentée dans la partie 4.4 du chapitre 4.

| Notation | Description | Unité |
|---|---|---|
| $W$ | Charge d'E/S d'une VM | - |
| $\mathtt{Req}$ | Requête donnée d'E/S | - |
| $\mathtt{req}_{\mathtt{type}}$ | Type de la requête d'E/S (i.e. lecture ou écriture) | $\{\mathtt{read}, \mathtt{write}\}$ |
| $\mathtt{req}_{\mathtt{addr}}$ | Adresse du début de la requête d'E/S | Octets |
| $\mathtt{req}_{\mathtt{size}}$ | Taille de la requêtes d'E/S | Octets |
| $\mathtt{bloc}_{\mathtt{size}}$ | Taille d'un bloc de données | Octets |
| $\mathtt{rate}_{\mathtt{IO}}$ | Taux d'arrivée des requêtes d'E/S | Requêtes/seconde |
| $\mathtt{T}_{\mathtt{mon}}$ | Temps de *monitoring* d es E/S | Secondes |
| $\mathtt{data}_{\mathtt{amount}}$ | Volume total de données généré par une VM donnée | Octets |
| $\mathtt{Hist}_{\mathtt{sizes}}$ | Histogramme regroupant toutes les tailles des requêtes d'E/S | Octets |
| $\mathtt{rate}_{\mathtt{rnd}}$ | Taux des requêtes aléatoires | $[0, 1]$ |
| $\mathtt{rate}_{\mathtt{seq}}$ | Taux des requêtes séquentielles (égale à 1-$\mathtt{rate}_{\mathtt{rnd}}$) | $[0, 1]$ |
| $\mathtt{rate}_{wrt}$ | Taux des requêtes d'écriture | $[0, 1]$ |
| $\mathtt{rate}_{\mathtt{read}}$ | Taux des requêtes de lecture (égale à 1-$\mathtt{rate}_{wrt}$) | $[0, 1]$ |

TABLE 17 : notation des paramètres pour la caractérisation des charges d'E/S

## A.2 NOTATION UTILISÉE DANS LE CHAPITRE 5

Les tableaux 18, 19, 20, et 21 donnent la notation utilisée dans la modélisation du coût de stockage des VM présentée dans le chapitre 5.

| Fonction | Description |
|---|---|
| $Cost$ | Coût total de stockage et d'exécution des E/S d'une VM |
| $Cost_{exe}$ | Coût d'exécution des E/S d'une VM sur un périphérique de stockage |
| $Cost_{mig}$ | Coût de migration d'une VM d'un périphérique de stockage vers un autre |
| $Cost_{e\_tco}$ | Coût total de cycle de vie |
| $Cost_{e\_wo}$ | Coût d'usure des périphériques de stockage |
| $Cost_{e\_egy}$ | Coût énergétique d'exécution des E/S d'une VM |
| $Cost_{per\_mb}$ | Coût de l'usure par mégaoctet écrit sur un SSD |
| $Cost_{str-stp}$ | Coût de l'usure par cycle *start-stop* d'un HDD |
| $Cost_{e\_pen}$ | Coût de pénalité liée à l'exécution des E/S des VM |
| $Cost_{m\_egy}$ | Coût énergétique de migration d'une VM |
| $Cost_{m\_wo}$ | Coût de l'usure liée à la migration d'une VM |
| $Bill$ | Facture d'utilisation d'un périphérique de stockage par une VM |
| $E$ | Énergie consommée par un composant contribuant à l'exécution des E/S d'une VM |
| $write_{tot}$ | Quantité totale de données écrites par une VM |
| $vm_{DTR}$ | Débit de transfert de données d'une VM |
| $req_{DTR}$ | Débit de transfert de données demandé par une VM |
| $dev_{DTR}$ | Débit de transfert de données d'un périphérique de stockage |
| $dev_{IOPS}$ | IOPS du périphérique de stockage |
| $NB_{str-stp}$ | Nombre de cycles *start-stop* durant une période donnée |
| $T_{mig}$ | Temps nécessaire pour migrer l'image de la VM |

TABLE 18 : fonctions du modèle de coût

| Notation | Description | Unité |
|---|---|---|
| VM | Variable représentant une VM donnée | - |
| D | Variable représentant un périphérique de stockage donnée | {HDD, SSD} |
| NRC | Constante représentant les coûts non-récurrents | € |
| $NB_{wrt}$ | Nombre de requêtes d'écriture exécutées par une VM données | Nombre entier |
| $rate_{i,j}$ | Taux de lecture/écriture des E/S séquentielles/aléatoires où $j \in \{read, wrt\}$ et $i \in \{seq, rnd\}$ | $[0, 1]$ |
| $rate_{IO}$ | Taux d'arrivée des requêtes d'E/S | Requêtes/seconde |
| $T_{mon}$ | Temps de *monitoring* d es E/S | Secondes |
| $req_{size}$ | Taille de la requêtes d'E/S | Octets |
| $IMG_{size}$ | Taille de l'image disque de la VM | Mo |
| $req_{IOPS}$ | IOPS demandé par la VM (spécifié dans le SLA) | IOPS |
| $req_{DTR}$ | Débit demandé par la VM (spécifié dans le SLA) | Mo/s |

Table 19 : notation des paramètres des VM

## A.3 NOTATION UTILISÉE DANS LE CHAPITRE 6

Les tableaux 22, et 23 présentent les notations utilisées dans le chapitre 6.

| Paramètre | Description | Unité |
|---|---|---|
| $P_{i,j}$ | Puissance moyenne durant l'exécution des E/S où $i \in \{seq, rnd\}$ et $j \in \{read, wrt\}$ | Watt |
| $P_{op}$,$P_{idl}$, $P_{stdb}$ | Puissance en *operating mode*, *idle mode*, et *standby mode* | Watt |
| $T_{i,j}$ | Temps moyen passé à l'exécution des E/S où $i \in \{seq, rnd\}$ et $j \in \{read, wrt\}$ | Watt |
| $T_{op}$, $T_{idl}$, $T_{stbb}$ | Temps passé respectivement en *operating mode*, *idle mode*, et *standby mode* | Secondes |
| $E_{spn}$ | Énergie consommée durant le passage du *standby mode* au *operating mode* pour HDD | Ws |
| $stg_{UP}$ | Prix unitaire du stockage (achat, maintenance, etc) | €/Go |
| $stg_{cap}$ | Capacité totale du périphérique de stockage | Go |
| $MAX_{wrt}$ | Quantité de donnée maximale à écrire sur le périphérique jusqu'à son usure | Go |
| $N_{str-stp}$ | Nombre de cycles *start-stop* jusqu'à l'usure du HDD | - |

TABLE 20 : notation des périphériques de stockage

| Paramètre | Description | Unité |
|---|---|---|
| $E_{UP}$ | Prix unitaire de l'énergie | €/KWh |
| $CS_{UP}$ | Prix unitaire du service *cloud* | €/Heure |
| $rate_{pen}$ | Taux de pénalité retranché de la facture du service *cloud* | $[0, 1]$ |

TABLE 21 : notations du service *cloud*

| Fonction | Description |
|---|---|
| $P$ | Puissance électrique en fonction de l'utilisation CPU |
| $cpu_{load}$ | Charge CPU en fonction des E/S et du périphérique de stockage |
| $h$ | Fonction charnière (*hinge loss function*) |

TABLE 22 : fonctions utilisées dans l'implantation

| Notation | Description | Unité |
|---|---|---|
| $u$ | Utilisation CPU | Pourcentage |
| $P_{max}$ | Puissance électrique maximale consommée à 100% d'utilisation CPU | Watt |
| $P_{ratio}$ | Ratio entre la puissance électrique à 100% et à 0% d'utilisation CPU | Watt |
| D | Périphérique de stockage | {HDD, SSD} |
| $W$ | Charge d'E/S d'une VM | - |
| $rate_{rnd}$ | Taux des requêtes aléatoires | $[0, 1]$ |
| $rate_{wrt}$ | Taux des requêtes d'écriture | $[0, 1]$ |
| $rate_{IO}$ | Taux d'arrivée des requêtes d'E/S | Requêtes/-seconde |
| $req_{size}$ | Taille de la requêtes d'E/S | Octets |

TABLE 23 : notation des paramètres utilisés dans la modélisation

## A.4 NOTATION UTILISÉE DANS LE CHAPITRE 7

Les tableaux 24, et 25 donnent la notation utilisée dans l'optimisation du placement de VM présentée dans le chapitre 7.

| Fonction | Description |
|---|---|
| $median$ | Fonction donnant la valeur médiane d'une variable statistique |
| $\widehat{g}$ | Fonction de prédiction de l'utilisation CPU |
| $T_{mig}$ | Temps de migration d'une VM |
| $RAM_u$ | Capacité mémoire utilisée par une VM |
| $CPU_{ratio}$ | Ratio entre la capacité CPU demandée par une VM, et celle actuellement utilisée |
| $CPU_{current}$ | Capacité CPU actuellement utilisée par une VM |
| $CPU_{requested}$ | Capacité CPU demandée par une VM |
| $perf$ | Performances d'un périphérique de stockage |
| $perf_{ratio}$ | Ratio entre les performances de stockage demandées par une VM, et celles actuellement utilisées |

TABLE 24 : fonctions utilisées dans l'optimisation du placement de VM

| Notation | Description | Unité |
|---|---|---|
| MAD | Écart médian absolu (*Median Absolute Deviation*) d'une variable statistique | - |
| $U_i$ | Utilisation CPU | Pourcentage |
| $T_u$ | Seuil supérieur d'utilisation CPU | Pourcentage |
| $s$ | Variable définissant l'agressivité d'un algorithme de consolidation | Nombre réel |
| IQR | Écart inter-quartiles (*InterQuartile Range*) d'une variable statistique | - |
| $Q_i$ | ième quartile d'une variable statistique | - |
| $t_m$ | Temps maximum de migration d'une VM | Seconde |
| | **VM** | |
| $vm_{CPU}$ | Capacité CPU demandée par une VM | Pourcentage |
| $vm_{RAM}$ | Capacité RAM demandée par une VM | Mo |
| $vm_{size}$ | Capacité de stockage demandée par une VM | Mo |
| $vm_{IOPS}$ | Performances de stockage demandées par une VM | IOPS |
| $W$ | Charge d'E/S d'une VM | - |
| $rate_{rnd}$ | Taux des requêtes aléatoires | $[0, 1]$ |
| $rate_{wrt}$ | Taux des requêtes d'écriture | $[0, 1]$ |
| $rate_{IO}$ | Taux d'arrivée des requêtes d'E/S | Requêtes/-seconde |
| $req_{size}$ | Taille de la requêtes d'E/S | Octets |
| $data_{amount}$ | Volume total de données généré par une VM donnée | Octets |
| | **PM** | |
| $pm_{CPU}$ | Capacité CPU d'une PM | Pourcentage |
| $pm_{RAM}$ | Capacité RAM d'une PM | Mo |
| $pm_{net}$ | Bande passante réseau allouée à une PM | Mbits/seconde |
| $d_{cap}$ | Capacité d'un périphérique de stockage | Go |
| $d_{IOPS}$ | Performances d'un périphériques de stockage | IOPS |

TABLE 25 : notation des paramètres utilisés dans l'optimisation du placement de VM

# Annexe B

# ALGORITHMES ET RÉSULTATS

## B.1 MONITORING DES E/S DES VM

### B.1.1 *Niveau hyperviseur*

**Algorithme 5 :** Traces au niveau Hyperviseur

**Données :** Hyp=$\{vm_1, vm_2, ..., vm_n\}$
**Résultat :** Traces=$\{t_1, t_2, ..., t_n\}$

```
1  initialization :
2  stats_E/S ← 0
3  HyperTrace (Hyp : hyperviseur)
   /* Parcourir les VM en cours d'exécution par l'hyperviseur        */
4  pour chaque Machine Virtuelle vm_j dans Hyp faire
5      pour chaque Disque Virtuel vd_k attaché à vm_j faire
           /* Récupérer les statistiques au niveau de l'hyperviseur  */
6          stats_E/S ← Libvirt.stats(vm_j, vd_k)
           /* Sauvegarder les nouvelles statistiques                 */
7          t_i ← {vm_j, vd_k, stats_E/S}
           /* Mettre à jour les traces                               */
8          Update(Traces(t_i))
9      fin
10 fin
11 retourner Traces
```

Ce pseudo code est une fonction appelée périodiquement. Elle prend en entrée le système de virtualisation avec un hyperviseur donné, et produit en sortie un ensemble de fichiers de traces, avec un fichier par VM.

À chaque appel de cette fonction, les statistique sur les E/S sont remises à zéro (ligne 2). Les VM présentes sur le système hôte ainsi que les disques virtuelles qui attachés sont parcourues (lignes 4 et 5 respectivement). Ensuite, *libvirt* est interrogée sur les statistiques d'E/S qui correspondent à chaque disque virtuel et

les nouvelles statistiques sont sauvegardées (lignes 6 et 7). Finalement, les traces de chaque VM sont mises à jour (ligne 8).
Les différents champs constituant chaque ligne de trace sont définis comme suit :

- l'instant en seconde où l'hyperviseur a été interrogé ;
- le nombre de requêtes de lecture/écriture exécutées par la VM au niveau de l'hyperviseur durant la période précédente ;
- la quantité de donnée lue/écrite en octets par type d'opération ;
- l'identifiant de la VM effectuant les opérations d'E/S ;
- le disque virtuel sur lequel les opérations d'E/S ont été effectuées.

Le tableau 26 présente l'exemple d'un fichier de trace obtenu à l'aide du traceur au niveau hyperviseur.

| time_stamp | #read_req | #read_bytes | #write_req | #write_bytes | VM | VD |
|---|---|---|---|---|---|---|
| 1476367655.94 | 33004 | 4433819648 | 5532 | 2102689792 | vm1 | vda |
| 1476367656.95 | 2 | 36864 | 0 | 0 | vm1 | vda |
| 1476367657.95 | 386 | 2215936 | 0 | 0 | vm1 | vda |
| 1476367658.95 | 183 | 819200 | 0 | 0 | vm1 | vda |
| 1476367663.96 | 0 | 0 | 88 | 29515776 | vm1 | vda |
| 1476367665.97 | 0 | 0 | 168 | 81788928 | vm1 | vda |
| 1476367668.97 | 0 | 0 | 4 | 16384 | vm1 | vda |
| 1476367669.98 | 0 | 0 | 1 | 4096 | vm1 | vda |
| 1476367670.98 | 0 | 0 | 60 | 29360128 | vm1 | vda |
| 1476367672.98 | 0 | 0 | 61 | 29360128 | vm1 | vda |
| 1476367673.98 | 0 | 0 | 5 | 20480 | vm1 | vda |

TABLE 26 : exemple de trace au niveau hyperviseur

Cet exemple permet de déduire que la VM $vm_1$ exécute des opérations de lecture dans un premier temps (de la ligne 1 à la ligne 4), puis des écritures le reste de la trace.

### B.1.2 *Niveau système de fichiers physique*

#### B.1.2.1 *Utilisation de la librairie libext2fs*

Pour utiliser l'API *libext2fs* tout en exécutant des opérations E/S au niveau des VM, nous avons besoins de deux informations essentielles :

1. l'ensemble de partitions sur lesquelles les images des VM sont stockées ;

2. les numéros d'*inodes* des fichiers représentant les disques virtuels des VM.

Il faut noter que deux fichiers stockés sur deux partitions différentes peuvent avoir le même numéro d'*inode*. Afin d'éviter ce type de confusion, chaque disque virtuel est représenté sous la forme suivante :

```
<partition>;<point_de_montage>;<nom_du_fichier>;<numéro_d'inode>
```

Le pseudo-code 6 présente le principe de liaison des différentes couches en utilisant l'API *libext2fs*.

**Algorithme 6 :** utilisation de libext2fs pour la liaison des traces

**Données :** Hyp=$\{vm_1, vm_2, ..., vm_n\}$
**Résultat :** Traces=$\{t_1, t_2, ..., t_n\}$

```
1  LinkLevels (Hyp : hyperviseur)
   /* Récupérer les informations sur le système de fichiers          */
2  pour chaque Machine Virtuelle vm_i dans Hyp faire
3      FS_infos[vm_i] ← Libvirt.info(vm_i);
4  fin
   /* Lancer le processus de traces parallèlement sur les VM         */
5  pour chaque Machine Virtuelle vm_i dans Hyp en parallèle faire
6      pour chaque Disque Virtuel vd_j attaché à vm_i faire
           /* Statistiques niveau hyperviseur                        */
7          Hyp_stats_E/S ← Libvirt.stats(Hyp, vm_i, vd_j)
           /* Traces niveau VFS                                      */
8          VFS_trace ← strace(vm_i)
           /* Trace niveau bloc d'e/s en utilisant blktrace          */
9          Blk_trace ← blktrace(FS_infos[vm_i].partition)
           /* Liaison en ligne des niveaux des traces en utilisant
              libext2fs                                              */
10         Link_trace ← libext2fs(FS_infos[vm_i])
           /* Sauvegarder la nouvelle trace                          */
11         t_i ← {vm_i, vd_j, hyp_stats_E/S, VFS_trace, Blk_trace}
           /* Mettre à jour les traces                               */
12         Update(Traces(t_i))
13     fin
14 fin
15 retourner Traces
```

Ce pseudo-code détaille le processus de liaison les traces (ligne 10) entre les trois couches de trace : hyperviseur (ligne numéro 7), VFS (ligne 8), et bloc (ligne 9). Le résultat de cette opération est un ensemble de fichiers de traces par VM (ligne 11).

## B.2 CARACTÉRISATION DES CHARGES D'E/S DES VM

### B.2.1 *Taux de lecture/écriture*

Le pseudo code 7 établit le calcul du taux de lecture à partir d'un fichier de trace.

**Algorithme 7 :** calcul du taux de lecture

**Données :** Trace={$req_1, req_2, ..., req_n$}, $bloc_{size}$
**Résultat :** $rate_{read}$

1 **initialization :**
2 $req_{lect} \leftarrow 0$
3 $rate_{read} \leftarrow 0$
4 $io_{num} \leftarrow 0$
5 **TauxLecture** (Trace : fichier de trace, $bloc_{size}$ : taille de bloc)
/* Itérer sur l'ensemble des requêtes d'E/S */
6 **tant que** $i < n$ **faire**
/* La requête est elle une lecture ? */
7 **si** $req_{type}(req_i)$ = *lecture* **alors**
/* Mettre à jour le nombre de requêtes de lecture */
8 $req_{lect} \leftarrow req_{lect} + (req_{size}(req_i)/bloc_{size})$
9 **fin**
10 $io_{num} \leftarrow io_{num} + (req_{size}(req_i)/bloc_{size})$
11 $i \leftarrow i + 1$
12 **fin**
13 $rate_{read} \leftarrow req_{lect} \ / \ io_{num}$
14 **retourner** $rate_{read}$

La fonction de calcul de taux de lecture/écriture prend en entrée un fichier de trace et la taille d'un bloc de données qui dépend du niveau de trace (voir partie 4.4.1). Le fichier de trace est parcouru (ligne 6) à la recherche des requêtes de lecture (ligne 7). Le nombre d'opérations de lecture est mis à jour en fonction de la taille des requêtes d'E/S et la taille du bloc de données (ligne 10). La taux de lecture est donc le ratio entre le nombre de requêtes de lecture et le nombre totale des opérations d'E/S exécutées (ligne 13).

### B.2.2 *Taux de séquentialité*

Le pseudo code 8 montre comment extraire le taux de séquentialité à partir d'un ficher de trace.

Cette fonction parcourt le fichier de trace (ligne 6), calcule l'adresse la fin d'une requête d'E/S (ligne 8), récupère l'adresse du début de la requête suivante (ligne

**Algorithme 8 :** calcul du taux de séquentialité

**Données :** Trace={$req_1, req_2, ..., req_n$}, $bloc_{size}$
**Résultat :** $rate_{seq}$

1 **initialization :**
2 $req_{seq} \leftarrow 0$
3 $rate_{seq} \leftarrow 0$
4 $io_{num} \leftarrow 0$
5 **TauxSequentialite** (Trace : fichier de trace, $bloc_{size}$ : taille de bloc)
/* Itérer sur l'ensemble des requêtes d'E/S */
6 **tant que** $i < n$ **faire**
/* La requête j suivant la requête i */
7 **pour** j *où* i<j<n **faire**
/* L'adresse de fin de la requête i */
8 $end \leftarrow req_{addr}(req_i) + req_{size}(req_i)$
/* L'adresse du début de la requête j */
9 $start \leftarrow req_{addr}(req_j)$
10 **si** $start = end + 1$ **alors**
11 $req_{seq} \leftarrow req_{seq} + (req_{size}(req_i)/bloc_{size})$
12 **sinon**
13 $req_{seq} \leftarrow req_{seq} + (req_{size}(req_i)/bloc_{size})$ -1
14 **fin**
15 **fin**
16 $io_{num} \leftarrow io_{num} + (req_{size}(req_i)/bloc_{size})$
17 $i \leftarrow i + 1$
18 **fin**
19 $rate_{seq} \leftarrow req_{seq}$ / $io_{num}$
20 **retourner** $rate_{seq}$

9), et les compare (ligne 7). Le nombre d'opérations séquentielles est mis à jour en fonction de la taille du bloc de données (lignes 16). Le taux de séquentialité est donc le ratio entre le nombre d'opérations séquentielles et le nombre total d'opérations (ligne 19).

### B.2.3 *Taille de requête E/S*

Le pseudo code 9 montre l'utilisation d'un tableau associatif (i.e. des pairs {clé :valeur}) pour obtenir un histogramme des tailles de requêtes. La construction de l'histogramme nécessite le parcours des requêtes (ligne 4) en récupérant leurs tailles (ligne 5). Si la taille a déjà été enregistrée dans le tableau (ligne 6), alors nous mettons à jour son taux d'apparition (ligne 7), sinon nous créons une nou-

**Algorithme 9 :** calcul de l'histogramme des tailles de requêtes

**Données :** Trace=$\{req_1, req_2, ..., req_n\}$
**Résultat :** $Hist_{size} < req_{size} : rate >$

```
1  initialization :
2  Hist_size < req_size : rate > ← ∅
3  HistogrammeTaille(Trace : fichier de trace)
   /* Itérer sur l'ensemble des requêtes d'E/S                    */
4  tant que i < n faire
       /* La requête i est-elle déjà dans l'histogramme ?         */
5      size ← req_size(req_i) ;
6      si size ∈ Hist_size alors
           /* Mettre à jour sa fréquence                          */
7          Hist_size{size} ← Hist_size{size} + 1
8      sinon
           /* Inscrire la nouvelle taille                         */
9          Hist_size{size} ← 1
10     fin
11     i ← i + 1;
12 fin
   /* Convertir la fréquence d'apparition des tailles en taux     */
13 pour chaque size ∈ Hist_size faire
14     Hist_size{size} ← Hist_size{size}/n
15 fin
16 retourner Hist_size
```

velle entrée dans le tableau pour la nouvelle taille de requête (ligne 9). Enfin, nous calculons le taux d'apparition de chaque taille de requête enregistrée dans l'histogramme (ligne 14).

# Annexe C

# OUTILLAGE

## C.1 SYSTÈMES DE STOCKAGE

### C.1.1 *Les systèmes RAID*

L'acronyme RAID a été défini comme *Redundant Arrays of Inexpensive Disks* par les auteurs de [159]. Actuellement, le terme RAID est utilisé pour signifier *Redundant Array of Independent Disks*. Les systèmes de stockage représentent un goulet d'étranglement dans les centres de données. Les mauvaises performances et l'indisponibilité des données sont des problèmes issus de la centralisation des accès aux systèmes de stockage.

Un système RAID est un système de stockage secondaire qui regroupe un ensemble de disques durs [9]. Ils sont connus sous le nom de "grappes de disques".

L'objectif principal de regrouper les disques est, d'une part, d'offrir un espace de stockage plus conséquent, et d'autre part de permettre la parallélisation des accès sur les différents périphériques afin d'augmenter les performances. Les RAID sont utilisés pour augmenter la fiabilité des données grâce à la redondance.

Il y a sept niveaux de RAID (de 0 jusqu'à 6), selon la configuration de l'ensemble des disques dont trois sont les plus courants, à savoir RAID0, RAID1 et RAID5 [47]. Les différents niveaux RAID ciblent l'amélioration des performances et la fiabilité de stockage de données.

### C.1.2 *Stockage de VM et disques virtuels*

Les fichiers des disques virtuels peuvent avoir l'une des deux formes suivantes [96] :

- *flat* où la taille du fichier est fixée dés sa création, et chaque bloc dans le système invité correspond à un bloc sur le disque du système hôte. Après sa création, la totalité de la taille du fichier est allouée et tous les blocs non-utilisés initialisés à zéros.

- *sparse* contrairement aux fichiers *flat*, les fichiers *sparse* contiennent seulement les données qui ont été écrites par la VM. La taille du fichier augmente au fur et à mesure des besoins de la VM en terme d'espace de stockage.

Les images disques des VM peuvent avoir différents formats. Le format des images disques dépend de son type (*flat* ou *sparse*), et de l'hyperviseur utilisé. Nous pouvons lister les formats les plus utilisés :

- ***raw*** : c'est le format le plus simple et le plus portable parmi tous. Un disque virtuel en format *raw* est un fichier binaire contenant les données brutes de l'image disque de la VM.
- ***vdi*** : *Virtual Disk Image*, format des disques virtuels utilisé pas *VirtualBox* [149].
- ***vmdk*** : *Virtual Machine DisK* format proposé et utilisé pas les hyperviseurs VMware.
- ***vhd(x)*** : *Virtual Hard Disk* format utilisé par les hyperviseurs Microsoft *Hyper-V*.
- ***a(k/m/r)i*** : formats d'image disque utilisés par Amazon (*Amazon Kernel/Machine/Ram Image*).
- ***iso*** : c'est plutôt un format d'archivage de contenu de disques optiques.
- ***qcow(2)*** : désigne *Qemu Copy-On-Write*, c'est le format utilisé par *QEMU*. Comme son nom l'indique, ce format utilise la technique *Copy-On-Write* qui permet à plusieurs VM d'utiliser le même disque virtuel. En utilisant cette technique, les VM peuvent accéder aux mêmes disques virtuelles tant qu'elles n'écrivent pas. Si une VM tente une modification, une copie de l'image disque est copiée et dédiée à cette VM (d'où le terme *copy-on-write*). Les modifications apportées par *qcow* et *qcow2* se focalisent principalement sur la compression et le chiffrement des données, et le support des instantanés (*snapshot*). Nous avons particulièrement détaillé ce format car il s'agit du format que nous avons utilisé dans nos expérimentations. Ce format est largement utilisé par la communauté. KVM/QEMU et Xen, qui sont parmi les hyperviseurs les plus courants dans les travaux de l'état de l'art l'utilisent.

## C.2 MONITORING DES E/S DES VM

### C.2.1 *Traceurs génériques*

*SystemTap* permet de tracer des fonctions à différents niveaux du noyau Linux. L'exemple suivant présente un script qui trace l'appel système `open`, affiche le nom exacte et le PID du processus exécutant l'appel, ainsi que le fichier ouvert.

```
probe syscall.open
{
        printf ("%s(%d) open (%s)\n", execname(), pid(), argstr)
}
probe timer.ms(4000) # after 4 seconds
{
        exit ()
}
```

Nous pouvons noter que cet exemple trace un appel système en utilisant le nom réservé `syscall` et l'appel système `open` séparés par un point. Cette procédure reste la même pour tracer une fonction noyau. Un exemple de trace des fonctions noyau `vfs_read` et `vfs_write` peut être réalisé comme suit :

```
probe kernel.function ("vfs_read"),
  kernel.function ("vfs_write")
{
        /* Traitement pour chaque appel de vfs_read / vfs_write */
}
```

Pour exécuter un script *SystemTap*, il suffit d'avoir installé les outils *SystemTap*, de lancer la commande `stap` en passant le script en question comme paramètre.

Pour utiliser *ftrace*, il suffit que la fonction soit activée dans le noyau (elle est activée par défaut dans les récentes distributions de GNU/Linux), et de monter le répertoire `/sys/kernel/debug` (celui-ci se crée automatiquement lorsque *ftrace* est activé) avec le système de fichier `debugfs`. Tous les fichiers en relation avec *ftrace* se trouvent dans le répertoire `/sys/kernel/debug/tracing`.

```
root@debian:~# echo function > /sys/kernel/debug/tracing/current_tracer
```

Cette opération permet de notifier *ftrace* de l'activation de la trace de toutes les fonctions noyau. Les traces obtenues sont sauvegardées dans le fichier `/sys/kernel/debug/tracing/trace`. Voici un exemple des premières lignes du fichier de trace :

```
root@debian:~# less /sys/kernel/debug/tracing/trace
# tracer: function
#
```

```
# entries-in-buffer/entries-written: 409996/144662086   #P:8
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#       TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#          | |       |   ||||       |         |
       less-21905 [007] .... 36216.834503: rw_verify_area <-vfs_read
       less-21905 [007] .... 36216.834504: fsnotify <-rw_verify_area
       less-21905 [007] .... 36216.834504: __vfs_read <-vfs_read
       less-21905 [007] .... 36216.834505: new_sync_read <-vfs_read
       less-21905 [007] .... 36216.834505: pipe_read <-new_sync_read
       less-21905 [007] .... 36216.834505: mutex_lock <-pipe_read
       less-21905 [007] .... 36216.834506: _cond_resched <-mutex_lock
       less-21905 [007] .... 36216.834506: mutex_unlock <-pipe_read
       less-21905 [007] .... 36216.834507: __close_fd <-SyS_close
       less-21905 [007] .... 36216.834507: _raw_spin_lock <-__close_fd
       less-21905 [007] .... 36216.834507: filp_close <-SyS_close
       less-21905 [007] .... 36216.834507: dnotify_flush <-filp_close
       ⋮
```

### C.2.2 *Niveau VFS*

#### C.2.2.1 *Utilisation de strace*

**a) Étape 1 (trace de processus et filtrage des fonctions) :** la commande suivante présente un exemple d'utilisation de *strace* dans notre contexte :

```
$ strace <vm_pid> -ttt -e trace=create,open,read,write,lseek,close
```

Pour le filtrage à la volé, l'option *-e trace=* de *strace* permet de spécifier les appels systèmes à tracer. L'option *-ttt* affiche le temps en microsecondes avant chaque appel système.
Le format des lignes du fichier de trace en sortie est donnée comme suit :

```
<time_stamp>;<i/o_systemcall_with_args>;<return_value>
```

- le premier champ présente l'estampillage de temps de l'appel système sous la forme suivante : `secondes.microsecondes;`
- le deuxième champ représente l'appel système sous la forme *func(*$arg_1$, $arg_2$, *...)*, où *func* est l'appel système et *(*$arg_1$, $arg_2$, *...)* sont les valeurs des arguments de la fonction ;
- le dernier champ est la valeur de retour de la fonction (e.g. le nombre d'octets écrits/lus pour les fonctions `write`/`read`).

**b) Étape 2 (isolation des traces par disque virtuel) :** D'une manière générale, les opérations d'E/S tracées sur un fichier existant ont la forme suivante :

```
open("/chemin/vers/le/fichier", flag1 | flag2 | ...) = fd
⋮
read(fd, "\des\octets\lus\...", count) = count
lseek(fd, 0 , SEEK_END) = offset
write(fd, "\des\octets\ecrits\...", count) = count
⋮
close(fd) = 0
```

Chaque accès est une fonction système d'E/S avec une ou plusieurs signatures prédéfinies. Ainsi, l'un des prototypes de la fonction `open` est décrit comme suit :
`int open(const char *pathname, int flags)`
Cette fonction ouvre le fichier en donnant son chemin exact et le mode d'accès (lecture seule, écriture seule, ou lecture et écriture). La valeur retournée par cette fonction est un nombre entier qui représente le descripteur de fichier ouvert. Cet entier sera utilisé par les opérations d'E/S tant que le fichier n'est pas fermé.

Le tableau 27 montre l'exemple d'une trace de VM exécutant des opérations d'E/S, à l'aide de *strace*.

| time_stamp | I/O | fd | #bytes | ret_val |
|---|---|---|---|---|
| 1476371428.384310 | write | 6 | 8 | 8 |
| 1476371428.384310 | write | 7 | 8 | 8 |
| 1476371428.384568 | read | 6 | 512 | 512 |
| 1476371428.384629 | write | 7 | 8 | 8 |
| 1476371428.384774 | read | 7 | 16 | 16 |
| 1476371428.385359 | write | 6 | 8 | 8 |
| 1476371428.385446 | write | 7 | 8 | 8 |
| 1476371428.385481 | read | 6 | 512 | 512 |
| 1476371428.385537 | write | 7 | 8 | 8 |
| 1476371428.385668 | read | 7 | 16 | 16 |

TABLE 27 : exemple de trace au niveau VFS

### C.2.2.2 *Utilisation de et résultats de jprobe*

**a) Étape 1 (inspection de l'exécution des E/S) :** la figure 66 présente le processus suivi pour l'inspection des fonctions d'E/S spécifiques au système de fichiers virtuel en utilisant GDB.
**b) Étape 2 (placement des sondes) :** les sondes ont été posées sur deux fonctions d'E/S définies dans `Linux/mm/filemap.c` :

- `generic_file_read_iter` pour les opérations de lecture ;

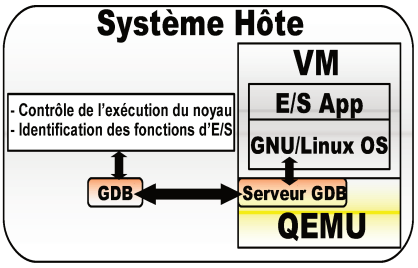


FIGURE 66 : utilisation de GDB pour l'identification des fonctions d'E/S à sonder

- `generic_file_write_iter` pour les opérations d'écriture.

La signature des deux fonctions (qui est indispensable pour les fonctions *handler*) est la suivante :

```
/*
* @iocb:des informations sur l'opération d'E/S (file,offset,...)
* @iter:une structure pour la gestion du tampon de lecture
*/
ssize_t
generic_file_read_iter(struct kiocb *iocb, struct iov_iter *iter);

ssize_t
generic_file_write_iter(struct kiocb *iocb, struct iov_iter *from)
```

Le tableau 28 montre l'exemple d'une trace de VM après le filtrage des opérations d'E/S au niveau VFS.

| time_stamp | I/O | offset | #bytes | level | process | vm_pid |
|---|---|---|---|---|---|---|
| 48.201808230 | R | 2338701312 | 12288 | VFS | qemu-system-x86 | 10024 |
| 48.210404783 | R | 7036846080 | 4096 | VFS | qemu-system-x86 | 10024 |
| 48.211709964 | R | 2318831616 | 4096 | VFS | qemu-system-x86 | 10024 |
| 48.218996494 | R | 2307981312 | 8192 | VFS | qemu-system-x86 | 10024 |
| 48.227876009 | R | 6916722688 | 4096 | VFS | qemu-system-x86 | 10024 |
| 48.820261330 | W | 4434075648 | 131072 | VFS | qemu-system-x86 | 10024 |
| 48.820397606 | W | 4434206720 | 131072 | VFS | qemu-system-x86 | 10024 |
| 48.820549169 | W | 4434337792 | 131072 | VFS | qemu-system-x86 | 10024 |
| 48.821871439 | R | 2321256448 | 12288 | VFS | qemu-system-x86 | 10024 |
| 48.821918064 | W | 4435378176 | 40960 | VFS | qemu-system-x86 | 10024 |

TABLE 28 : exemple de trace au niveau VFS

### C.2.3 *Niveau bloc d'E/S*

Mis à part les pilotes spécifiques aux périphériques de stockage physiques, le niveau bloc d'E/S représente la dernière couche de la pile logicielle d'E/S avant d'atteindre le périphérique de stockage physique (voir partie 2.3).

La couche logicielle de bloc d'E/S est chargée de diriger les requêtes d'E/S en provenance des applications vers les périphériques de stockage physiques [18]. La couche bloc d'E/S contient un élément essentiel qui est l'ordonnanceur des requêtes d'E/S (I/O *scheduler*). Les anciennes versions du noyau Linux (avant la version 3.13) implantent un seul ordonnanceur d'E/S qui gère l'ensemble des requêtes pour tous les périphériques de stockage.

Cette implantation a causé des problèmes de performances du fait que l'ordonnanceur représente un réel goulet d'étranglement. À partir de la version 3.13 du noyau Linux, la couche bloc désormais supporte plusieurs files d'attente des E/S, une file par cœur CPU. Cette approche a été proposée par Matias Bjørling et *al.* [28], dans le but de supporter les périphériques de stockage performants de type SSD et les processeurs multi-cœurs [28]. La structure interne actuelle peut être simplifiée de la manière présentée par la figure 67.

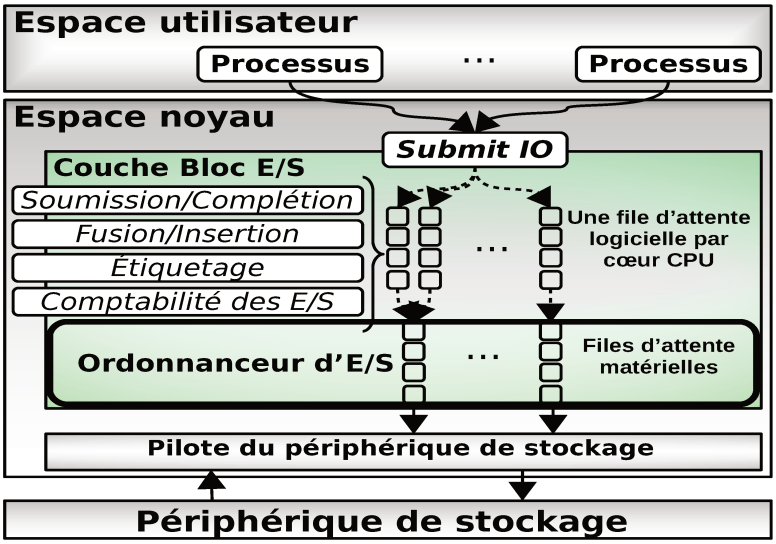


FIGURE 67 : structure et interaction de la couche bloc d'E/S [28]

#### C.2.3.1 *Utilisation de blktrace*

La figure 68 montre un exemple de la trace d'une VM exécutant des opérations d'E/S, obtenues à l'aide de *blktrace*.

Le PID du processus exécutant les requêtes d'E/S (champ numéro 1) nous permet d'isoler les traces pour distinguer les VM. Le PID du processus d'une VM est le même utilisé pour la trace au niveau VFS. Le type d'opération (champ numéro 3) nous permet de garder uniquement les requêtes de lecture/écriture.

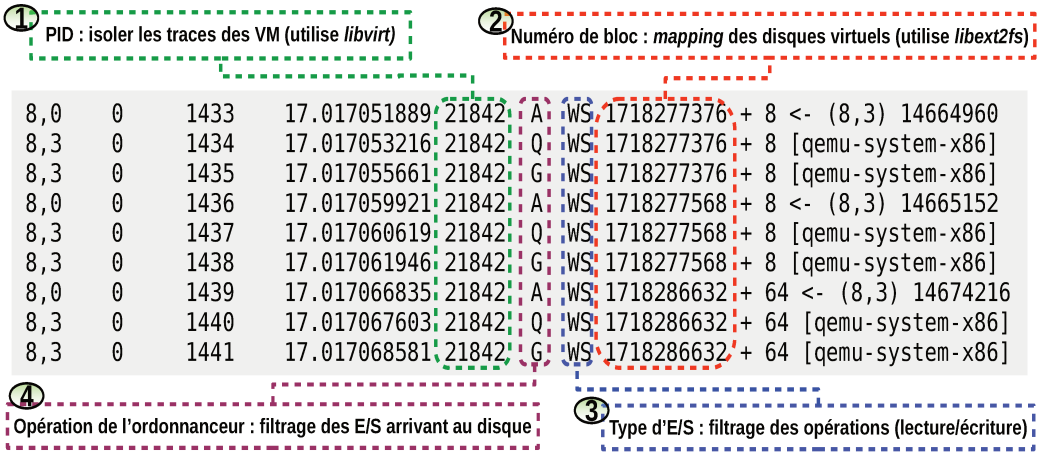


FIGURE 68 : paramètres utilisés pour le post-traitement

Le champs indiquant les opérations effectuées au niveau de la file d'attente des requêtes d'E/S (champ numéro 4) est utilisé pour ne conserver que les requêtes d'E/S achevées.

### C.2.3.2 *Utilisation de jprobe*

Après avoir inspecté le fonctionnement des E/S (voir figure 66), notre choix s'est porté sur la fonction `generic_make_request()`. Cette fonction remplit les conditions souhaitées en étant positionnée au niveau de la couche bloc d'E/S (i.e. déclarée dans les sources du noyau Linux dans `linux/include/linux/blkdev.h`, implantée dans `linux/block/blk-core.c`), et est utilisée par les pilotes des périphériques de stockage (i.e. référencée dans `Linux/drivers/block` et `Linux/dirvers/md/`).

La fonction `generic_make_request()` se charge de soumettre le *buffer* d'une opération d'E/S (i.e. lecture/écriture) à l'ordonnanceur d'E/S, qui transfert par la suite au pilote du périphérique de stockage correspondant. La signature de la fonction `generic_make_request()` est définie comme suit :

```
/*
* @bio: structure décrivant l'opération d'E/S, le buffer sur, la mémoire,
* et le bloc sur le périphérique de stockage
*/
blk_qc_t generic_make_request(struct bio *bio)
```

La fonction `generic_make_request()` est utilisée pour exécuter des requêtes d'E/S décrites dans la structure `bio`. En effet, la structure `bio` nous permet d'avoir toutes les informations nécessaires sur l'opération d'E/S en cours d'exécution. Voici la structure `bio` avec uniquement les éléments utilisés pour notre traceur :

```
struct bio {
struct block_device *bi_bdev; /* Périphérique de stockage */
unsigned long       bi_rw;    /* Type d'opération d'E/S */
struct bvec_iter    bi_iter;  /* Numéro de bloc, nombre de blocs,
```

```
                                * numéro d'inode */
};
```

Le paramètre *bi_bdev* est la représentation dans le noyau du périphérique de stockage ou de la partition sur laquelle l'opération d'E/S est exécutée. Ce paramètre nous permet de distinguer les différents disques ou partitions, et ainsi de tracer plusieurs périphériques à la fois (contrairement à *blktrace*). Le paramètre `bi_rw` définit le type d'opération d'E/S et peut prendre READ ou WRITE comme valeur. `bi_iter` est le paramètre le plus important, car il permet d'avoir :

- le numéro du premier bloc à partir duquel l'opération d'E/S commence ;
- le nombre de blocs consécutifs à lire/écrire à partir du premier bloc ;
- l'*inode* du fichier sur lequel l'opération d'E/S est exécutée, et ainsi de faire la liaison avec les couches hautes du traceur.

Le tableau 29 montre l'exemple d'une trace de VM après le filtrage des opérations d'E/S au niveau bloc.

| time_stamp | I/O | block_num | #bytes | level | process | vm_pid |
|---|---|---|---|---|---|---|
| 0.013511914 | R | 27263920 | 65536 | BLK | qemu-system-x86 | 10024 |
| 0.042776809 | R | 1409024 | 8192 | BLK | qemu-system-x86 | 10024 |
| 0.143623920 | R | 1410864 | 73728 | BLK | qemu-system-x86 | 10024 |
| 0.151510258 | R | 36281192 | 4096 | BLK | qemu-system-x86 | 10024 |
| 0.151542525 | R | 36281208 | 8192 | BLK | qemu-system-x86 | 10024 |
| 0.164979220 | R | 27263744 | 16384 | BLK | qemu-system-x86 | 10024 |
| 0.782172300 | W | 406896 | 12288 | BLK | qemu-system-x86 | 10024 |
| 0.782183894 | W | 406952 | 4096 | BLK | qemu-system-x86 | 10024 |
| 0.782184802 | W | 406928 | 4096 | BLK | qemu-system-x86 | 10024 |
| 0.782194789 | W | 411024 | 28672 | BLK | qemu-system-x86 | 10024 |

TABLE 29 : exemple de trace au niveau bloc

## RÉFÉRENCES


[1] Libreoffice calc : Linest function. https://help.libreoffice.org/Calc/Array_Functions/fr#Other_LINEST_Results:. Visité en Aout 2016.

[2] Jason rudy : py-earth project. https://github.com/jcrudy/py-earth. Accessed in Aug 2016.

[3] Ms excel : Linest function. https://support.office.com/en-us/article/LINEST-function-84d7d0d9-6e50-4101-977a-fa7abf772b6d. Visité en Aout 2016.

[4] Google : From 112 servers to a $5b-plus quarterly data center bill. http://www.datacenterknowledge.com/archives/2014/07/23/from-112-servers-to-5b-spent-on-google-data-centers-per-quarter/. Visite en Nov 2016.

[5] I/o tracer kernel module. https://github.com/b-com/iotracer. Accessed in Aug 2016.

[6] Scikit-learn. http://scikit-learn.org. Visité en Aout 2016.

[7] SimGrid : Versatile simulation of distributed systems. http://simgrid.gforge.inria.fr/. Accessed : 2017-03-11.

[8] strace(1) - linux man page. https://linux.die.net/man/1/strace. Visité en Nov 2016.

[9] *Systemes a Haute Disponibilite*. Ed. Techniques Ingénieur. URL https://books.google.fr/books?id=TcJmL48d3bIC.

[10] 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*, 2008.

[11] Irfan Ahmad. Easy and efficient disk i/o workload characterization in vmware esx server. In *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*, pages 149–158. IEEE, 2007.

[12] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab. Hamid, Muhammad Shiraz, Abdullah Yousafzai, and Feng Xia. A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications*, 2015.

[13] Mohammed Alhamad, Tharam Dillon, and Elizabeth Chang. A survey on sla and performance measurement in cloud computing. *On the Move to Meaningful Internet Systems : OTM 2011*, pages 469–477, 2011.

[14] Mike Allen. Data center power costs and requirements. https://www.datacenters.com/news/infrastructure/135-data-center-power-costs-and-requirements. Visité en Nov 2016.

[15] Amazon. Amazon ec2. https://www.amazonaws.cn/en/ec2/, Visité le 05/03/2017.

[16] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53 (4) :50–58, 2010.

[17] Jens Axboe. blktrace user guide. https://www.cse.unsw.edu.au/~aaronc/iosched/doc/blktrace.html. Visité en Nov 2016.

[18] Jens Axboe. Linux block io—present and future. In *Ottawa Linux Symp*, pages 51–61, 2004.

[19] Jens Axboe. Fio. https://github.com/axboe/fio, Visité le 08/06/2016.

[20] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 164–177. ACM, 2003.

[21] Luiz André Barroso. The price of performance. *Queue*, 3(7) :48–53, 2005.

[22] Dávid Bartók and Zoltán Ádám Mann. A branch-and-bound approach to virtual machine placement. *Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam*, page 49, 2015.

[23] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[24] Anton Beloglazov and Rajkumar Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 577–578. IEEE, 2010.

[25] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation : Practice and Experience*, 24(13) :1397–1420, 2012.

[26] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer System*, 28, May 2012.

[27] David Bernstein. Containers and cloud : From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3) :81–84, 2014.

[28] Matias Bjørling, Jens Axboe, David Nellans, and Philippe Bonnet. Linux block io : introducing multi-queue ssd access on multi-core systems. In *Proceedings of the 6th International Systems and Storage Conference*, page 22. ACM, 2013.

[29] A.E.H. Bohra and V. Chaudhary. Vmeter : Power modelling for virtualized clouds. In *IPDPS Workshops*, 2010.

[30] Matthias Bolte, Michael Sievers, Georg Birkenheuer, Oliver Niehörster, and André Brinkmann. Non-intrusive virtualization management using libvirt. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 574–579. European Design and Automation Association, 2010.

[31] Rajesh Bose, Sandip Roy, and Debabrata Sarddar. *On Demand IOPS Calculation in Cloud Environment to Ease Linux-Based Application Delivery*, pages 71–77. Springer Singapore, Singapore, 2017. ISBN 978-981-10-2035-3.

[32] Luc Bouganim and Philippe Bonnet. Flash Device Support for Database Management. In *5th Biennal Conference on Innovative Data Systems Research (CIDR)*, pages 1–8, Asilomar, California, United States, January 2011. URL https://hal.archives-ouvertes.fr/hal-00666180.

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

[34] J. Boukhobza and P. Olivier. *Flash Memory Integration : Performance and Energy Issues*. Elsevier Science, 2017. ISBN 9780081011584. URL https://books.google.fr/books?id=zEoADQAAQBAJ.

[35] Jalil Boukhobza. Flashing in the Cloud : Shedding some Light on NAND Flash Memory Storage Systems. In *Data Intensive Storage Services for Cloud Environments*. IGI Global, 2013.

[36] D.P. Bovet and M. Cesati. *Understanding the Linux Kernel : From I/O Ports to Process Management*. O'Reilly Media, 2005. ISBN 9780596554910. URL https://books.google.fr/books?id=h0lltXyJ8aIC.

[37] Per Nikolaj D Bukh and Raj Jain. The art of computer systems performance analysis, techniques for experimental design, measurement, simulation and modeling, 1992.

[38] Axel Busch, Qais Noorshams, Samuel Kounev, Anne Koziolek, Ralf Reussner, and Erich Amrehn. Automated workload characterization for i/o performance analysis in virtualized environments. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, pages 265–276. ACM, 2015.

[39] Rajkumar Buyya and Manzur Murshed. Gridsim : A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation : practice and experience*, 14 (13-15) :1175–1220, 2002.

[40] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software : Practice and Experience*, 41(1) :23–50, 2011.

[41] 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, 2009.

[42] Rémy Card, Theodore Ts'o, and Stephen Tweedi. Design and implementation of the second extended filesystem. In *Proceedings of the First Dutch International Symposium on Linux*, 1994.

[43] Aaron Carroll. fio, Visited in 04/13/2015. URL http://linux.die.net/man/1/fio.

[44] Henri Casanova. Simgrid : A toolkit for the simulation of application scheduling. In *Cluster computing and the grid, 2001. proceedings. first ieee/acm international symposium on*, pages 430–437. IEEE, 2001.

[45] Henri Casanova, Arnaud Legrand, and Martin Quinson. Simgrid : A generic framework for large-scale distributed experiments. In *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*, pages 126–131. IEEE, 2008.

[46] Feng Chen, Tian Luo, and Xiaodong Zhang. Caftl : A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11. USENIX Association, 2011.

[47] Peter M Chen, Edward K Lee, Garth A Gibson, Randy H Katz, and David A Patterson. Raid : High-performance, reliable secondary storage. *ACM Computing Surveys (CSUR)*, 26(2) :145–185, 1994.

[48] Yuan Chen. Flash memory reliability nepp 2008 task final report. Technical report, National Aeronautics and Space Administration, 2009. URL https://nepp.nasa.gov/files/16376/08_106_1%20JPL%20Chen%20Flash%20Memory.pdf.

[49] Yue Cheng, M Safdar Iqbal, Aayush Gupta, and Ali R Butt. Cast : Tiering storage for data analytics in the cloud. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 45–56. ACM, 2015.

[50] William S Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American statistical association*, 74(368) :829–836, 1979.

[51] William S Cleveland and Clive Loader. Smoothing by local regression : Principles and methods. In *Statistical theory and computational aspects of smoothing*, pages 10–49. Springer, 1996.

[52] Edward G Coffman Jr, Michael R Garey, and David S Johnson. Approximation algorithms for bin packing : A survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., 1996.

[53] Russell Coker. Bonnie++, Visited in 04/12/2016. URL http://www.coker.com.au/bonnie++/.

[54] Maxime Colmant, Mascha Kurpicz, Pascal Felber, Loïc Huertas, Romain Rouvoy, and Anita Sobe. Process-level power estimation in vm-based systems. In *European Conference on Computer Systems*, 2015.

[55] Autonomic Computing et al. An architectural blueprint for autonomic computing. *IBM White Paper*, 31, 2006.

[56] Transaction Processing Performance Council. Active tpc benchmarks. http://www.tpc.org/information/benchmarks.asp. Visité en Nov 2016.

[57] Datasheet. Samsung ssd 840 pro series. Technical report, 2013.

[58] Datasheet. Samsung ssd 850 pro. Technical report, 2015. URL http://www.samsung.com/semiconductor/minisite/ssd/downloads/document/Samsung_SSD_850_PRO_Data_Sheet_rev_2_0.pdf.

[59] Silas De Munck, Kurt Vanmechelen, and Jan Broeckhove. Improving the scalability of simgrid using dynamic routing. In *International Conference on Computational Science*, pages 406–415. Springer, 2009.

[60] D.M. Dhamdhere. *Operating Systems : A Concept-based Approach,2E*. McGraw-Hill Higher Education, 2006. ISBN 9780070611948. URL https://books.google.fr/books?id=kbBn4X9x2mcC.

[61] Clara Dismuke and Richard Lindrooth. Ordinary least squares. *Methods and Designs for Outcomes Research*, 93 :93–104, 2006.

[62] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh Prabhu. iperf, Visited in 04/12/2016. URL https://iperf.fr/.

[63] Kaoutar El Maghraoui, Gokul Kandiraju, Joefon Jann, and Pratap Pattnaik. Modeling and simulating flash based solid-state disks for operating systems. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 15–26. ACM, 2010.

[64] Jerome H Friedman. Multivariate adaptive regression splines. *The annals of statistics*, pages 1–67, 1991.

[65] Marc E Frincu and Ciprian Craciun. Multi-objective meta-heuristics for scheduling applications with high availability requirements and cost constraints in multi-cloud environments. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 267–274. IEEE, 2011.

[66] Yongqiang Gao, Haibing Guan, Zhengwei Qi, Yang Hou, and Liang Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, 79(8) :1230–1242, 2013.

[67] Saurabh Kumar Garg, Srinivasa K. Gopalaiyengar, and Rajkumar Buyya. Sla-based resource provisioning for heterogeneous workloads in a virtualized cloud datacenter. In *International conference on Algorithms and architectures for parallel processing*, 2011.

[68] Saurabh Kumar Garg, Adel Nadjaran Toosi, Srinivasa K. Gopalaiyengar, and Rajkumar Buyya. Sla-based virtual machine management for heterogeneous workloads in a cloud datacenter. *J. Network and Computer Applications*, 2014.

[69] Geoff Gasior. Maxtor's diamondmax 10 hard drive. Technical report, Seagate, Visité en Jan 2016. URL http://techreport.com/review/7903/maxtor-diamondmax-10-hard-drive.

[70] Devarshi Ghoshal, Richard Shane Canon, and Lavanya Ramakrishnan. I/o performance of virtualized cloud environments. In *Proceedings of the second international workshop on Data intensive computing in the clouds*, pages 71–80. ACM, 2011.

[71] D. Gmach, J. Rolia, and L. Cherkasova. Resource and virtualization costs up in the cloud : Models and design choices. In *International Conference on Dependable Systems Networks*, 2011.

[72] 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 *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 326–335. IEEE, 2008.

[73] Gene H Golub, Michael Heath, and Grace Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2) :215–223, 1979.

[74] Google. Google cloud storage, google prediction api, and google bigquery sla, Visited in 03/09/2015. URL https://cloud.google.com/storage/sla.

[75] Google. Compute engine. https://cloud.google.com/compute/, Visité le 05/03/2017.

[76] Aaron Grattafiori. Ncc group whitepaper : Understanding and hardening linux containers. Technical Report 1.0, NCC Group, April 2016. URL https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2016/april/ncc_group_understanding_hardening_linux_containers-10pdf/.

[77] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. The cost of a cloud : research problems in data center networks. *ACM SIGCOMM computer communication review*, 39(1) :68–73, 2008.

[78] B. Gregg. *Systems Performance : Enterprise and the Cloud*. Pearson Education, 2013. ISBN 9780133390100. URL https://books.google.fr/books?id=pTYkAQAAQBAJ.

[79] Laura M Grupp, Adrian M Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H Siegel, and Jack K Wolf. Characterizing flash memory : anomalies, observations, and applications. In *42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 24–33. IEEE, 2009.

[80] Marco Guazzone, Cosimo Anglano, and Massimo Canonico. Exploiting vm migration for the automated power and performance management of green cloud computing systems. In *International Workshop on Energy Efficient Data Centers*, pages 81–92. Springer, 2012.

[81] Brian Guenter, Navendu Jain, and Charles Williams. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *INFOCOM, 2011 Proceedings IEEE*, pages 1332–1340. IEEE, 2011.

[82] Ajay Gulati, Chethan Kumar, and Irfan Ahmad. Storage workload characterization and consolidation in virtualized environments. In *Workshop*

*on Virtualization Performance : Analysis, Characterization, and Tools (VPACT)*, 2009.

[83] Ajay Gulati, Chethan Kumar, and Irfan Ahmad. Modeling workloads and devices for io load balancing in virtualized environments. *ACM SIGMETRICS Performance Evaluation Review*, 37(3) :61–66, 2010.

[84] Ajay Gulati, Chethan Kumar, Irfan Ahmad, and Karan Kumar. Basil : Automated io load balancing across storage devices. In *FAST*, volume 10, pages 169–182, 2010.

[85] Rw Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 26 :147–160, 1950.

[86] Rui Han, Li Guo, Moustafa M Ghanem, and Yike Guo. Lightweight resource scaling for cloud applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 644–651. IEEE, 2012.

[87] David F Hinnant. Benchmarking unix systems. *Byte*, 9(8) :132, 1984.

[88] Takahiro Hirofuchi, Adrien Lèbre, and Laurent Pouilloux. Adding a live migration model into simgrid : One more step toward the simulation of infrastructure-as-a-service concerns. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 96–103. IEEE, 2013.

[89] Takahiro Hirofuchi, Adrien Lebre, and Laurent Pouilloux. Simgrid vm : Virtual machine support for a simulation framework of distributed systems. *IEEE Transactions on Cloud Computing*, 2015.

[90] Paul Horn. Ibm's perspective on the state of information technology, 2001. URL http://people.scs.carleton.ca/~soma/biosec/readings/autonomic_computing.pdf.

[91] Qiang Huang, Fengqian Gao, Rui Wang, and Zhengwei Qi. Power consumption of virtual machine live migration in clouds. In *International Conference on Communications and Mobile Computing*, 2011.

[92] Abdul R Hummaida, Norman W Paton, and Rizos Sakellariou. Adaptation in cloud resource configuration : a survey. *Journal of Cloud Computing*, 5(1) : 1–16, 2016.

[93] Chris Hyser, Bret Mckee, Rob Gardner, and Brian J Watson. Autonomic virtual machine placement in the data center. *Hewlett Packard Laboratories, Tech. Rep. HPL-2007-189*, pages 2007–189, 2007.

[94] IBM. Ibm 350 disk storage unit, Visité le 27/05/2016. URL http://www-03.ibm.com/ibm/history/exhibits/storage/storage_350.html.

[95] Keren Jin and Ethan L. Miller. The effectiveness of deduplication on virtual machine disk images. In *Proceedings of SYSTOR 2009 : The Israeli Experimental Systems Conference*, SYSTOR '09, pages 7 :1–7 :12, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-623-6. doi : 10.1145/1534530.1534540. URL http://doi.acm.org/10.1145/1534530.1534540.

[96] Keren Jin and Ethan L Miller. The effectiveness of deduplication on virtual machine disk images. In *Proceedings of SYSTOR 2009 : The Israeli Experimental Systems Conference*, page 7. ACM, 2009.

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

[98] David S Johnson. *Near-optimal bin packing algorithms*. PhD thesis, Massachusetts Institute of Technology, 1973.

[99] Karthik Kambatla and Yanpei Chen. The truth about mapreduce performance on ssds. In *LISA*, pages 109–118, 2014.

[100] Aman Kansal, Feng Zhao, Jie Liu, Nupur Kothari, and Arka A. Bhattacharya. Virtual machine power metering and provisioning. In *ACM Symposium on Cloud Computing*, 2010.

[101] Jeffrey Katcher. Postmark : A new file system benchmark. Technical report, Technical Report TR3022, Network Appliance, 1997.

[102] Jim Keniston, Prasanna S Panchamukhi, and Masami Hiramatsu. Kernel probes (kprobes). https://www.kernel.org/doc/Documentation/kprobes.txt. Visité en Nov 2016.

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

[104] 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 (NOMS), 2012 IEEE*, pages 1287–1294. IEEE, 2012.

[105] Hyojun Kim and Seongjun Ahn. Bplru : A buffer management scheme for improving random writes in flash storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08. USENIX Association, 2008.

[106] Jesung Kim, Jong Min Kim, Sam H Noh, Sang Lyul Min, and Yookun Cho. A space-efficient flash translation layer for compactflash systems. *IEEE Transactions on Consumer Electronics*, 48(2) :366–375, 2002.

[107] Nakku Kim, Jungwook Cho, and Euiseong Seo. Energy-based accounting and scheduling of virtual machines in a cloud system. In *IEEE GreenCom*, 2011.

[108] Youngjae Kim, Aayush Gupta, Bhuvan Urgaonkar, Piotr Berman, and Anand Sivasubramaniam. Hybridstore : A cost-efficient, high-performance storage system combining ssds and hdds. In *IEEE MASCOTS*, 2011.

[109] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm : the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, Ottawa, Ontario, Canada, JUN 2007.

[110] Alexey Kopytov. Sysbench manual. `http://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf`, Visité le 08/06/2016.

[111] Bhavani Krishnan, Hrishikesh Amur, Ada Gavrilovska, and Karsten Schwan. Vm power metering : Feasibility and challenges. *SIGMETRICS PE.Rev*, 2011.

[112] Mascha Kurpicz, Anita Sobe, et al. How much does a vm cost ? energy-proportional accounting in vm-based environments. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 651–658. IEEE, 2016.

[113] D. Kusnetzky. *Virtualization : A Manager's Guide*. Real Time Bks. O'Reilly Media, Incorporated, 2011. ISBN 9781449306458. URL `https://books.google.fr/books?id=dV_L5CbsDscC`.

[114] Ulrich Lampe, Melanie Siebenhaar, Ronny Hans, Dieter Schuller, and Ralf Steinmetz. Let the clouds compute : cost-efficient workload distribution in infrastructure clouds. In *International Conference on Grid Economics and Business Models*, pages 91–101. Springer, 2012.

[115] Kien Le, Ricardo Bianchini, Jingru Zhang, Yogesh Jaluria, Jiandong Meng, and Thu D. Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[116] Adrien Lebre, Arnaud Legrand, Frédéric Suter, and Pierre Veyre. Adding storage simulation capacities to the simgrid toolkit : Concepts, models, and api. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 251–260. IEEE, 2015.

[117] Adrien Lebre, Jonathan Pastor, and Mario Südholt. Vmplaces : A generic tool to investigate and compare vm placement algorithms. In *European Conference on Parallel Processing*, pages 317–329. Springer, 2015.

[118] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. What's inside the cloud ? an architectural map of the cloud landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, May 2009.

[119] Bo Li, Jianxin Li, Jinpeng Huai, Tianyu Wo, Qin Li, and Liang Zhong. Enacloud : An energy-saving application live placement approach for cloud computing environments. In *2009 IEEE International Conference on Cloud Computing*, pages 17–24. IEEE, 2009.

[120] Wubin Li, Johan Tordsson, and Erik Elmroth. Virtual machine placement for predictable and time-constrained peak loads. In *International Workshop on Grid Economics and Business Models*, pages 120–134. Springer, 2011.

[121] Wubin Li, Petter Svärd, Johan Tordsson, and Erik Elmroth. Cost-optimal cloud service placement under dynamic pricing schemes. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 187–194. IEEE Computer Society, 2013.

[122] Zhichao Li, Amanpreet Mukker, and Erez Zadok. On the importance of evaluating storage systems' $costs. *USENIX HoStorage*, 2014.

[123] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, and Xiaofei Liao. Performance and energy modeling for live migration of virtual machines. In *HPDC*, 2011.

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

[125] A.A. Mamun, G.X. Guo, and C. Bi. *Hard Disk Drive : Mechatronics and Control*. Automation and Control Engineering. CRC Press, 2006. ISBN 9781420004106. URL https://books.google.fr/books?id=_OHKBQAAQBAJ.

[126] Z. Á. Mann. Multicore-aware virtual machine placement in cloud data centers. *IEEE Transactions on Computers*, 65(11) :3357–3369, Nov 2016.

[127] Zoltán Ádám Mann. Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. *ACM Computing Surveys*, 48, August 2015.

[128] Bo Mao, Hong Jiang, Suzhen Wu, Lei Tian, Dan Feng, Jianxi Chen, and Lingfang Zeng. Hpda : A hybrid parity-based disk array for enhanced

performance and reliability. *ACM Transactions on Storage (TOS)*, 8(1) :4, 2012.

[129] Ali Mashtizadeh, Emré Celebi, Tal Garfinkel, and Min Cai. The design and evolution of live storage migration in vmware esx. In *USENIX Annual Technical Conference*, 2011.

[130] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. The new ext4 filesystem : current status and future plans. In *Proceedings of the Linux symposium*, volume 2, pages 21–33. Citeseer, 2007.

[131] Andréa Matsunaga and José AB Fortes. On the use of machine learning to predict the time and resources consumed by applications. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 495–504. IEEE Computer Society, 2010.

[132] Larry McVoy. Lmbench - tools for performance analysis. http://www.bitmover.com/lmbench/, Visité le 08/06/2016.

[133] GI Meijer. Cooling energy-hungry data centers. *Science*, 328(5976) :318–319, 2010.

[134] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011.

[135] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[136] R. Micheloni, L. Crippa, and A. Marelli. *Inside NAND Flash Memories*. Springer Netherlands, 2010. ISBN 9789048194315. URL https://books.google.fr/books?id=vaq11vKwo_kC.

[137] Inc Micron Technology. Small-block vs. large-block nand flash devices, 2005. URL https://www.micron.com/~/media/documents/products/technical-note/nand-flash/tn2907.pdf.

[138] Inc Micron Technology and Intel. Intel and micron develop the world's fastest nand flash memory with 5x faster performance, 2008. URL http://www.intel.com/pressroom/archive/releases/2008/20080201corp.htm.

[139] Microsoft. Sla for cloud services, Visited in 03/09/2015. URL http://azure.microsoft.com/en-us/support/legal/sla/cloud-services/v1_0/.

[140] Mayank Mishra and Anirudha Sahoo. On theory of vm placement : Anomalies in existing methodologies and their mitigation using a novel vector

based approach. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 275–282. IEEE, 2011.

[141] Al Muller and Seburn Wilson. Virtualization with vmware esx server. 2005.

[142] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. Migrating server storage to ssds : analysis of tradeoffs. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 145–158. ACM, 2009.

[143] Michael Nelson, Beng-Hong Lim, Greg Hutchins, et al. Fast transparent migration for virtual machines. In *USENIX Annual technical conference, general track*, pages 391–394, 2005.

[144] William D Norcott and Don Capps. Iozone filesystem benchmark. *URL : www. iozone. org*, 55, 2003.

[145] Pierre Olivier. *Estimation de performances et de consommation énergétique de systèmes de stockage à base de mémoire flash dans les systèmes embarqués. (Performance and power consumption estimation for embedded flash-based storage systems)*. PhD thesis, University of Southern Brittany, Morbihan, France, 2014.

[146] Pierre Olivier, Jalil Boukhobza, Mathieu Soula, Michelle Le Grand, Ismat Chaib Draa, and Eric Senn. A tracing toolset for embedded linux flash file system. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, pages 153–158. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[147] Openstack. Watcher, Visité en Nov 2016. URL https://wiki.openstack.org/wiki/Watcher.

[148] OpenStack. Hypervisor support matrix, Visité le 03/08/2016.

[149] Oracle. Oracle virtualbox, Visité le 03/08/2016.

[150] J. O'Reilly. *Network Storage : Tools and Technologies for Storing Your Company's Data*. Elsevier Science, 2016. ISBN 9780128038659. URL https://books.google.fr/books?id=Vop4CgAAQBAJ.

[151] Anne-Cecile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Comput. Surv.*, 46(4) :47 :1–47 :31, March 2014.

[152] Hamza Ouarnoughi, Jalil Boukhobza, Frank Singhoff, and Stéphane Rubini. A multi-level I/O tracer for timing and performance storage systems

in iaas cloud. In *3rd IEEE International Workshop on Real-time and distributed computing in emerging applications*, 2014.

[153] Hamza Ouarnoughi, Jalil Boukhobza, Frank Singhoff, and Stéphane Rubini. A cost model for virtual machine storage in cloud iaas context. In *24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP*, pages 664–671, 2016.

[154] Hamza Ouarnoughi, Jalil Boukhobza, Frank Singhoff, Stéphane Rubini, and Erwann Kassis. Considering I/O processing in cloudsim for performance and energy evaluation. In *High Performance Computing - ISC High Performance 2016 International Workshops*, pages 591–603, 2016.

[155] Hamza Ouarnoughi, Jalil Boukhobza, Frank Singhoff, and Stéphane Rubini. Integrating i/os in cloudsim for performance and energy estimation. *SIGOPS Oper. Syst. Rev.*, 50(2) :27–36, January 2017. ISSN 0163-5980.

[156] Edouard Outin, Jean-Emile Dartois, Olivier Barais, and Jean-Louis Pazat. Enhancing cloud energy models for optimizing datacenters efficiency. In *2015 International Conference on Cloud and Autonomic Computing, Boston, MA, USA, September 21-25, 2015*, pages 93–100, 2015.

[157] Edouard Outin, Jean-Emile Dartois, Olivier Barais, and Jean-Louis Pazat. Seeking for the optimal energy modelisation accuracy to allow efficient datacenter optimizations. In *IEEE/ACM 16th International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2016, Cartagena, Colombia, May 16-19, 2016*, pages 535–539, 2016.

[158] Seonyeong Park, Youngjae Kim, Bhuvan Urgaonkar, Joonwon Lee, and Euiseong Seo. A comprehensive study of energy efficiency and performance of flash-based ssd. *JSA*, 2011.

[159] David A Patterson, Garth Gibson, and Randy H Katz. *A case for redundant arrays of inexpensive disks (RAID)*, volume 17. ACM, 1988.

[160] 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), 2016 17th IEEE/ACIS International Conference on*, pages 319–324. IEEE, 2016.

[161] Raritan. Intelligent metered rack power distribution : `http://ram.raritanassets.com/resources/data_sheets/PXE-Data-Sheet.pdf`. Technical report, 2013. URL `http://ram.raritanassets.com/resources/data_sheets/PXE-Data-Sheet.pdf`.

[162] RedHat. libvirt : The virtualization api, Visité le 03/08/2016.

[163] Alma Riska and Erik Riedel. Disk drive level workload characterization. In *USENIX Annual Technical Conference, General Track*, volume 2006, pages 97–102, 2006.

[164] Alma Riska and Erik Riedel. Disk drive workload captured in logs collected during the field return incoming test. In *WASL*, 2008.

[165] Alma Riska and Erik Riedel. Evaluation of disk-level workloads at different time-scales. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 158–167. IEEE, 2009.

[166] S. Rohaut. *Linux : maîtrisez l'administration du système*. Ressources informatiques. Éd. ENI, 2009. ISBN 9782746051287. URL https://books.google.fr/books?id=tFPG5tCA-S4C.

[167] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural Computation*, 16 (5) :1063–1076, 2004.

[168] Chris Ruemmler and John Wilkes. An introduction to disk drive modeling. *Computer*, 27(3) :17–28, 1994.

[169] David Sacks. Demystifying storage networking. *DAS, SAN, NAS, NAS Gateways, Fibre Channel, and iSCSI, IBM Storage Networking,(Jun. 2001)*, 2001.

[170] Bianca Schroeder and Garth A. Gibson. Disk failures in the real world : What does an mttf of 1,000,000 hours mean to you? In *USENIX FAST*, 2007.

[171] Seagate. Barracuda st1000dm003. Technical report, http ://www.seagate.com, Visité en Mar 2016. URL http://www.seagate.com/staticfiles/docs/pdf/datasheet/disc/barracuda-ds1737-1-1111us.pdf.

[172] Amazon Web Services. Amazon ec2 service level agreement, Visited in 03/09/2015. URL https://aws.amazon.com/ec2/sla/.

[173] Amazon Web Services. Description détaillée d'amazon ebs. https://aws.amazon.com/fr/ebs/details/, Visité le 08/06/2016.

[174] S. Sivathanu, Ling Liu, Mei Yiduo, and Xing Pu. Storage management in virtualized cloud environment. In *International Conference on Cloud Computing*, 2010.

[175] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. Extending ssd lifetimes with disk-based write caches. In *FAST*, volume 10, pages 101–114, 2010.

[176] Jan Stoess, Christian Lang, and Frank Bellosa. Energy management for hypervisor-based virtual machines. In *USENIX Annual Technical Conference*, 2007.

[177] A. Strunk and W. Dargie. Does live migration of virtual machines cost energy ? In *International Conference on Advanced Information Networking and Applications*, 2013.

[178] Seiichi Sugaya. Trends in enterprise hard disk drives. *Fujitsu Sci. Tech. J*, 42(1) :61–71, 2006.

[179] Arie Tal. Two flash technologies compared : Nor vs nand. *White Paper of M-SYstems*, 2002.

[180] Hana Teyeb, Ali Balma, Samir Tata, and Nejib Ben Hadj-Alouane. Traffic-aware virtual machine migration scheduling problem in geographically distributed data centers. In *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*, pages 798–801. IEEE, 2016.

[181] Johan Tordsson, Rubén S Montero, Rafael Moreno-Vozmediano, and Ignacio M Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems*, 28(2) :358–367, 2012.

[182] G. S. Tran, A. Tchana, D. Hagimont, and N. D. Palma. Cooperative resource management in a iaas. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pages 611–618, March 2015.

[183] Hai Nam TRAN. *Cache Memory Aware Priority Assignment and Scheduling Simulation of Real-Time Embedded Systems*. PhD thesis, University of West Brittany, Brest, France, 2017.

[184] Theodore Ts'o. The ext2fs library, 2005. URL `http://www.giis.co.in/libext2fs.pdf`.

[185] Ruben Van den Bossche, Kurt Vanmechelen, and Jan Broeckhove. Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 228–235. IEEE, 2010.

[186] Rik van Riel and Peter W. Morreale. Documentation for the linux kernel. `https://www.kernel.org/doc/Documentation/sysctl/vm.txt`, Visité le 08/09/2016.

[187] Tarasov Vasily, Zadok Erez, and Shepler Spencer. Filebench : a flexible framework for file system benchmarking. *USENIX ;login : magazine Spring*, 41(1) :6–12, 2016.

[188] Anthony Velte and Toby Velte. *Microsoft virtualization with Hyper-V*. McGraw-Hill, Inc., 2009.

[189] Akshat Verma, Gargi Dasgupta, Tapan Kumar Nayak, Pradipta De, and Ravi Kothari. Server workload analysis for power minimization using consolidation. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, pages 28–28. USENIX Association, 2009.

[190] J. Waller. *Performance Benchmarking of Application Monitoring Frameworks*. Books on Demand, 2015. ISBN 9783738689310. URL https://books.google.fr/books?id=qsErBgAAQBAJ.

[191] Lipeng Wan, Zheng Lu, Qing Cao, Feiyi Wang, Sarp Oral, and Bradley Settlemyer. Ssd-optimized workload placement with adaptive learning and classification in hpc environments. In *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–6. IEEE, 2014.

[192] Brian Ward. *The book of VMware : the complete guide to VMware workstation*, volume 1. No Starch Press San Francisco, 2002.

[193] Tom White. *Hadoop : The definitive guide*. " O'Reilly Media, Inc.", 2012.

[194] Lars Wirzenius and Joanna Oja. The linux system administrators' guide. *versión 0.6*, 2, 1993.

[195] Lars Wirzenius, Joanna Oja, Stephen Stafford, and Alex Weeks. Linux system administrator's guide. version 0.9.

[196] Bruce L Worthington, Gregory R Ganger, Yale N Patt, and John Wilkes. *On-line extraction of SCSI disk drive parameters*, volume 23. ACM, 1995.

[197] Jing Xu and Jose AB Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 179–188. IEEE, 2010.

[198] Darrell D. E. Long Yan Li. Which storage device is the greenest ? modeling the energy cost of i/o workloads. *IEEE MASCOTS*, 2014.

[199] Kim Youngjae, Gupta Aayush, Urgaonkar Bhuvan, Berman Piotr, and Sivasubramaniam Anand. Hybridplan : a capacity planning technique for projecting storage requirements in hybrid storage systems. *The Journal of Supercomputing*, 2014.

[200] Lamia Youseff, Rich Wolski, Brent Gorda, and Chandra Krintz. *Paravirtualization for HPC Systems*, pages 474–486. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-49862-9. doi : 10.1007/11942634_49. URL http://dx.doi.org/10.1007/11942634_49.

[201] P. Zanella, Y. Ligier, and E. Lazard. *Architecture et technologie des ordinateurs : cours et exercices corrigés*. Sciences SUP. : Informatique. Dunod, 2013. ISBN 9782100598182. URL https://books.google.fr/books?id=eTMinwEACAAJ.

[202] Ning Zhang, Junichi Tatemura, Jignesh M Patel, and Hakan Hacigümüş. Towards cost-effective storage provisioning for dbmss. *Proceedings of the VLDB Endowment*, 5(4) :274–285, 2011.

[203] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing : state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1) : 7–18, 2010.

[204] Rui Zhang, Ramani Routray, David Eyers, David Chambliss, Prasenjit Sarkar, Douglas Willcocks, and Peter Pietzuch. Io tetris : Deep storage consolidation for the cloud via fine-grained workload analysis. In *International Conference on Cloud Computing*, 2011.

[205] Ruijin Zhou, Fang Liu, Chao Li, and Tao Li. Optimizing virtual machine live storage migration in heterogeneous storage environment. In *International Conference on Virtual Execution Environments*, 2013.