Ordonnancement des migrations à chaud de machines virtuelles

Vincent Kherbache

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Domaine : INFORMATIQUE

Ordonnancement des migrations à chaud de machines virtuelles

Présentée et soutenue publiquement par

Vincent KHERBACHE

Thèse préparée à
Inria Sophia Antipolis
et soutenue le 07 Décembre 2016.

Jury :

Adrien LÈBRE	Maître de conférence	Inria / École des Mines de Nantes	Examinateur
Christine MORIN	Directrice de recherche	IRISA / Inria Rennes	Rapportrice
Éric MADELAINE	Chargé de recherche	Inria Sophia Antipolis	Dir. de thèse / Invité
Fabien HERMENIER	Maître de conférence	I3S / Univ. Nice Sophia Antipolis	Resp. scientifique
Georges DA COSTA	Maître de conférence	IRIT (Toulouse)	Rapporteur
Guillaume PIERRE	Professeur	ISTIC / Univ. de Rennes 1	Examinateur
Guillaume URVOY-KELLER	Professeur	I3S / Univ. Nice Sophia Antipolis	Examinateur

Ordonnancement des migrations à chaud de machines virtuelles.

Vincent Kherbache

Migrer à chaud une machine virtuelle (VM) est une opération basique dans un centre de données. Tous les jours, des VMs sont migrées pour répartir la charge, économiser de l'énergie ou préparer la maintenance de serveurs. Bien que les problèmes de placement des VMs soient beaucoup étudiés, on observe que la gestion des migrations permettant de transiter vers ces nouveaux placements reste un domaine de second plan. Cette phase est cependant critique car chaque migration à un coût en terme de CPU, de bande passante et d'énergie. Des algorithmes de décision reposent alors sur des hypothèses irréalistes et calculent des ordonnancements conduisant à des migrations inutilement longues et incontrôlables qui réduisent les bénéfices attendus de la ré-organisation des VMs.

Dans cette thèse nous nous sommes fixé comme objectif d'améliorer la qualité des ordonnancements de migrations dans les centres de données. Pour cela, nous avons d'abord modélisé l'ordonnancement de migrations en considérant l'architecture réseau et l'activité mémoire des VMs. Pour évaluer l'efficacité de notre modèle, nous avons ensuite implémenté un ordonnanceur de migrations au sein du gestionnaire de VMs BtrPlace. Nous avons ensuite étendu notre ordonnanceur en développant des contraintes d'ordonnancement, des objectifs personnalisés, une heuristique de recherche ainsi qu'un modèle énergétique.

Nous avons validé notre approche par l'étude pratique de scénarios d'ordonnancement réalisés en environnement réel. Nous avons ainsi pu analyser la précision de notre modèle de migration, valider la qualité des décisions prises par notre modèle d'ordonnancement et évaluer l'extensibilité ainsi que le passage à l'échelle de notre solution.

Mots clés : Virtualisation, migration à chaud, ordonnancement.

Live-migrations scheduling of virtual machines.

Vincent Kherbache

A live-migration of a virtual machine (VM) is a basic operation in a data center. Every day, VMs are migrated to distribute the load, save energy or prepare maintenance operations on production servers. Although VM placement problems have been extensively studied, we observe that the migrations management needed to apply these new placements did not get much attention. This phase is however critical as each migration has a cost in terms of CPU, bandwidth and energy. Decision algorithms are thus based on unrealistic assumptions and compute schedules which can lead to unnecessarily long and uncontrollable migrations. This reduces the ultimate benefits expected from the VMs re-organization.

In this thesis, our main objective is to improve the efficiency of live-migrations scheduling within data centers. To achieve our goal, we have first modeled the scheduling of live migrations based on the network architecture and the VMs memory activity. To evaluate the efficiency of our model, we have then implemented and optimized a migrations scheduler within the VMs manager BtrPlace. We have then extended our scheduler by developing scheduling constraints, custom objectives, a search heuristic and an energy model.

We have validated our approach by the practical study of many scheduling scenarios executed in a real environment. We have then analyzed the accuracy of our migration model, assessed the quality of the decisions taken by our scheduling model, and evaluated the extensibility and the scalability of our solution.

Keywords: Virtualization, live-migration, scheduling.

Remerciements

J'exprime tout d'abord mes remerciements à l'ensemble des membres du jury pour avoir accepté de participer à la concrétisation de mon doctorat.

Je tiens à remercier mon directeur de thèse Monsieur Éric Madelaine pour m'avoir permis de réaliser cette thèse et pour avoir toujours été attentif et disponible malgré ses nombreuses charges de travail.

Je remercie tout particulièrement mon responsable scientifique Monsieur Fabien Hermenier pour tout ce qu'il m'a apporté et pour son soutien sans faille tout au long de cette thèse. Je pense notamment aux nombreuses heures passées à corriger mon anglais et à sa manière à lui de me rappeler précisément le contexte de la thèse après une nuit blanche d'expérimentations.

Je remercie chaleureusement toutes les personnes qui m'ont aidé pendant l'élaboration de ma thèse et en particulier Fabrice Huet et Ludovic Henrio qui, par leur nombreux conseils et leur humour bien à eux, m'ont permis d'appréhender avec sérénité les challenges techniques mais aussi mathématiques de cette thèse.

Ce travail n'aurait pas été possible sans le financement apporté par le projet européen DC4Cities ainsi que la mise à disposition de la plateforme expérimentale Grid'5000 dont je remercie les administrateurs pour avoir toujours répondu à mes demandes techniques.

Au terme de ce parcours, je remercie enfin celles et ceux qui me sont chers et que j'ai quelque peu délaissés ces derniers mois pour achever cette thèse. Leurs attentions et encouragements m'ont accompagné tout au long de ces années. Je suis redevable à mes parents, Monique et Jamel Kherbache, pour leur soutien moral et matériel et leur confiance indéfectible dans mes choix.

Enfin, j'ai une pensée particulière pour ma conjointe, Anne-Laure Lupa, dont la patience est restée intacte après toutes ces années de vie commune.

Table des matières

1 Introduction **1**

1.1 Objectifs . . . 3
1.2 Contributions . . . 4
1.2.1 Modélisation de l'ordonnancement de migrations . . . 4
1.2.2 Implémentation d'un ordonnanceur de migrations . . . 5
1.2.3 Applications et évaluation . . . 5
1.3 Organisation du document . . . 6
1.4 Diffusion scientifique . . . 7
1.5 Diffusion logicielle et reproductibilité . . . 8

I État de l'art **9**

2 Migration à chaud de machines virtuelles **11**

2.1 Protocoles de migration à chaud . . . 12
2.1.1 Algorithme de pré-copie . . . 13
2.1.2 Algorithme de post-copie . . . 14
2.1.3 Vers de la post-copie hybride . . . 16
2.2 Implémentation dans les hyperviseurs . . . 17
2.2.1 Détails d'implémentation . . . 17
2.2.2 Amélioration du processus de migration . . . 20
2.3 Virtualisation et migration du stockage . . . 21
2.3.1 Virtualisation du stockage . . . 22
2.3.2 Migration de stockage virtualisé . . . 22
2.4 Modèles de performance . . . 23
2.4.1 Modèles linéaires . . . 24
2.4.2 Fréquence d'écriture des pages mémoire . . . 25
2.5 Discussion . . . 26

3 Ordonnancement des migrations **29**

3.1 Principes d'ordonnancement . . . 30
3.1.1 Objectifs et contraintes . . . 30
3.1.2 Critères de performance . . . 31
3.2 Gestion du réseau . . . 32
3.2.1 Architecture réseau . . . 32
3.2.2 Réseau de migration . . . 35
3.3 État de l'art . . . 36

3.3.1 Techniques d'ordonnancement 36
3.3.2 Optimisation des migrations 39
3.3.3 Contrôle de l'ordonnancement 41
3.4 Discussion 43

II Contributions 45

4 Modèle d'ordonnancement 47
4.1 Modèle de migration à chaud 47
4.1.1 Consommation mémoire des hôtes 48
4.1.2 Activité mémoire des VMs 49
4.2 Modèle réseau 55
4.2.1 Description du modèle 55
4.2.2 Bande passante et parallélisation 56
4.3 Résolution du problème d'ordonnancement 59
4.3.1 Gestion de projet à contraintes de ressources 59
4.3.2 Multi-mode RCPSP 59
4.4 Conclusion 60

5 Implémentation dans BtrPlace 63
5.1 BtrPlace 64
5.1.1 La programmation par contraintes 64
5.1.2 Modélisation du problème de reconfiguration 65
5.1.3 Objectif de la résolution 65
5.2 Implémentation de l'ordonnancement 66
5.2.1 Modèle réseau 66
5.2.2 Modèle de migration 68
5.2.3 Configuration et application de l'ordonnancement . . . 70
5.3 Optimisations 71
5.3.1 Bande passante maximale 71
5.3.2 Heuristique 72
5.4 Extensions 73
5.4.1 Algorithme de post-copie 73
5.4.2 Contrôle temporel 74
5.4.3 Modèle énergétique 76
5.5 Conclusion 78

6 Évaluation 79
6.1 Configuration expérimentale 80
6.2 Décisions d'ordonnancement 81
6.2.1 Parallélisation optimale 81

6.2.2 Efficacité du modèle d'ordonnancement 84
6.3 Précision du modèle de migration 89
6.3.1 Modèles de comparaison 89
6.3.2 Analyse des erreurs de prédiction 90
6.3.3 Analyse de la déviation des migrations 92
6.4 Efficacité énergétique 93
6.4.1 Économie d'énergie 94
6.4.2 Évaluation de la limitation de puissance 96
6.5 Passage à l'échelle de l'ordonnanceur 97
6.5.1 Configuration expérimentale 97
6.5.2 Optimisation 98
6.5.3 Surcoût du nouveau modèle d'ordonnancement 101
6.6 Conclusion 103

7 Conclusion **105**
7.1 Résumé 105
7.2 Perspectives 107
7.2.1 Extensions réseaux 108
7.2.2 Convergence du placement et de l'ordonnancement . . 109
7.2.3 Viabilité de l'approche à long terme 109

Table des figures

2.1 Algorithme de pré-copie. 13
2.2 Algorithme de post-copie. 15
2.3 Algorithme de post-copie hybride. 16

3.1 Architecture réseau hiérarchique. 33

4.1 Aperçu de la consommation mémoire de deux hôtes physique (source et destination) lors de la migration parallèle de 4 VMs. La Figure (b) est une vue rapprochée de la consommation mémoire de l'hôte source lorsque les 4 migrations débutent. . . . 49
4.2 Activité mémoire générée par *httperf* en 30 ms. 51
4.3 Activité mémoire générée par *ab*. 52
4.4 Durées de migration à chaud entre 2 hyperviseurs KVM. . . . 56
4.5 Exemple d'un scénario de décommissionnement. 57
4.6 Ordonnancement idéal pour le scénario en Figure 4.5. 58

6.1 Configuration expérimentale. 81
6.2 Ordonnancement utilisant l'ancien modèle. 82
6.3 Ordonnancement utilisant le nouveau modèle. 83
6.4 Durées de migration. 85
6.5 Temps de complétion. 87
6.6 Déviation normalisée des durées de migration par rapport aux exécutions réelles. 91
6.7 Déviation absolue des durées de migration par rapport aux exécutions réelles. 92
6.8 Puissance instantanée consommée. 94
6.9 Limitation de la puissance instantanée consommée. 96
6.10 Comparaison des durées de résolution de BtrPlace en fonction de l'activation de l'optimisation *MaxBandwidth* et du facteur d'échelle. Lorsque l'optimisation est désactivée, 4 bandes passantes différentes sont disponibles par migration. 99
6.11 Comparaison des durées de résolution du nouveau modèle d'ordonnancement de BtrPlace par rapport à son modèle d'origine et du facteur d'échelle. 102

Liste des tableaux

6.1 Durées de migration absolues et ralentissement relatif à un ordonnancement séquentiel. 85
6.2 Temps de complétion absolus et accélération relative à un ordonnancement séquentiel. 87
6.3 Déviation normalisée des durées de migration par rapport aux exécutions réelles. 92
6.4 Calibration du modèle énergétique 94
6.5 Pourcentages d'instances résolues en moins d'une minute en fonction de l'activation de l'optimisation *MaxBandwidth* et du facteur d'échelle. 100

Chapitre 1

Introduction

Sommaire

1.1 Objectifs 3
1.2 Contributions 4
1.2.1 Modélisation de l'ordonnancement de migrations . . . 4
1.2.2 Implémentation d'un ordonnanceur de migrations . . . 5
1.2.3 Applications et évaluation 5
1.3 Organisation du document 6
1.4 Diffusion scientifique 7
1.5 Diffusion logicielle et reproductibilité 8

Commercialisés dès le début des années 60, les premiers ordinateurs dits *centraux* (*main frames*) sont principalement destinés aux très grandes entreprises (banques, compagnies aériennes, etc.) et aux centres de calcul scientifique. Pour exploiter au mieux la capacité de ces ordinateurs, la *virtualisation* fait son apparition et divise de manière logique les ressources système entre différentes applications pour pouvoir les exécuter de façon concurrente. Au début des années 90 l'avancée technologique rend les ordinateurs centraux obsolètes (mauvais rapport prix/performances), ces derniers sont alors peu à peu remplacés par de multiples micro-ordinateurs plus compacts que l'on nomme *serveurs*. Les pièces dans lesquelles sont stockés ces serveurs se font également connaître sous le nom de *centres de données*. Depuis lors, la virtualisation ainsi que les centres de données ont considérablement évolué.

La montée d'Internet au milieu des années 90 pousse les entreprises à disposer de connections rapides pour y réaliser des opérations permanentes et de nombreuses entreprises décident alors de construire leur propre centre de données. En accumulant de plus en plus de données qu'elles doivent stocker et protéger, les entreprises font rapidement face à l'augmentation de la taille de leurs infrastructures qui nécessitent désormais des moyens de refroidissement conséquents et entraînent des consommations énergétiques élevées. L'entretien, la maintenance et la surveillance requièrent également des moyens logistiques et financiers considérables. De plus, l'évolution rapide des technologies et du matériel (puissance de calcul, capacité de stockage et vitesse de

transfert accrues) rendent la gestion des centres de données compliquée. Le matériel devenant rapidement obsolète, la durée de vie d'un centre de données est estimée à environ 25 ans.

Depuis les années 2000, on assiste à une révolution des technologies de virtualisation et à l'émergence des centres de données virtualisés. Initialement destinée à exécuter plusieurs applications sur un même ordinateur, la notion de virtualisation s'est élargie. Elle réfère désormais à la simulation complète de composants physiques et à l'exécution de *machines virtuelles* (VMs) agissant comme de vrais ordinateurs autonomes avec leur propre système d'exploitation. Un programme, appelé *hyperviseur*, est alors exécuté sur les machines physiques pour imiter l'ensemble des composants matériels d'un ordinateur et permettre l'hébergement d'une ou plusieurs VMs [Kiv+07 ; Bar+03]. Les applications exécutées sur les VMs sont ainsi séparées des ressources matérielles sous-jacentes. Les avantages apportés par la virtualisation sont nombreux, parmi les critères les plus importants on peut relever la flexibilité apportée par l'indépendance matérielle des VMs, les facilités de gestion tel que la possibilité de réplication d'une VM ou encore la sécurité apportée par l'isolation par rapport au système hôte.

Grâce aux techniques de virtualisation, les centres de données évoluent désormais depuis un modèle basé sur la propriété des infrastructures, du matériel et des logiciels, vers un modèle d'abonnement et de ressources à la demande. Ce changement de paradigme, communément appelé « *informatique dans les nuages* »*(cloud computing)*, s'est très rapidement répandu et on distingue aujourd'hui les infrastructures de *nuages* privés et publiques. Dans un *nuage* privé, une entreprise dispose d'une quantité de ressources restreinte qu'elle utilise pour ses propres besoins. La virtualisation permet d'améliorer la gestion de l'infrastructure et d'optimiser l'utilisation des ressources, par exemple via le déploiement de VMs à la demande pour réaliser des tâches récurrentes ou exceptionnelles. Les *nuages* publiques, quant à eux, sont gérés par des entreprises disposant généralement d'une très grande quantité de ressources qu'ils proposent à la location, par le biais de la virtualisation, à des entités externes. Ainsi, les entreprises qui n'ont pas les moyens d'investir dans un centre de données par exemple peuvent louer des ressources de calcul, de réseau et de stockage à la demande depuis des centres de données distants via Internet. Les applications et les données de ces entreprises se situent alors dans un *nuage* composé de machines distantes interconnectées et parfois réparties géographiquement. Le problème de confidentialité des données reste toutefois un frein majeur à l'utilisation de *nuages* publiques pour les entreprises disposant de données sensibles et privées.

De nos jours, la virtualisation et l'émergence de l'*infrastructure dirigée par le logiciel* (SLI) aident les centres de données à devenir de plus en plus efficaces,

en fournissant des services critiques d'entreprise dans le monde entier. Apporter les avantages de la virtualisation à chaque couche du centre de données est l'une des principales caractéristiques du SLI. Dès lors, la pile complète de distribution des applications bénéficie de la disponibilité, la flexibilité et l'automatisation apportés par la virtualisation d'une manière qui non seulement maximise, mais augmente les ressources disponibles. En effet, la virtualisation offre de nouvelles opportunités de consolidation des applications qui peuvent désormais s'abstraire des contraintes matérielles via la migration à chaud des machines virtuelles [Cla+05]. Cette technique permet désormais de déplacer les applications de manière quasi-transparente entre les différents serveurs physiques d'un centre de données. La consolidation dynamique des machines virtuelles est aujourd'hui constamment utilisée pour maximiser la performance des applications, optimiser l'utilisation des ressources physiques, réduire la consommation énergétique des infrastructures mais également pour réaliser des opérations de maintenance [Her+09 ; VAN08 ; BB10 ; LQL12 ; VBJ14].

Dans la pratique, un gestionnaire de VMs calcule un nouveau placement pour certaines VMs qui satisfasse un objectif spécifique (performance, énergie, etc.) puis un ordonnancement indiquant quand lancer chaque migration. Bien que les problèmes de placement des VMs soient beaucoup étudiés, on observe que la gestion des migrations permettant de transiter vers ces nouveaux placements reste un domaine de second plan [VBJ14]. Cette phase est cependant critique car chaque migration à un coût en terme de CPU, de bande passante et d'énergie. Des algorithmes de décision reposent alors sur des hypothèses irréalistes et calculent des ordonnancements conduisant à des migrations inutilement longues et incontrôlables qui réduisent finalement les bénéfices attendus de la ré-organisation des VMs. De manière générale, avec l'augmentation du nombre de serveurs et de VMs par serveur, planifier efficacement de nombreuses migrations entre de multiples serveurs devient difficile [SA10].

1.1 Objectifs

L'objectif fixé dans cette thèse est d'améliorer la qualité des ordonnancements de migrations dans les centres de données. Pour mener à bien cet objectif, il est d'abord nécessaire de comprendre et de maîtriser les différents paramètres qui influent sur la durée et la faisabilité de la migration à chaud d'une machine virtuelle. En effet, établir un ordre optimal des migrations est une opération complexe qui nécessite une connaissance approfondie de l'infrastructure des centres de données. L'ordre d'exécution des migrations doit par exemple tenir compte de la disponibilité des ressources réseau, du temps à allouer à l'opération et de l'interruption occasionnée aux VMs. Des com-

promis doivent alors être effectués en fonction des besoins et contraintes de l'administrateur mais également de l'utilisation des VMs. L'ordonnancement de plusieurs migrations doit également obéir à des règles de parallélisation qu'il s'agit de déterminer et de généraliser. L'objectif étant de produire des ordonnancements efficaces indépendamment des contraintes logicielles et matérielles du centre de données. Nous pensons qu'il est nécessaire de fournir aux administrateurs des moyens simples pour générer des ordonnancement fiables et efficaces qui tiennent compte des particularités de leur infrastructure. Chaque administrateur ayant des besoins spécifiques et des contraintes particulières, il est important de disposer d'un outil flexible et extensible pour pouvoir contrôler finement l'ordonnancement, au cas par cas.

1.2 Contributions

Ce document décrit nos contributions dans le domaine de l'ordonnancement des migrations à chaud de machines virtuelles. Trois axes principaux résultent de nos travaux : la modélisation de l'ordonnancement de migrations à chaud reposant sur un modèle réaliste de l'architecture réseau et de l'activité mémoire des VMs ; l'implémentation d'un ordonnanceur de migrations au sein du gestionnaire de VMs BtrPlace [HLM13] et la définition de contraintes d'ordonnancement temporelles et énergétiques ; l'étude pratique de scénarios d'ordonnancement pour valider les décisions prises par notre modèle d'ordonnancement, analyser la précision de notre modèle de migration ainsi qu'évaluer l'extensibilité et le passage à l'échelle de notre solution.

1.2.1 Modélisation de l'ordonnancement de migrations

Nous montrons dans un premier temps l'intérêt de considérer l'activité mémoire des machines virtuelles et l'architecture réseau du centre de données pour estimer précisément les durées de migration et gérer le partage des ressources réseau dans le temps. Nous proposons ensuite un modèle d'ordonnancement reposant sur un modèle de migration précis et un modèle réseau réaliste pour déterminer l'ordre des migrations et la bande passante à leur allouer.

Pour déterminer les règles d'ordonnancement de migrations, nous étudions le comportement des migrations dans différentes situations représentatives [KMH14]. Nous évaluons ainsi l'impact du partage de bande passante sur la durée des migrations, analysons les opportunités de parallélisation en fonction de l'architecture réseau et déduisons les critères de groupement des migrations pour optimiser l'ordonnancement. Nous analysons finalement le

problème d'ordonnancement des migrations et déduisons la complexité de sa résolution.

1.2.2 Implémentation d'un ordonnanceur de migrations

Après avoir discuté de la nécessité de disposer d'un outil flexible et autonome pour calculer l'ordonnancement des migrations, nous décidons de nous appuyer sur le gestionnaire de VMs existant BtrPlace. Ce dernier se sert de la programmation par contraintes pour modéliser le placement des VMs sur les hôtes et calculer des plans de reconfiguration de centres de données.

En implémentant notre modèle d'ordonnancement au sein de BtrPlace nous découvrons d'abord une optimisation qui réduit la complexité du problème en pré-calculant la durée des migrations. Nous développons ensuite une heuristique spécifique à l'ordonnancement de migrations qui permet de mener rapidement le solveur vers des solutions de qualité. Nous développons également des contraintes permettant d'agir sur les décisions d'ordonnancement de BtrPlace en imposant par exemple la parallélisation ou la séquentialisation d'un ensemble de VMs. En s'appuyant sur les capacités d'extensibilité de BtrPlace, nous implémentons un algorithme de migration à chaud alternatif ainsi qu'un modèle et des contraintes énergétiques permettant d'agir sur l'ordonnancement en fonction de la consommation énergétique des actions de reconfigurations.

1.2.3 Applications et évaluation

Afin de valider notre approche, nous réalisons une série d'expérimentations représentatives en environnements réel et simulé. Nous évaluons les décisions d'ordonnancement prises par notre modèle à partir de plans de migrations aléatoires et comparons les durées de migration ainsi que les temps de complétion obtenus par rapport à deux ordonnanceurs existants. Nous validons également la précision de notre modèle en comparant l'exécution de plans de migrations aléatoires en simulation par rapport à un environnement réel, et évaluons les résultats obtenus par rapport à trois autres modèles de migration représentatifs des solutions de prédiction actuelles. La précision moyenne de notre modèle de migration est ainsi évaluée à 93,9% et affiche de meilleurs résultats que les trois autres modèles.

Nous validons l'intérêt pratique du modèle énergétique via des expérimentations à large échelle reproduisant des opérations de maintenance qui nécessitent le dé-commissionnement d'un grand nombre d'hôtes. L'intégration du modèle énergétique permet de réduire la consommation des ordonnancements calculés en optimisant les moments de démarrage et d'extinction des hôtes par

exemple. Lors de l'exécution d'un scénario de décommissionnement à grande échelle, notre modèle réduit de 21,5% la consommation énergétique par rapport à la version originale de BtrPlace. Nous validons également la capacité de BtrPlace à respecter des contraintes énergétiques en ajustant automatiquement le parallélisme et en reportant certaines actions de reconfigurations.

Finalement, nous évaluons le passage à l'échelle de notre solution qui révèle une capacité de calcul confortable et des durées de résolution très satisfaisantes. Les temps de calcul représentent environ 1% des durées d'exécution des ordonnancements générés.

1.3 Organisation du document

Ce document est structuré en deux parties. La première partie dresse un état de l'art global relatif à la migration à chaud de machines virtuelles et à l'ordonnancement de migrations multiples. L'analyse de l'état de l'art sur la migration à chaud au chapitre 2 exhibe la difficulté de généralisation des modèles de performance et souligne le besoin de définir un modèle analytique simple basé sur une analyse rapide de l'activité mémoire des VMs pour réaliser des prédictions rapides et précises des durées de migration à chaud. L'analyse de la problématique et des études portant sur l'ordonnancement des migrations au chapitre 3 démontre l'importance de considérer l'infrastructure réseau dans sa globalité pour définir des règles d'ordonnancement qui assurent une utilisation efficace du réseau tout en minimisant les durées de migration.

La seconde partie est dédiée à la présentation des différents éléments de contribution de cette thèse. Dans le chapitre 4, nous proposons un modèle d'ordonnancement qui repose à la fois sur un modèle de migration à chaud permettant de prédire rapidement la durée des migrations et sur un modèle réseau permettant de gérer le partage des ressources réseau dans le temps. Dans le chapitre 5, nous détaillons l'implémentation de notre modèle d'ordonnancement au sein du gestionnaire de VMs BtrPlace ainsi que les différentes extensions réalisées pendant cette thèse. Dans le chapitre 6, nous évaluons notre ordonnanceur de migrations au travers d'une série d'expérimentations réalisées en environnements réel et simulé. Nous validons ainsi la qualité des décisions d'ordonnancement, la précision de notre modèle de migration, l'extensibilité de notre approche et les performances observées lors du passage à l'échelle de scénarios de migration. Finalement, nous concluons ce manuscrit au chapitre 7 en réalisant d'abord un bilan de cette thèse puis en discutant de différentes perspectives.

1.4 Diffusion scientifique

Les travaux présentés dans cette thèse ont fait l'objet de plusieurs publications listées ci-dessous :

Conférence internationale avec comité de lecture :

[KMH15c] Vincent Kherbache, Eric Madelaine et Fabien Hermenier. "Scheduling Live-Migrations for Fast, Adaptable and Energy-Efficient Relocation Operations." In : *UCC'15*. Sous la dir. d'Ioan Raicu, Omer F. Rana et Rajkumar Buyya. Limassol, Chypres : IEEE, déc. 2015, p. 205–216

Conférence nationale avec comité de lecture :

[KMH15b] Vincent Kherbache, Eric Madelaine et Fabien Hermenier. "Ordonnancement contrôlé de migrations à chaud". In : *Compas'15*. Lille, France, juil. 2015

Atelier international avec comité de lecture :

[KMH14] Vincent Kherbache, Eric Madelaine et Fabien Hermenier. "Planning Live-Migrations to Prepare Servers for Maintenance." In : *Euro-Par : Parallel Processing - Workshops*. Sous la dir. de Luís Lopes et al. T. 8806. Lecture Notes in Computer Science. Springer, août 2014, p. 498–507

Chapitre de livre :

[DHK15] Sophie Demassey, Fabien Hermenier et Vincent Kherbache. "Optimized Packings with Applications". In : sous la dir. de Giorgio Fasano et D. János Pintér. Cham : Springer International Publishing, 2015. Chap. Dynamic Packing with Side Constraints for Datacenter Resource Management, p. 19–35

Article de journal :

[Mic+16] Étienne Michon, Julien Gossa, Stéphane Genaud, Léo Unbekandt et Vincent Kherbache. "Schlouder : A broker for IaaS clouds". In : *Future Generation Computer Systems* (2016)

Poster :

[KMH15a] Vincent Kherbache, Eric Madelaine et Fabien Hermenier. *Memory and Network Aware Scheduling of Virtual Machine Migrations*. EuroSys'15. Poster. Avr. 2015

1.5 Diffusion logicielle et reproductibilité

Les contributions au gestionnaire de VMs BtrPlace [1] présentées dans cette thèse ont été officiellement intégrées au code source de BtrPlace et sont publiées sous la licence publique générale limitée GNU (GNU LGPL) [2].

L'ensemble des expérimentations réalisées (décrites au Chapitre 6) sont reproductibles, tout les éléments et informations nécessaires à leur reproduction sont disponibles dans un dépôt publique du projet GitHub officiel de BtrPlace [3].

1. http://www.btrplace.org
2. https://github.com/btrplace/scheduler
3. https://github.com/btrplace/kherbache-thesis

Première partie

État de l'art

Chapitre 2

Migration à chaud de machines virtuelles

Sommaire

2.1 Protocoles de migration à chaud 12
2.1.1 Algorithme de pré-copie 13
2.1.2 Algorithme de post-copie 14
2.1.3 Vers de la post-copie hybride 16
2.2 Implémentation dans les hyperviseurs 17
2.2.1 Détails d'implémentation 17
2.2.2 Amélioration du processus de migration 20
2.3 Virtualisation et migration du stockage 21
2.3.1 Virtualisation du stockage 22
2.3.2 Migration de stockage virtualisé 22
2.4 Modèles de performance 23
2.4.1 Modèles linéaires 24
2.4.2 Fréquence d'écriture des pages mémoire 25
2.5 Discussion 26

La migration des machines virtuelles entre les différents hôtes physiques est rapidement devenu incontournable dans les centres de données. Les capacités d'hébergement des serveurs devenant de plus en plus élevées, les besoins de consolidation et de répartition de charge ont suscité un fort intérêt pour la migration des VMs. En offrant la possibilité de basculer dynamiquement des charges de travail entre les serveurs, la migration est quotidiennement utilisée pour améliorer la performance des applications, réduire la consommation énergétique des centres de données ou encore pour préparer des tâches de maintenance sur des serveurs en production. L'inconvénient majeur de la migration réside dans la nécessité d'arrêter ou de mettre en pause la VM pendant toute la durée de son déplacement. Les applications et services qui s'exécutent sur les VMs sont généralement critiques et il est donc difficilement envisageable de les arrêter même pendant une courte période. Lorsqu'une VM

est suspendue, ses connections réseaux actives sont arrêtées et la durée d'interruption nécessaire à sa migration est trop élevée pour permettre la reprise des connexions dans un état cohérent. Il est ainsi généralement nécessaire de redémarrer entièrement les services s'exécutant sur la VM pour pouvoir reprendre un fonctionnement normal. C'est pourquoi de nouvelles stratégies de migration dîtes « *à chaud* » [Cla+05] induisant une durée d'interruption suffisamment faible pour être négligeable ont été mises au point.

Dans ce chapitre nous dressons un état de l'art relatif au processus de migration à chaud. Après une description des différents protocoles de migration et de leur implémentation dans les hyperviseurs, nous décrivons les particularités de la gestion du stockage pour la migration. Nous présentons enfin les différents modèles de performances utilisés pour estimer la durée des migrations.

2.1 Protocoles de migration à chaud

La migration à chaud [Cla+05] consiste à déplacer une VM en cours d'exécution depuis son hôte physique vers un autre tout en assurant une durée d'interruption minimale de la machine virtuelle. Les VMs fournissent une plateforme naturelle pour la migration en encapsulant toutes les données ainsi que les états logiciels et matériels nécessaires à leur fonctionnement. La migration implique une transmission sur le réseau de l'ensemble de ces informations caractérisant l'état de la VM au travers d'une connexion TCP.

Parmi ces informations on distingue deux types de données nécessaires à transmettre indépendamment de la configuration de la VM. Premièrement, la mémoire vive utilisée par la machine virtuelle représente la majeure partie des données à transférer vers l'hôte de destination. Le second type de données à transférer pour pouvoir ré-activer la machine virtuelle dans son état d'origine sont l'état de son ou ses CPU(s) virtuel(s), les paramètres des interfaces réseaux et le détail des connexions actives. Dans le reste de ce document nous nommerons ces informations les *données annexes* car elles sont nombreuses mais ne représentent qu'une faible proportion de la quantité totale des données à transmettre. D'autres informations tels que les données des systèmes de fichier, aussi appelée mémoire de masse, peuvent être quant à elles partagées avec l'hôte de destination. Il n'est donc pas toujours nécessaire de les transférer durant le processus de migration à chaud.

Le coût de la migration à chaud, du point de vue de l'administrateur de la VM, est ainsi principalement lié à l'utilisation des ressources réseaux du centre de données. Du point de vue de l'utilisateur final de la VM, une diminution temporaire de performance est le principal coût imposé par la

migration à chaud. Cette perte de qualité de service est matérialisée par une faible durée d'interruption de la VM ainsi que d'une possible diminution de la bande passante réseau allouée pendant la durée de migration.

Parmi les différentes approches de *migration à chaud*, on retrouve deux principaux algorithmes, l'algorithme de *pré-copie* et l'algorithme de *post-copie*. Les termes *pré-* et *post*-copie font référence à la phase de mise en pause de la VM. La pré-copie consiste à transmettre les données de la VM sur l'hôte destination avant son réveil alors que la post-copie effectue la transmission des données seulement après le réveil sur l'hôte destination.

2.1.1 Algorithme de pré-copie

L'algorithme de pré-copie [Cla+05], représenté en Figure 2.1, est l'implémentation la plus répandue de la migration à chaud. Il consiste principalement à transmettre l'ensemble des pages mémoire de la VM vers l'hôte de destination tout en garantissant une faible durée d'interruption.

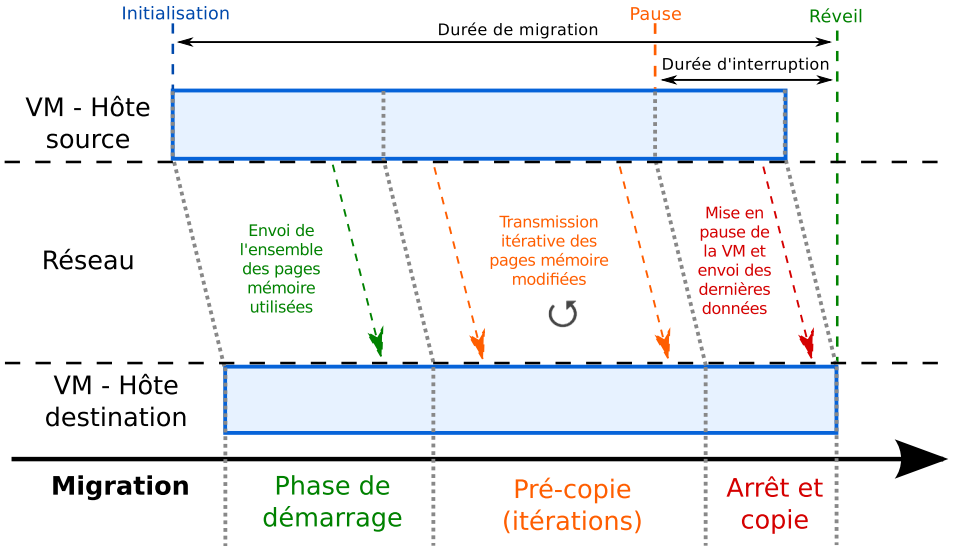


FIGURE 2.1 – Algorithme de pré-copie.

L'algorithme de pré-copie est un processus de transmission itératif. La première étape consiste à envoyer l'ensemble des pages mémoire utilisées vers le serveur de destination. Ce dernier réceptionne les pages et les stocke en mémoire en reconstituant ainsi l'état mémoire de la VM tel que capturé sur le serveur source. La VM étant toujours en cours d'exécution, certaines de ses pages mémoire sont modifiées par les applications durant le transfert et doivent donc être renvoyées. C'est ainsi que le processus itératif de l'algorithme de pré-copie se dévoile, en effet les pages mémoire modifiées durant

chaque transfert sont envoyés lors de la prochaine itération. Ainsi, la quantité de pages mémoire à transférer diminue au fur et à mesure des itérations jusqu'à atteindre la dernière étape d'*arrêt et copie*. L'étape d'arrêt et copie consiste à mettre en pause la VM sur le serveur source, transférer les dernières pages modifiées ainsi que les données annexes, et enfin réveiller la VM sur le serveur destination. Cette étape intervient sous différentes conditions, en fonction des contraintes d'interruption définies par l'administrateur ou encore de l'implémentation faite dans l'hyperviseur utilisé. Dans la majorité des cas, l'administrateur fixe un temps d'interruption maximal autorisé pour la migration d'une VM. Ainsi, au fur et a mesure des itérations, l'hyperviseur estime la durée nécessaire pour envoyer les nouvelles pages modifiées. La migration passe alors en phase d'arrêt et copie si le temps nécessaire estimé pour l'envoi des pages restantes et des données annexes est inférieur à la durée d'interruption maximale exigée.

Cette approche possède des avantages et des inconvénients. L'avantage majeur réside dans l'assurance d'un temps d'interruption contrôlé et borné pour la VM, ce qui permet par exemple de garantir un taux de disponibilité fiable entre un fournisseur de service et ses clients. Le problème majeur intervient lorsque la durée d'interruption maximale est trop faible pour pouvoir envoyer les dernières pages modifiées au serveur destination. Dans ce cas la migration peut durer indéfiniment.

2.1.2 Algorithme de post-copie

L'algorithme de post-copie [HDG10], à l'inverse de la pré-copie, débute par l'arrêt immédiat de la machine virtuelle sur le serveur source puis transfère uniquement les données annexes avant de réveiller la VM sur le serveur de destination. Il est représenté en Figure 2.2.

La durée d'interruption de la VM est très faible en comparaison de l'algorithme de pré-copie puisqu'elle ne dépend que de la seule transmission des données annexes vers le serveur destination. Ainsi, la fréquence de modification des pages mémoire n'impacte plus la durée d'interruption de la VM à migrer. Cependant, la VM est réveillée sur l'hôte cible sans aucune donnée mémoire. Elle doit alors récupérer les pages mémoire manquantes au travers du réseau selon différents procédés entrainant une perte de performance plus ou moins importante. En effet, la perte de performance occasionnée peut engendrer des interruptions de connections réseaux et ainsi nuire fortement aux services s'exécutant sur la VM. Plusieurs techniques sont utilisées pour récupérer les pages mémoire manquantes, ces techniques peuvent être combinées et utilisées conjointement pour limiter la perte de performance occasionnée à la VM. Ces techniques sont les suivantes :

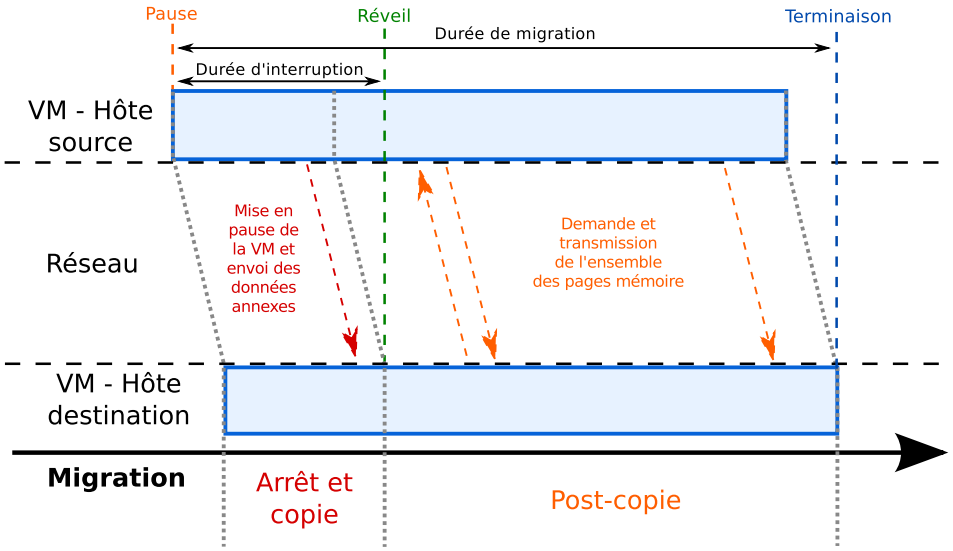


FIGURE 2.2 – Algorithme de post-copie.

La récupération à la demande (*Demand-paging*) [HDG10] : Le serveur de destination initie la demande de récupération d'une ou plusieurs pages mémoire manquantes au serveur source lorsque la VM en a besoin. Cette méthode est contraignante car elle induit un délai d'attente entre le moment ou la VM a besoin d'une page mémoire et le moment où elle est récupéré sur l'hôte de destination au travers du réseau. Ce comportement provoque un ralentissement du fonctionnement de la VM.

La poussée active (*Active-push*) [HDG10] : Cette méthode permet de transférer en continu les pages mémoire restantes sur le serveur de destination à l'initiative du serveur source. Plus important, la *poussée active* permet d'envoyer en priorité les pages mémoire ayant une dépendance résiduelle avec les pages déjà demandées. Dès lors les risques de ralentissement liées à la *récupération à la demande* sont réduits. La durée totale de migration est ainsi minimisée et le temps de ralentissement de la VM réduit en faisant parvenir l'ensemble des pages mémoire manquantes le plus rapidement possible au serveur de destination.

La pré-pagination (*Pre-paging*) [HDG10 ; HG09] : Cette dernière méthode permet d'améliorer la *poussée active* en prédisant les prochaines pages mémoire qui ont le plus de chances d'être accédées rapidement par la VM. En effet, en se basant sur les répétitions et schémas d'accès aux pages mémoire, cette méthode réduit le risque d'erreurs réseau mais aussi la durée de la dernière phase de la *post-copie*.

L'avantage de la post-copie est qu'elle permet de garantir une durée d'interruption fixe très faible quelle que soit l'activité mémoire de la VM à migrer. Elle est ainsi avantageuse pour la migration des VMs ayant un fort taux d'écriture mémoire, en garantissant une durée d'interruption minimale, contrairement à la pré-copie, moyennant une réduction des performances de la VM. L'inconvénient majeur par rapport à la pré-copie repose sur la robustesse de l'approche. En effet, étant donné que la VM est directement réveillée sur l'hôte de destination dans un état inconsistant, la défaillance de l'un des deux hôtes, source ou destination, en cours de migration entraîne la perte inévitable de l'intégrité de l'état mémoire de la VM. À l'inverse, en pré-copie, la défaillance de l'hôte de destination en cours de migration n'a aucun impact sur la VM qui continue alors son exécution normale sur l'hôte source sans aucune perte de données.

2.1.3 Vers de la post-copie hybride

Afin de réduire les problèmes de performances de la *post-copie*, une variante nommée *post-copie hybride* a été définie [Luo+08]. Cet algorithme est considéré comme une post-copie hybride car il reprend les principales caractéristiques de la post-copie en ajoutant une étape de la pré-copie comme illustré en Figure 2.3.

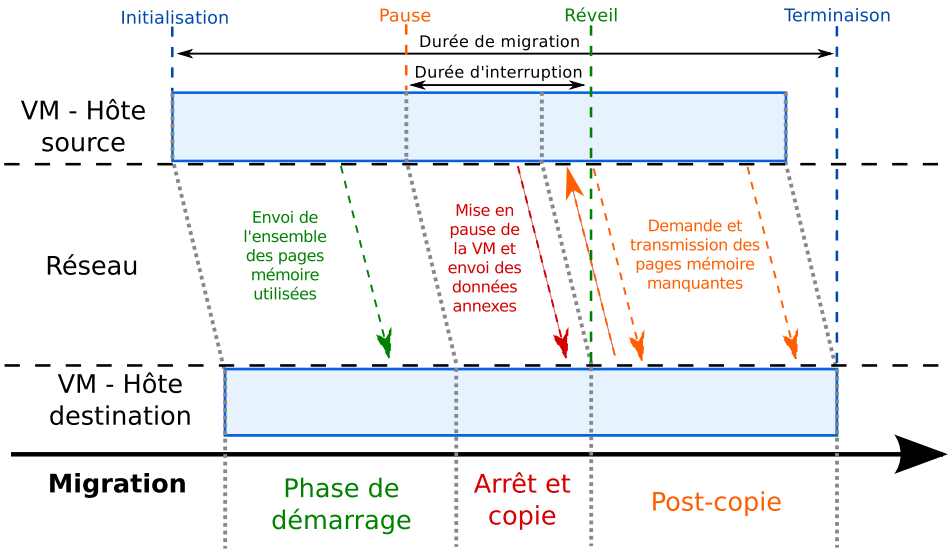


FIGURE 2.3 – Algorithme de post-copie hybride.

En effet l'algorithme effectue la première itération de *pré-copie*, qui revient à transférer l'ensemble des pages mémoire utilisées par la VM, avant de réveiller la VM sur l'hôte de destination. Ainsi, seules les pages mémoire

modifiées depuis la mise en pause de la VM doivent être transférés en suivant les principes de la post-copie, ce qui restreint la perte de performance occasionnée à la VM. La durée d'interruption reste donc minimale et identique à la *post-copie* classique, ce qui en fait une bonne alternative pour les VMs affichant une forte activité mémoire. Cependant, la latence entre le début de la migration et le moment ou la VM devient disponible sur l'hôte destination est considérablement prolongé.

2.2 Implémentation dans les hyperviseurs

Les approches présentées sont à la fois similaires par les étapes qu'elles contiennent et différentes par l'ordre dans lequel elles sont réalisées. Leurs performances et caractéristiques sont néanmoins très différentes. L'algorithme de pré-copie a été le premier protocole proposé pour la migration à chaud de machines virtuelles. Il est rapidement devenu un standard et l'on retrouve aujourd'hui une implémentation du protocole dans tout les hyperviseurs existants, tels que Xen [Cla+05], KVM (Qemu) [Kiv+07], VMware [NLH+05] ou encore Hyper-V [Swa10]. A l'inverse, l'algorithme de post-copie est très peu répandu dans le monde de la virtualisation. Les seules implémentations existantes ont été réalisées à titre expérimental bien que les résultats obtenus soient prometteurs. C'est pourquoi dans la suite de ce document, l'appellation de *migration à chaud* seule fait allusion à l'algorithme de migration par pré-copie.

2.2.1 Détails d'implémentation

Pour déterminer les pages mémoire à transmettre lors de chaque itération, l'hyperviseur maintient une matrice de bits représentant les états de chacune des pages mémoire de la VM à migrer. Un bit à 0 représente une page mémoire identique à son état précédent, et un bit à 1 indique une modification de la page depuis la dernière synchronisation. À intervalles réguliers, l'hyperviseur sonde les pages mémoire des machines virtuelles et maintient la matrice à jour. Les pages représentés par bit à 1 doivent donc être transmises au serveur de destination. La taille des pages mémoire allouées aux machines virtuelles est généralement de 4 Ko. Bien que la taille des pages puisse varier de 4 à 16 Ko voir 4 Mo si l'*extension de la taille des pages* (PSE) est activée, les exemples de migration de machines virtuelles ayant une allocation mémoire de pages supérieurs à 4 Ko sont rares. En effet, l'utilisation de pages mémoire de faible tailles permet d'augmenter la granularité de la transmission des pages sur le réseau et assure un meilleur contrôle de la durée d'interruption. De

plus, lors de l'envoi itératif des pages mémoire, l'hyperviseur adapte la vitesse de synchronisation de la matrice de bit et la transmission des pages pour converger vers un cycle optimal permettant d'utiliser le réseau à sa capacité maximale. On observe ainsi une augmentation progressive de la vitesse de transmission en début de migration jusqu'à atteindre la vitesse de transfert maximale.

Toutes les VMs ont un ensemble de pages mémoire, appelées *pages chaudes*, qui sont amenées à être modifiées très fréquemment. Ces *pages chaudes* sont principalement utilisées pour la pile et les variables locales des processus en cours d'exécution ainsi que les pages allouées pour les tampons réseaux et disques. La quantité de pages à envoyer durant chaque tour de l'algorithme de pré-copie converge vers la quantité de *pages chaudes*. Cependant, ces pages peuvent être transmises plusieurs fois lors de différentes itérations du processus de migration ce qui en augmente la durée. Ainsi, la durée d'interruption est fortement dépendante de la quantité de pages chaudes de la VM.

Quelques différences apparaissent cependant entre les implémentations faites dans les hyperviseurs. Ces différences ont un impact à la fois sur la durée de migration, la quantité de données transmise sur le réseau et la durée d'interruption des machines virtuelles. Les implémentations changeant très rapidement dans les hyperviseurs, certaines optimisations ont pu être introduites après l'écriture de ce document. À titre d'exemple, le code concernant la migration à chaud de l'hyperviseur KVM à été intégralement ré-écrit durant la période de cette thèse.

Dans la suite de cette section, nous analysons essentiellement les différences des deux hyperviseurs libres les plus utilisés que sont *KVM (Qemu)* [Kiv+07] et *Xen* [Bar+03]. La différence majeure se situe dans la gestion de la durée d'interruption occasionnée à la VM.

2.2.1.1 Durée d'interruption bornée

Avec une durée d'interruption maximale prédéfinie, tel qu'implémenté dans l'hyperviseur KVM, la migration passe en phase d'*arrêt et copie* lorsque les pages mémoire modifiées lors de l'itération précédente peuvent être envoyées en temps inférieur à la durée d'interruption maximale fixée. En pratique, la quantité de pages modifiées diminue au fur et à mesure des itérations et converge vers la quantité de pages chaudes dont la fréquence d'écriture est souvent supérieure à la vitesse de transmission sur le réseau. La retransmission des pages s'arrête alors lorsque la quantité de pages modifiées est devenue suffisamment faible ce qui fait que la durée d'interruption occasionnée est donc toujours légèrement inférieure mais tout de même très proche de la borne donnée.

Si la durée d'interruption maximale fixée est trop faible pour permettre l'envoi de l'ensemble des pages chaudes, alors l'algorithme de pré-copie peut itérer indéfiniment sans jamais atteindre la phase d'*arrêt et copie*. De plus, la quantité de pages chaudes peut varier plus ou moins fortement lors des phases successives de synchronisation de la matrice de bits et ainsi rendre la terminaison de la migration imprédictible. Ainsi, en considérant une durée d'interruption de service maximale prédéterminée, la terminaison de la migration d'une VM à forte empreinte mémoire n'est pas toujours assurée et peut s'avérer être totalement imprévisible.

2.2.1.2 Durée d'interruption minimale

D'autres implémentations tel que dans l'hyperviseur Xen ne garantissent pas de durée d'interruption minimale par défaut, mais s'assurent de minimiser cette dernière. Pour pouvoir assurer une durée d'interruption minimale quel que soit le type de service s'exécutant au sein de la VM, Xen identifie les pages chaudes en cours de migration. En pratique, les pages mémoire étant modifiées durant deux itérations consécutives sont identifiées comme des pages chaudes. Pour éviter les envois répétitifs de ces pages affichant une fréquence d'écriture élevé, leur transmission est alors reportée jusqu'à la phase d'*arrêt et copie* ce qui permet de conserver des durées de migration et d'interruption minimales. En effet, en gardant l'envoi des pages chaudes pour la dernière étape de la pré-copie, Xen évite de les renvoyer plusieurs fois durant la migration et réduit ainsi le temps de migration.

La technique permettant de différer l'envoi des pages chaudes est inspirée des travaux de Svärd et al. [Sv11b]. Ces derniers proposent une modification de l'hyperviseur KVM permettant de réordonner le transfert de l'ensemble des pages mémoire modifiées en priorisant les pages affichant un faible taux d'écriture. Cette technique est plus avancée que la simple détection des pages mémoire modifiées en deux itérations consécutives, mais elle induit une consommation de ressources plus élevée et n'a d'ailleurs pas été implémenté officiellement dans l'hyperviseur KVM. De plus, la durée d'interruption est automatiquement adaptée à l'intensité d'écriture des pages mémoire de chaque machine virtuelle. Ainsi la quantité de pages chaudes n'impacte plus la faisabilité des migrations mais leur durée d'interruption.

Opter pour une durée d'interruption dynamique entraîne aussi quelques désavantages. En effet, dans le cas d'un fournisseur de services *cloud*, ce dernier n'est potentiellement plus en mesure de garantir son contrat de qualité de service (SLA) et peut être contraint de payer des pénalités.

2.2.2 Amélioration du processus de migration

Parmi les techniques mises au point pour améliorer et parfaire le processus de migration à chaud, on retrouve globalement l'idée de restreindre la quantité de données à transmettre sur le réseau afin de réduire la durée de migration et la durée d'interruption de la VM.

L'auto-ballonnement dynamique (*dynamic self-ballooning*) [Wal02] est une première fonctionnalité importante initialement introduite dans VMware en 2002 et aujourd'hui implémentée à la fois dans Xen et KVM. Elle consiste à allouer dynamiquement de la mémoire à une machine virtuelle à la demande en fonction de ses besoins. L'allocation dynamique de mémoire permet d'abord d'augmenter la consolidation dans les centres des données virtualisés grâce au surengagement des ressources mémoire (*memory overcommitment*). Il est ainsi possible d'héberger une quantité plus importante de VMs en ajustant l'allocation mémoire de manière à pouvoir absorber les pics de consommation temporaire de l'ensemble de VMs sans saturer les ressources matérielles des hôtes. Cette fonctionnalité n'est cependant pas sans risques car elle suppose que toutes les VMs hébergées sur un hôte physique n'atteindront jamais leur consommations maximales en même temps. Au niveau de la migration à chaud, l'auto-ballonnement dynamique permet de limiter la transmission des pages à la quantité de mémoire réellement utilisée par la machine virtuelle, et donc de réduire sa durée de migration en évitant l'envoi de pages vides.

La *compression des pages mémoire* [Jin+09] est une première approche introduite expérimentalement dans l'hyperviseur Xen dès 2009. Elle consiste à compresser les pages mémoire sur l'hôte source avant leur transfert, puis les décompresser une fois réceptionnées sur l'hôte destination. L'inconvénient de cette approche étant le temps et les ressources nécessaires à la (dé)compression des pages mémoire. Afin d'améliorer cette technique, les auteurs ont ensuite décidé d'utiliser différents algorithmes de compression de manière dynamique en fonction du taux de similarité des pages pour obtenir un meilleur ratio *compression/performance*.

La *compression du delta* [Sv11a ; Sv11c ; Woo+15] est une méthode consistant à ne transmettre que les différences entre les versions des pages mémoire sous forme compressée dans le but de réduite la quantité de données transférées. Une page mémoire contient des données sous forme binaire. Un *XOR* est alors appliqué entre la version courante d'une page et celle de l'itération précédente pour obtenir son *delta*. Le delta a la même taille que

la page mémoire d'origine mais peut être compressée bien plus efficacement car de manière générale la proportion des modifications faites sur les pages mémoire, entre deux itérations, reste faible.

La *dé-duplication de données* [Zha+10 ; TSH12] est la seconde idée majeure permettant l'amélioration du processus de migration à chaud. Certaines pages mémoire pouvant être identiques, la dé-duplication de données consiste à identifier les données redondantes pour ne les transmettre qu'une seule fois. Des tables de hachage indexant les parties communes (par portions de 64 octets) des pages mémoire sont alors créées et maintenues sur les deux hôtes source et destination. Comme pour la compression de données, la dé-duplication utilise des algorithmes d'encodage pour éliminer les données redondantes. Les algorithmes les plus performants consommant beaucoup de ressources, des compromis doivent être fait entre l'efficacité de l'encodage et le temps de calcul requis pour pouvoir réellement réduire les durées de migration. D'autres techniques de dé-duplication liées aux migrations concurrentes de plusieurs VMs sont abordées en section 3.

2.3 Virtualisation et migration du stockage

Historiquement, la migration à chaud ne déplaçait que la mémoire et l'état des périphériques de la VM (registres CPU et des interfaces réseaux), ce qui restreignait la migration entre serveurs partageant un même stockage commun tel que NFS ou iSCSI.

De nos jours, la migration à chaud du stockage permet de déplacer l'ensemble des données d'une machine virtuelle en incluant son image disque, ce qui permet de passer outre la nécessité du stockage partagé. En effet, alors que la taille mémoire des VMs se situe généralement entre 2 et 8 Go, l'espace disque dédiée à une VM est plus important et atteint jusqu'à 30 Go en moyenne [TSH12]. Dès lors, le temps passé à renvoyer les données modifiées sur le disque virtuel (de manière itérative) devient rapidement conséquent. On assiste donc à une forte augmentation de la durée de migration à chaud, par exemple avec un réseau local gigabit (1 Gbps), 10 minutes sont nécessaires pour pouvoir migrer un espace disque de 20 Go. C'est pourquoi il est généralement recommandé de mettre en place un réseau secondaire dédié à la migration, ou encore de réserver et garantir une bande passante fixe pour la migration.

2.3.1 Virtualisation du stockage

Les systèmes de stockage peuvent fournir deux types d'accès au stockage, soit par bloc soit par fichier. Les protocoles fournissant un accès par bloc sont nombreux, on retrouve principalement FC, iSCSI, SAS ou encore FICON [Tro+11]. Les protocoles les plus utilisés fournissant un accès par fichier sont généralement NFS et SMB [Say02]. Deux types de virtualisation du stockage ont ainsi été définis, la virtualisation d'accès par bloc et la virtualisation d'accès par fichier. La virtualisation de l'accès par bloc fait référence à la séparation entre stockage logique et physique pour permettre un accès aux données indépendamment des protocoles utilisés. La virtualisation de l'accès par fichier quant à lui élimine les dépendances entre l'accès des données au niveau des fichiers et l'endroit où les fichiers sont physiquement stockés. Cela permet par exemple de faire apparaître plusieurs systèmes de fichiers distincts comme un simple système de fichier ayant des espaces de noms multiples. Cependant, la migration à chaud du stockage profite essentiellement de la virtualisation d'accès par bloc, c'est pourquoi nous ne nous réfèrerons qu'à ce type précis de virtualisation de stockage dans le reste de ce document.

La virtualisation du stockage réfère au processus d'abstraction du stockage physique dans des conteneurs virtualisés appelés disques virtuels pouvant être utilisés par les applications. Cette couche d'indirection entre les applications et le stockage physique permet d'agréger des systèmes de stockage physique, même hétérogènes, dans des ensembles (*pools*) logiques. Les disques virtuels, abstraits depuis ces ensembles, évoluent de manière dynamique et transparente pour les applications qui les utilisent. Par exemple l'espace d'un disque virtuel peut être redimensionné à la volée lors de l'ajout d'un nouveau stockage physique à l'ensemble logique dont il est issu.

L'avantage premier réside dans la possibilité de consolidation du stockage sans se soucier des différents protocoles utilisés, permettant aux applications de facilement partager des ressources de stockage hétérogènes. Mais la virtualisation du stockage fournit bien d'autres avantages tels que le redimensionnement dynamique, la réplication, la compression ou encore la migration du stockage.

2.3.2 Migration de stockage virtualisé

L'intégration de la virtualisation de stockage avec la virtualisation serveur a permis d'améliorer la consolidation et la répartition de charge en optimisant l'utilisation des ressources à l'intérieur des centres de données. De manière plus générale, la gestion commune de ces deux types de virtualisation a fortement augmenté l'efficacité des centres de données virtualisés via l'utilisation

de la meilleure combinaison de ressources serveur et stockage pour chaque application.

Ainsi, parmi les avantages de la virtualisation du stockage on retrouve la possibilité de migration à chaud des disques virtuels [Luo+08]. La migration de disques virtuels est réalisée de la même manière que la mémoire suivant les algorithmes de pré-copie ou de post-copie. Ainsi, les applications entraînant des écritures disques fréquentes, tel que par exemple les transactions dans les bases de données [LD93], ralentissent considérablement la migration et augmentent la durée d'interruption de la VM.

Des recherches ont été entreprises dans la réduction du trafic nécessaire à la migration de stockage [TSH12]. Tout comme pour la migration de la mémoire, des techniques de dé-duplication sont utilisées et permettent de réduire la quantité de données à transférer entre les hôtes physiques. La taille de l'image disque étant généralement supérieure à la mémoire allouée aux VMs, les opportunités de duplication et de compression des parties du disque sont bien plus nombreuses et le gain plus élevé que pour la mémoire.

2.4 Modèles de performance

La performance de la migration à chaud est principalement caractérisée par la durée de l'opération, la durée d'interruption de la VM, mais aussi de la quantité de données transmises sur le réseau pendant la migration. Plus récemment, la consommation énergétique de la migration est devenue un nouveau critère de performance.

La performance de la migration à chaud est affectée par plusieurs facteurs. Premièrement la quantité de mémoire utilisée par la VM impacte fortement la durée totale de migration et donc la quantité de trafic réseau généré. Deuxièmement le taux d'écriture mémoire, qui reflète le type d'activité mémoire des différentes applications, impacte le nombre d'itérations de l'algorithme de pré-copie et affecte indirectement la durée de migration et le temps d'interruption. Troisièmement, la vitesse de transfert réseau est cruciale pour la performance de la migration.

La performance de la migration à chaud a été largement étudiée ces 10 dernières années. De nombreux modèles de prédiction permettant d'estimer les performances de la migration à chaud ont été définis. Parmi les différentes approches existantes, la distinction majeure se situe au niveau de la caractérisation de l'activité mémoire de la machine virtuelle et ainsi de la modélisation de la quantité de pages mémoire modifiées durant le processus de migration. En effet, le taux d'écriture des pages mémoire varie plus ou moins fortement en fonction des applications en cours d'exécution, il est donc difficile à modéliser.

2.4.1 Modèles linéaires

Selon les principes de la migration à chaud, la durée de migration d_{mig} évolue linéairement par rapport à la quantité de mémoire à transmettre m_{mig} et la bande passante disponible pour la migration bp. La durée d'interruption occasionnée à la VM i_{mig}, généralement très faible, n'a que très peu d'impact sur la durée de migration.

La formule 2.1 représente ainsi la durée de migration estimée.

$$d_{mig} = \frac{m_{mig}}{bp} + i_{mig} \tag{2.1}$$

Certains outils de simulation tel que CloudSim [Cal+11] par exemple considèrent que m_{mig} correspond simplement à la quantité de mémoire utilisée par la VM en début de migration. Cependant, comme illustré en section 2.1, la quantité de mémoire m_{mig} transmise lors de la migration est difficilement prédictible car elle varie en fonction de la vitesse d'écriture des pages mémoire de la VM et de la bande passante.

Certains travaux [Ako+10 ; STP11 ; ST13a ; Ver+11] considèrent alors un taux d'écriture des pages mémoire constant. La quantité de mémoire à transmettre m_{mig} dépend ainsi de la quantité de mémoire utilisée par la VM en début de migration ainsi que de son taux d'écriture mémoire.

En pratique, le taux d'écriture des pages mémoire varie plus ou moins fortement en fonction du type d'application s'exécutant sur la VM. En effet, tel que décrit en section 2.2 la quantité de pages chaudes mais aussi les particularités de l'implémentation de l'algorithme de pré-copie ont une importance majeure dans l'estimation de la durée de migration. Par exemple, Akoush et al. [Ako+10] définissent un modèle de performance itératif basé sur l'algorithme de pré-copie tel qu'implémenté dans l'hyperviseur Xen. La fonctionnalité permettant d'éviter les envois répétitifs des pages chaudes est considérée mais leur modèle de prédiction analytique ne repose que sur un simple taux d'écriture statique des pages mémoire. Une spécialisation du modèle a donc ensuite été définie en cas de trop forte variabilité de l'activité mémoire de la VM. L'idée étant d'enregistrer l'activité mémoire des VMs en considérant qu'elle se répète dans le temps de manière déterministe. Il est alors possible de prédire le taux d'écriture mémoire des VMs en fonction des informations précédemment collectées mais cette technique reste inadéquate pour des VMs affichant des activités mémoire non prédictibles.

2.4.2 Fréquence d'écriture des pages mémoire

Afin d'améliorer les performances des approches linéaires classiques, des études plus récentes tentent de modéliser la quantité de pages chaudes des VMs qui joue un rôle important dans la durée de migration mais aussi la durée d'interruption des VMs.

Dans une première étude, Liu et al. [Liu+11] statuent que la quantité de pages chaudes est approximativement proportionnelle au nombre total de pages modifiées lors de chaque itération de pré-copie. En s'appuyant sur les statistiques d'écriture des pages mémoire issues de plusieurs types d'applications, les auteurs utilisent une méthode d'apprentissage par régression linéaire pour généraliser une formule permettant de déduire la quantité de pages chaudes d'une VM en fonction du taux moyen de réécriture de ses pages mémoire. Ainsi pour estimer la durée de migration d'une VM, le taux d'écriture moyen de ses pages mémoire est d'abord calculé à partir de l'analyse de son activité au sein de l'hyperviseur Xen. La durée de migration est ensuite estimée à l'aide d'un algorithme itératif simulant l'implémentation de la pré-copie dans l'hyperviseur Xen. De plus, lorsqu'une VM est amenée à migrer plusieurs fois, les auteurs utilisent une *moyenne mobile pondérée exponentiellement* (EWMA) en combinant les anciens taux mémoires calculés pour la VM afin d'améliorer la prédiction de la durée de migration. Les expérimentations effectuées montrent une très bonne précision de l'estimation des durées de migration sur différents types d'applications. L'inconvénient de cette approche réside principalement dans l'algorithme itératif utilisé. En effet, les risques d'erreur liés aux approximations ainsi que la complexité du calcul augmentent avec le traitement de chacune des itérations de l'algorithme de pré-copie. De plus la spécialisation de l'algorithme suivant l'implémentation faite dans Xen limite l'intérêt du modèle à ce seul hyperviseur.

Une seconde étude de Zheng et al. [Zhe+13], reposant cette fois sur l'hyperviseur KVM, propose un modèle analytique permettant de prédire le temps de fin d'une migration. Leur solution consiste en une approche en temps réel qui permet d'estimer la proportion de pages chaudes en calculant un *ratio d'accès* pour chacune des pages mémoire de la VM. Le ratio d'accès d'une page repose sur sa fréquence de modification et représente la contribution de la page par rapport à l'ensemble des pages mémoire modifiées lors de la capture de l'état mémoire de la VM. La solution proposée repose sur une modification de l'hyperviseur KVM permettant d'analyser l'activité mémoire des VMs et d'agir directement sur le processus de migration. À l'aide d'un algorithme d'échantillonnage, leur solution consiste à calculer puis maintenir périodiquement à jour le ratio d'accès de chacune des pages mémoire afin d'améliorer la précision de l'estimation au fur et à mesure de la migration. Bien que cette

solution soit efficace, elle est toutefois spécifique à KVM et nécessite une modification profonde de l'hyperviseur. De plus, un ajustement en temps réel du ratio d'accès des pages est nécessaire pour obtenir une estimation précise.

Une dernière étude réalisée par Breitgand et al. [BKR11] se sert d'un modèle probabiliste pour estimer la fréquence d'écriture de chaque page mémoire et quantifier la proportion de pages chaudes d'une VM. La solution proposée consiste d'abord à récupérer une trace de la VM à migrer en réalisant une capture de toutes ses opérations mémoire sur une période de deux minutes. En s'appuyant sur ces traces, les auteurs définissent alors un modèle permettant de calculer la probabilité d'écriture de chaque page mémoire et ce à chaque itération de l'algorithme de pré-copie pour pouvoir estimer la durée de migration à chaud. Les expérimentations ont été réalisées à l'aide de l'hyperviseur Xen qui fournit l'ensemble des outils nécessaires à la capture et l'analyse de l'activité mémoire des VMs, contrairement à KVM. On observe de bonnes performances du modèle probabiliste sur des VMs de taille mémoire différentes. Cependant, la nécessité de capturer les opérations mémoire des VMs sur une période de deux minutes restreint fortement l'intérêt du modèle proposé.

Bien que ces différentes approches permettent d'obtenir des estimations précises des durées de migration, elles requièrent toutefois une analyse approfondie de l'activité mémoire des VMs à migrer. Ainsi, le temps et les ressources nécessaires à l'analyse de chaque page mémoire et de leur fréquence d'écriture sont un frein à la prédiction des durées de migration. Plus généralement, la volonté de reproduction des algorithmes de migration tel qu'implémentés dans les hyperviseurs complexifie les modèles et les rendent dépendants des particularités et spécificités propres à chaque implémentation tel que décrites en section 2.2.

2.5 Discussion

Les différences d'implémentation et l'évolution constante des méthodes d'optimisation du processus de migration tendent à rendre les modèles de prédiction de performance trop spécifiques. Par exemple, la propriété d'incrémentation progressive de la vitesse de transfert en cours de migration telle que décrite en section 2.2 peut impacter légèrement la durée de migration. Cependant cette fonctionnalité n'a été réellement prise en considération que dans deux études [SS09a ; ST13b] car les auteurs ont principalement concentrés leurs recherches sur l'utilisation des ressources réseaux pour l'étude de migrations concurrentes de machines virtuelles. De plus, d'autres variantes d'implémentation telles que la fréquence de synchronisation du vecteur de bits, représentant l'état des pages mémoire, ou encore la fenêtre variable de

transmission des pages lors de chaque itération sont un frein pour la généralisation des modèles de performance. Il est ainsi difficile de comparer les différentes approches itératives entre elles.

La complexité des modèles de performance est également problématique. En effet, une analyse approfondie des pages mémoire ralenti le processus d'estimation de durée de migration et restreint fortement les possibilités de passage à l'échelle des modèles de prédiction. Par exemple, lors d'une opération de maintenance ou de décommissionnement qui nécessite de migrer un grand nombre de VMs le plus rapidement possible.

Il faudrait ainsi être capable de s'abstraire du modèle itératif de transmission des pages en définissant un modèle analytique simple et générique permettant d'établir des prédictions rapides et précises des durées de migration à chaud. La capture et l'analyse de l'activité mémoire des VMs restant nécessaires, elle doivent être rendues courtes et minimales pour permettre une extraction rapide des paramètres clefs nécessaires à la prédiction de la performance des migrations.

Chapitre 3

Ordonnancement des migrations

Sommaire

3.1 Principes d'ordonnancement 30
3.1.1 Objectifs et contraintes 30
3.1.2 Critères de performance 31
3.2 Gestion du réseau 32
3.2.1 Architecture réseau 32
3.2.2 Réseau de migration 35
3.3 État de l'art 36
3.3.1 Techniques d'ordonnancement 36
3.3.2 Optimisation des migrations 39
3.3.3 Contrôle de l'ordonnancement 41
3.4 Discussion 43

Depuis l'avènement de la virtualisation, la migration à chaud de machines virtuelles est continuellement requise dans les centres de données virtualisés. En effet, les algorithmes de placement dynamiques sont largement employés pour calculer de nouveaux placements de VMs toujours plus efficaces. Les VMs sont ainsi migrées pour maximiser leurs performances, répartir la charge entre les hôtes physiques, réaliser des tâches de maintenance ou encore réduire la consommation énergétique des centres de données. Pour appliquer les nouveaux placements calculés pour un ensemble de VMs, de nombreuses migrations à chaud doivent être réalisées. Il s'agit alors, premièrement, de terminer l'ensemble des migrations le plus rapidement possible pour pouvoir bénéficier des avantages des nouveaux placements au plus tôt ; deuxièmement, de réaliser les migrations de manière efficace, en évitant les congestions réseaux par exemple et en minimisant les pertes de performances occasionnées aux VMs qui peuvent limiter l'intérêt même des nouveaux placements.

Dans ce chapitre, nous présentons les points clefs de l'ordonnancement des migrations à chaud de VMs et discutons des techniques utilisées dans l'état de l'art pour optimiser et coordonner ces migrations. Dans un premier temps, nous présentons les principes et objectifs de l'ordonnancement des migrations de VMs, ainsi que les critères de performance permettant de déterminer son

efficacité. Dans une seconde partie, nous discutons de la gestion du réseau et plus généralement de l'importance de l'architecture réseau dans le calcul d'un ordonnancement de migrations. Finalement, nous dressons un état de l'art complet et détaillé des différentes techniques d'ordonnancement, d'optimisation et de contrôle de migrations multiples.

3.1 Principes d'ordonnancement

La nécessité d'ordonnancer des migrations résulte directement de la décision de déplacement d'un ensemble de VMs. Dans nos travaux, nous nous plaçons dans le contexte de l'ordonnancement de migrations pur. C'est à dire que les décisions d'ordonnancement sont considérées déconnectées des décisions de placement qui ont été prises en amont. Nous considérons alors que l'efficacité de l'ordonnancement calculé est indépendante de celle du nouveau placement des VMs. Ainsi, la qualité de l'ordonnancement ne peut pas être pénalisée par des décisions de placement n'ayant pas tenu compte des contraintes liées à l'ordonnancement des migrations sous-jacentes.

La relation entre décisions de placement et d'ordonnancement sera abordée dans le chapitre 7 mais reste en dehors du contexte de cette thèse qui traite essentiellement de l'ordonnancement des migrations.

3.1.1 Objectifs et contraintes

L'objectif principal de l'ordonnancement consiste à réaliser l'ensemble des migrations de la manière la plus efficace possible. La notion d'efficacité de l'ordonnancement diffère selon la nature de la tâche à réaliser mais aussi de l'urgence qu'elle suscite. En effet, de nombreux scénarios peuvent amener à vouloir déplacer un ensemble de VMs. Parmi ces scénarios on distingue principalement deux types d'opérations de natures différentes.

Premièrement, les tâches récurrentes, c'est-à-dire des opérations régulières ou périodiques réalisées à des fins d'équilibrage de charge, d'économie d'énergie, ou en vue d'optimiser les performances des applications par exemple. Dans ce cas précis, la réalisation des migrations occasionne un gain immédiat pour le centre de données, que ce soit en terme de performance ou encore de consommation énergétique. Les migrations sont alors réalisées au plus tôt, de manière autonome ou semi-automatique, pour pouvoir profiter des optimisations dès que possible.

Le second type d'opération concerne les tâches rares à caractères exceptionnels. Par exemple lors de la planification d'une tâche de maintenance telle que la mise à jour d'hyperviseurs ou encore le remplacement ou la reconfiguration

d'équipements réseaux. Ces tâches de maintenance nécessitent généralement le décommissionnement de baies entières de machines pour une durée limitée. Dans ce cas, la réalisation des migrations conduit le centre de données dans un état temporaire, réduisant potentiellement ses performances pendant toute la durée de maintenance. De plus, à la différence des tâches récurrentes, ces opérations requièrent l'intervention d'une personne physique et sont planifiées minutieusement à des moments opportuns pour limiter l'impact sur l'infrastructure du centre de données.

Certaines situations exceptionnelles peuvent aussi entraîner la migration massive de VMs. Par exemple à la veille d'une perturbation imminente tel qu'un événement climatique important (tornade, séisme, tsunami, etc.) pouvant entraîner un arrêt total des services (panne d'électricité, détérioration des équipements, etc.). Pour pouvoir gérer de telles situations, des scénarios de *disaster recovery* sont mis en place et peuvent impliquer la migration d'une grande quantité de VMs vers un centre de données distant en une période de temps très courte. Au vue de la criticité de ce type de situations, le calcul de l'ordonnancement des migrations doit être réalisé très rapidement et son efficacité doit être optimale. L'ordonnancement peut alors être sujet à des contraintes particulières, par exemple certaines VMs plus critiques que d'autres peuvent être migrées en priorité et la durée maximale d'interruption de certaines VMs peut être augmenté pour réduire la durée de l'opération.

Bien que ces différentes situations entraînent des contraintes d'ordonnancement particulières qui diffèrent selon la nature et l'urgence de l'opération, les critères de performance de l'ordonnancement restent identiques. En effet, il s'agit toujours de terminer l'ensemble des migrations le plus rapidement possible tout en minimisant les perturbations occasionnées.

3.1.2 Critères de performance

Ordonnancer un ensemble de migrations consiste à déterminer précisément l'ordre dans lequel elles doivent être exécutées. Pour réaliser un ordonnancement optimal, il est d'abord essentiel de garantir une performance optimale pour l'ensemble des migrations. Tel que décrit dans le chapitre précédent, la performance d'une migration à chaud est principalement caractérisée par sa durée et la période d'interruption occasionnée à la VM. Il s'agit alors, premièrement, de garantir des durées de migration et d'interruption minimales pour chacune des migrations. Dans un second temps, en plus d'assurer des performances maximales pour chaque migration, il est impératif de considérer l'opération de reconfiguration dans son ensemble. En effet, la reconfiguration ne se termine que lorsque toutes les migrations ont été réalisées. Ainsi, pour terminer le plus rapidement possible, l'ordre d'exécution et le niveau de paral-

lélisation des migrations doivent être définies méticuleusement pour minimiser la durée totale de l'opération.

Le calcul d'un ordonnancement efficace revient à déterminer l'ordre et le niveau de parallélisation des migrations de manière à terminer le plus rapidement possible tout en conservant une qualité de service acceptable aux utilisateurs des VMs. En pratique, les décisions de parallélisation des migrations doivent essentiellement tenir compte de l'architecture réseau du centre de données.

3.2 Gestion du réseau

La gestion du réseau joue un rôle important dans l'ordonnancement de migrations. En effet, la durée d'une migration à chaud est directement lié à la bande passante disponible pour cette dernière. Ainsi, afin de garantir une bande passante maximale pour chaque migration il est nécessaire de connaître la capacité et l'utilisation des différents équipements réseaux du centre de données. Par exemple, les commutateurs qui composent le réseau peuvent être bloquants, c'est-à-dire avec des capacités de commutation limitées par rapport à la quantité des interfaces réseau disponibles et de leur débit théorique. De plus, pour garantir une parallélisation optimale et éviter tout phénomène de congestion, il est impératif d'étudier la topologie réseau du centre de données.

3.2.1 Architecture réseau

La topologie réseau d'un centre de données est un élément important à considérer pour l'ordonnancement de migrations à chaud. Les architectures réseaux les plus répandues dans les centres de données actuels sont basées sur des modèles hiérarchiques [AFLV08]. Ces architectures, rappelant un arbre hiérarchique, sont basées sur différentes couches de commutateurs tel que représenté en Figure 3.1. La couche plus basse est communément appelée *couche d'accès*, et raccorde directement les serveurs entre eux. La méthode de raccordement la plus répandue pour la couche d'accès est la topologie *Top of Rack* (ToR) qui consiste à installer un commutateur réseau en haut de chaque baie de serveurs pour fournir la connectivité nécessaire à l'ensemble des serveurs de la baie. Cette solution permet entre autre de conserver le plus gros du câblage des serveurs à l'intérieur de la baie et de faciliter le raccordement à la couche supérieure. La couche supérieure est une couche intermédiaire dite *d'agrégation* (ou encore *de distribution*) à laquelle la couche d'accès est connectée de façon redondante. La redondance des connections entre commutateurs est un des points clefs de ce type d'architecture qui permet à la fois d'agréger

des liens de communication et d'assurer une tolérance en cas de panne d'un commutateur. La dernière couche, la plus haute, est la couche de *cœur de réseau* qui consiste à relier les commutateurs d'agrégation entre eux. Ainsi, les commutateurs ToR sont raccordés de manière redondante aux commutateurs d'agrégation, eux-mêmes reliés à un ou plusieurs commutateurs de cœur de réseau, de manière hiérarchique. De cette façon, il est possible de rajouter une nouvelle baie de serveurs avec un minimum de raccordement, en connectant son commutateur ToR sur les commutateurs d'agrégation.

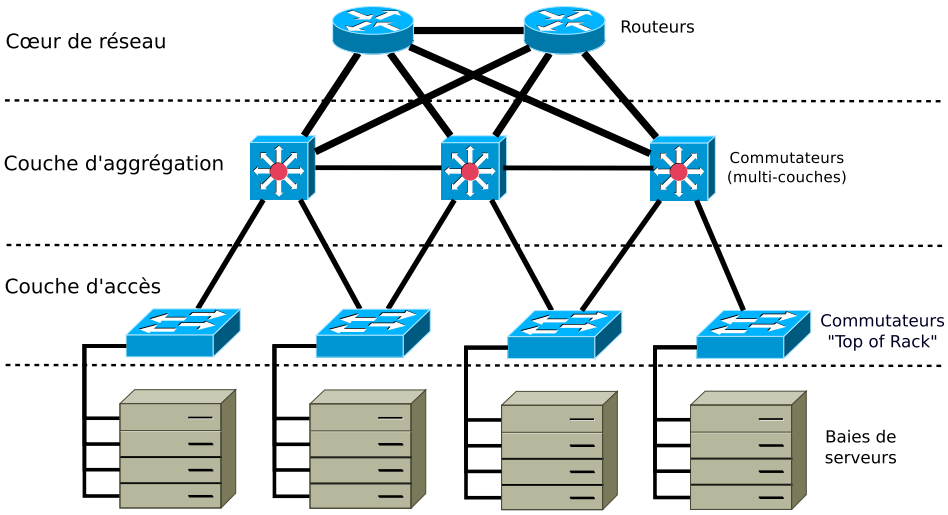


FIGURE 3.1 – Architecture réseau hiérarchique.

Le point faible des topologies hiérarchiques réside dans le cœur de réseau qui peut être sujet à certains goulets d'étranglements. En effet, les multiples couches se traduisent aussi par un impact en terme de latence, certains serveurs devant remonter jusqu'à la couche cœur pour communiquer avec d'autres serveurs. Les commutateurs de couches supérieurs doivent ainsi traiter une quantité de trafic plus importante et les besoins en performance sont conséquents. Ces réseaux en forme d'arbres sont de types *fat tree*, c'est à dire que les branches supérieures de l'arbre sont plus larges que les branches inférieures. Cela signifie que les liens réseaux proches du cœur ont une bande passante plus élevée que les liens des couches inférieures. Les liens les plus faibles étant ceux de la couche d'accès qui relient les machines d'une baie à leur commutateur ToR , on retrouve généralement une bande passante allant de 1 à 10 Gbps dans le cas d'un réseau Ethernet, et jusqu'à 40 Gbps pour un réseau Infiniband.

3.2.1.1 Réseaux bloquants

On distingue deux catégories différentes de commutateurs, bloquants et non-bloquants. La capacité d'un commutateur correspond à la bande passante théorique maximale qu'il est capable de traiter, on parle alors d'une capacité de commutation de fond de panier maximum. Ainsi, un commutateur bloquant dispose d'un fond de panier restreint par rapport au débit global théorique de l'ensemble de ses interfaces. A l'opposé, un commutateur non-bloquant peut théoriquement traiter autant de trafic que lui permet ses interfaces réseaux utilisées à leur capacité maximale, et donc sans être limité par le fond de panier. La limitation principale de l'utilisation des commutateurs non-bloquant réside dans leur coût très élevé, il est donc courant de disposer de réseaux bloquants pour une question de budget, surtout au niveau du cœur du réseau qui nécessite du matériel très performant.

Dans un réseau bloquant, afin d'éviter de saturer le cœur de réseau, il est coutume de limiter le plus possible la remontée de trafic jusqu'aux commutateurs cœurs en privilégiant par exemple les transferts intra-baies ou entre les baies connectées à un même commutateur d'agrégation. Dans le cas de l'ordonnancement de migrations, les capacités limitées d'un réseau bloquant viennent restreindre les possibilités de parallélisation des migrations.

3.2.1.2 Limitations du parallélisme

La capacité de parallélisation des migrations est d'abord limitée par le débit des interfaces réseaux des hôtes. Ces derniers étant généralement homogènes, la parallélisation des migrations entre deux hôtes ayant la même capacité de transfert occasionne inéluctablement une réduction de bande passante pour chaque migration. Cette limitation de bande passante doit être évitée car elle réduit la performance des migrations en augmentant à la fois la durée des migrations et la quantité de données transmises sur le réseau. En effet, en raison de la nature itérative du processus de migration liée à l'activité mémoire des VMs, plus la bande passante allouée à une migration est faible et plus la quantité de pages mémoire transmises sur le réseau est élevée.

La seconde limitation de la parallélisation des migrations est directement liée aux réseaux bloquants. Par exemple, lors du décommissionnement d'une ou plusieurs baies de serveurs, les migrations occasionnées proviennent d'un même endroit de l'infrastructure et ont ainsi de fortes chances de passer par une même portion limitée du réseau. La parallélisation des migrations inter-baies est ainsi limité par la capacité des commutateurs des différentes couches de l'architecture réseau. C'est pourquoi l'architecture réseau est importante et doit être prise en considération lors de l'ordonnancement de migrations afin d'éviter les problèmes de congestions et d'assurer des durées de migration

minimales quel que soit la quantité, la provenance et la destination des VMs.

3.2.2 Réseau de migration

Pour calculer un ordonnancement précis et réaliste, il est nécessaire d'estimer la durée de chaque migration le plus précisément possible. Pour ce faire, il est important de s'assurer que chacune d'entre elles dispose d'une bande passante stable. Il est alors possible de réserver une certaine quantité de bande passante pour la migration, par exemple en régulant le débit (*bandwidth throttling*) des services s'exécutant sur l'hôte de la VM à migrer, y compris le débit provenant de la VM elle-même [Kri98]. En effet, les applications s'exécutant sur les VMs peuvent générer un trafic réseau important qui va venir concurrencer le trafic issu de la migration. La réservation de ressource doit être réalisée avec précaution, il faut pouvoir réserver une quantité suffisante de bande passante pour la migration sans impacter trop fortement la qualité de service des applications [BDP93]. Cette méthode demande ainsi une bonne connaissance des besoins en ressources réseau des applications bien qu'il soit difficile de prédire le futur trafic réseau d'une VM pour toute la durée de sa migration.

Pour éviter les problèmes liés à la réservation de bande passante, il est généralement recommandé de disposer d'un *réseau de maintenance* secondaire physiquement séparé du *réseau de production* [Hyp ; Vmw ; Ope]. Ce réseau de maintenance est particulièrement utile pour les centres de données virtualisés car il permet d'isoler naturellement le trafic légitime des VMs du trafic lié à la maintenance de l'infrastructure tel que pour les migrations de VMs mais aussi la maintenance des serveurs (mises à jour, surveillance, etc.). Bien que cette solution soit efficace, elle nécessite néanmoins un raccordement réseau dédié pouvant nécessiter l'achat d'équipements réseaux supplémentaires et reste donc une solution onéreuse.

Le trafic réseau généré par les VMs peut occasionner d'autres scénarios de congestion. Certains services ou applications sont souvent répartis sur plusieurs VMs. Il peut s'agir par exemple d'applications distribuées dont les différents composants logiciels sont hébergés sur plusieurs VMs, ou encore d'un service fortement utilisé et nécessitant des réplicas pour passer à l'échelle. La particularité de ces applications est la grande quantité de trafic émis entre les différentes VMs. Ces VMs qui communiquent fortement entre elles sont généralement regroupées et placées au sein d'un même serveur ou d'une même baie de serveurs. Leur regroupement permet de restreindre l'impact du trafic inter-VMs aux seuls commutateurs des couches basses de l'architecture réseau et ainsi éviter les risques de congestion en réduisant l'utilisation du réseau. Cependant, le risque de congestion n'est pas pour autant écarté et peut survenir de manière inattendue lors d'opérations de maintenance, par exemple lors

du décommissionnement d'une baie de serveurs hébergeant un grand nombre de VMs inter-communicantes. En effet, au fur et à mesure des migrations, l'ordonnancement calculé peut entrainer des situations intermédiaires où le placement des VMs inter-communicantes implique de fortes communications entre la baie d'origine et celle de destination. Dans ce cas de figure, l'ordonnancement des migrations doit alors impérativement tenir compte du trafic inter-VMs pour éviter tout risque de congestion.

3.3 État de l'art

Dans cette section nous décrivons et analysons les différentes études et travaux portant sur l'ordonnancement de migrations. Alors que de nombreuses études se sont concentrées sur le calcul de nouveaux placements des machines virtuelles, peu d'entre elles se sont intéressées à l'ordonnancement des migrations occasionnées par ces nouveaux placements. Nous commençons par une analyse des études abordant les problématiques liées à l'ordonnancement de migrations et les techniques permettant de les adresser. Nous nous concentrons ensuite sur les différentes méthodes d'optimisation proposées pour améliorer les migrations concurrentes de VMs. Finalement, nous discutons de techniques avancées permettant de contrôler et de coordonner des groupes de migrations.

3.3.1 Techniques d'ordonnancement

Kejiang et al. [Ye+12] ont étudié la migration à chaud de groupes de VMs formant une *grappe virtuelle* (VC). Une grappe virtuelle est définit comme un groupe logique de VMs configurées pour travailler ensemble dans un objectif commun. Par exemple dans le cas de *calcul à haute performance* (HPC) ou encore de calcul parallèle, qui nécessitent tout deux la coordination d'un ensemble de VMs. Les VMs d'une même VC partagent souvent une même ressource de stockage et ont la particularité de communiquer fortement entre elles pour échanger des données ou à des fins de synchronisation. Une VM spécifique, appelée VM maître, est généralement désignée pour contrôler l'exécution globale des tâches de la grappe. Les auteurs ont étudié différentes stratégies de migration à chaud et apportent quelques remarques et pistes de réflexions pour optimiser la migration à chaud d'une VC. Premièrement, il apparaît que la migration de la VM maître entraîne une plus forte baisse de performances que pour les autres VMs et que ces dernières doivent être migrées en priorité. Deuxièmement, il est noté qu'une forte parallélisation des migrations réduit drastiquement les performances de la VC et qu'il est nécessaire de trouver le bon niveau de parallélisme qui minimise la perte de performance occasionnée.

L'analyse des performances des migrations a été réalisé sur la base de trois métriques : la durée d'interruption des VMs, la durée totale de migration et la performance globale de la VC. Cependant, aucun indice n'est donné pour calculer le niveau de parallélisation approprié lors de la migration d'un ensemble de VMs. La principale limitation de l'étude concerne la co-localisation supposée des VMs d'une même VC. Bien que le regroupement des VMs au sein d'un même hôte physique améliore les performances en réduisant les latences dues aux communications réseau, la migration d'une VC revient à migrer un ensemble de VMs inter-communicantes d'un hôte physique vers un autre ce qui limite fortement la capacité de parallélisation des migrations comme expliquée en section 3.2.1.2.

Dans une seconde étude, Kejiang et al. [Ye+11] étudient l'efficacité de la migration à chaud de plusieurs VMs entre deux hôtes physiques en testant différentes méthodes de réservation de ressources et stratégies de migration. Les auteurs proposent un système basé sur l'hyperviseur Xen permettant de réserver des ressources mémoire et CPU sur les hôtes pour préparer et optimiser la migration de multiples VMs. Trois métriques sont considérées pour évaluer l'efficacité des différentes stratégies de migration : la durée d'interruption, la durée totale de migration et la réduction des performances des applications. Différentes stratégies de migration sont évaluées et les expérimentations réalisées soulèvent quelques pistes permettant d'optimiser l'ordonnancement de migrations. Il apparaît que la parallélisation des migrations entraîne une perte de performance de l'ensemble des VMs hébergées sur l'hôte source. Cette perte de performance est due à la consommation CPU des processus de migration qui vient concurrencer l'utilisation CPU des autres VMs hébergées sur l'hôte. De façon similaire, l'augmentation de la disponibilité de ressources CPU lors d'une migration a permis de réduire sa durée. Ainsi, cette étude montre qu'une quantité de ressources CPU suffisante doit être disponible sur l'hôte source avant de démarrer une ou plusieurs migrations concurrentes. Cela permet à la fois d'éviter de perturber les autres VMs et de réduire la durée des migrations. Cette étude ne considère toutefois que les migrations entre deux hôtes, les problématiques liées à l'architecture réseau ne sont donc pas prises en compte tel que la réservation de bande passante par exemple. De plus, les tests de parallélisation ne sont réalisés que sur des VMs à très faible activité mémoire afin de paralléliser les migrations sur un même chemin réseau sans perte de performance visible.

Une étude d'Alexander et al. [SS09b] souligne l'importance de prendre en considération la topologie réseau pour l'ordonnancement de migrations à chaud. Pour calculer l'ordonnancement, les auteurs considèrent la topologie réseau, la réservation de bande passante pour les migrations et le type d'applications exécutées sur les VMs. La topologie réseau est représentée sous la

forme d'un arbre hiérarchique et la redondance des connections entre commutateurs offre une capacité de sélection des chemins de migration pour la réservation de bande passante. Un premier modèle simplifié considère un réseau dédié à la migration où la bande passante disponible par lien est stable. Dans ce cas, le problème d'ordonnancement est alors ramené à un problème de répartition de charge où les demandes de bande passante des migrations sont réparties entre les différents liens d'une même couche de l'architecture réseau. Le second modèle envisagé considère cette fois la fluctuation de la bande passante disponible sur les liens réseau et nécessite un contrôle actif de la bande passante allouée aux migrations. Dans ce modèle, l'objectif revient à minimiser les risques de congestion réseau dues aux migrations concurrentes par la prédiction de l'utilisation du réseau et le contrôle actif de la bande passante des migrations. Il est intéressant de noter que, pour ce dernier modèle, les auteurs déconseillent la parallélisation des migrations passant par un même lien réseau afin de prévenir tout risque de congestion. Parmi les idées proposées on retrouve aussi la priorisation des migrations, où les plus importantes ne doivent pas être ralenties ou reportés.

Tusher et al. [ST13b] proposent un algorithme permettant d'ordonnancer un ensemble de migrations. L'objectif étant de calculer un plan de migration qui minimise la durée totale de migration et la durée totale d'interruption des VMs. Les auteurs prennent en compte l'architecture réseau qu'ils modélisent à l'aide d'un graphe non orienté pondéré (*weighted undirected graph*), ainsi que le trafic inter-VMs et le taux de modification des pages mémoire des VMs. L'heuristique proposée consiste à calculer un coût de migration pour chaque VM en se basant sur : l'utilisation CPU et mémoire de la VM, le taux de modification des pages mémoire et la quantité de trafic échangé avec d'autres VMs. Les VMs affichant les coûts de migration les plus faibles sont choisies en priorité pour la migration. Les migrations sont ensuite parallélisées le plus possible, la condition étant que la bande passante allouée à chaque migration reste supérieure au taux de réécriture des pages mémoires de la VM à migrer. Cette condition permet de s'assurer que chaque migration disposera de suffisamment de bande passante pour terminer. La solution proposée considère cependant un simple taux constant d'écriture des pages mémoires ce qui restreint la précision de l'estimation des durées de migration. De plus, les migrations sont massivement parallélisées jusqu'à la limite de leur faisabilité, ces dernières ne bénéficient alors plus de la bande passante maximale disponible sur le chemin de migration. Cette sur-parallélisation des migrations entraîne ainsi une prolongation de la durée totale de migration. Finalement, bien que la quantité de trafic inter-VMs soit prise en compte dans le calcul de l'ordonnancement, les étapes intermédiaires de migration ne sont pas considérées et peuvent entraîner une forte augmentation du trafic réseau et une perte de

performance de l'application tel que décrit en section 3.2.2.

3.3.2 Optimisation des migrations

Dès 2009, Wood et al. proposent *Memory Buddies* [Woo+09], un système de placement de VMs exploitant la fonctionnalité de *partage de page basé sur le contenu* (CBPS) permettant de réduire l'utilisation mémoire des hôtes physiques. Le CBPS est implémenté dans les serveurs ESX de VMware par exemple et consiste à identifier et partager des pages mémoire identiques entre différentes VMs hébergées sur un même hôte physique. L'idée est de supprimer les duplicatas en ne conservant qu'une seule copie partagée entre les différentes VMs et ainsi libérer de la mémoire sur l'hôte. Il s'avère que les VMs ayant le même système d'exploitation, les mêmes applications ou bibliothèques logicielles ont une quantité significative de pages mémoires identiques. Memory Buddies consiste premièrement à identifier les VMs ayant le plus de pages mémoires en commun entre les différents hôtes du centre de données. Un serveur de destination est ensuite sélectionné pour chaque VM de manière à profiter au maximum du partage de pages mémoire, tout en minimisant le nombre de migrations requises. Cette méthode de consolidation a permis par exemple d'augmenter la capacité d'un centre de données de 17 % par rapport à son placement d'origine. Finalement, les VMs sont migrées sur leurs hôtes de destination avec un niveau de parallélisme fixe déterminé manuellement. En effet, les auteurs considèrent que la parallélisation des migrations doit être limitée pour réduire la durée globale de migration. Cependant, la limite de parallélisation est arbitraire et concerne l'ensemble des migrations à réaliser sans tenir compte de la localisation des hôtes ou de la topologie du réseau. La parallélisation automatique des migrations n'est pas considérée ni abordée dans cette étude. De plus, bien que l'utilisation mémoire des hôtes soit réduite, la quantité de pages mémoire transmise pour chaque migration reste la même. En effet, l'identification des pages mémoire communes ne s'effectue qu'après la migration des VMs. Ainsi, l'hôte de destination réceptionne d'abord l'ensemble des pages mémoire des VMs (y compris les duplicatas), puis voit ensuite sa consommation mémoire diminuer graduellement à mesure que le CBPS identifie et partage les pages identiques.

Deux ans plus tard, une étude de Deshpande et al. [DWG11] propose d'exploiter la duplication des pages mémoire entre plusieurs VMs colocalisées pour optimiser leurs migrations respectives. Ici, la technique de dé-duplication des pages mémoire présentée en section 2.2.2 est appliquée pour les migrations concurrentes de VMs colocalisées. Le système proposé a été implémenté dans l'hyperviseur KVM. Il s'agit de traquer le contenu mémoire identique entre les différentes VMs à migrer pour ne l'envoyer qu'une seule fois et ainsi ré-

duire la quantité totale de données envoyées lors des migrations. Une phase de préparation est cependant requise avant la migration des VMs et consiste à pré-calculer le hash des pages mémoires des VMs et identifier les duplicatas. Une version en-ligne à été évaluée pour détecter les pages dupliquées pendant le processus de migration, cependant le temps nécessaire au calcul des hash et à leur comparaison entraîne un ralentissement trop important par rapport aux bénéfices apportés par la dé-duplication. Différentes approches et optimisations ont été étudiées. Premièrement la dé-duplication (basée sur le hachage de contenu) est réalisée selon deux granularité différentes : à l'échelle de la page mémoire entière et de son contenu. En effet, les auteurs appliquent également une compression différentielle sur les pages mémoires quasi-identiques pour exploiter la similarité de leur contenu en ne transférant que leurs différences. Cependant, un compromis doit être trouvé entre la performance de la dé-duplication qui réduit la quantité de données à transmettre, et le temps de calcul des hash qui augmente la durée de migration. Une seconde optimisation a été introduite et consiste à recalculer le hash des pages mémoire modifiées en cours de migration. En effet, le hash des pages mémoire étant calculé pendant la phase de préparation, les pages réécrites durant la migration ne profitent alors plus de la dé-duplication. Ainsi, une approche en-ligne est appliquée pour toutes les pages ayant été modifiées après le début de la migration. Leur technique de dé-duplication a ainsi permis de réduire le trafic réseau, généré par la migration de multiples VMs, de 15 à 18% par rapport à la version existante de KVM. Toutefois, en exploitant la colocalisation des VMs, l'optimisation proposée ne s'applique que pour la migration de plusieurs VMs depuis un unique hôte physique vers un autre. De plus, une parallélisation totale des migrations est nécessaire pour pouvoir profiter au maximum des bénéfices de la dé-duplication.

Dans une seconde étude, Deshpande et al. [DKG12] ont ensuite appliqué la dé-duplication des pages mémoires au niveau de baies entières de serveurs. L'objectif étant d'optimiser les migrations simultanées de VMs au sein d'une même baie de machines vers une second baie. De façon similaire à leur étude précédente [DWG11], les auteurs proposent un mécanisme permettant d'identifier, de traquer et d'éviter la transmission multiple des pages mémoires identiques en provenance des VMs d'une même baie. Pour réaliser la dé-duplication au niveau de la baie, un serveur d'indexation dédié est utilisé pour maintenir une table de hachage globale permettant d'identifier les hôtes disposant de pages mémoires communes (identifiées sous un même hash). Les expérimentations réalisées montrent une réduction du trafic réseau généré par les migrations de 44 % par rapport à la version existante de KVM et de 17 % par rapport à leurs anciens travaux reposant sur une dé-duplication par hôte. Toutefois, l'utilisation et le maintien d'une table de hachage globale au niveau de

la baie à un coût, et on observe une augmentation des durées de migration de 7 % par rapport à une dé-duplication par hôte. De plus, alors que l'infrastructure de test utilisée permet la migration simultanée de l'ensemble des VMs, les capacités du réseau ne permettent pas toujours d'assurer la terminaison de l'ensemble des migrations.

3.3.3 Contrôle de l'ordonnancement

Zheng et al. ont développé Pacer [Zhe+13], un système de gestion de la progression des migrations à chaud implémenté dans l'hyperviseur KVM. Le modèle analytique utilisé pour prédire la durée des migrations a été discuté dans le chapitre précédent en section 2.4.2. Le principe consiste à sonder périodiquement les pages mémoire modifiées en cours de migration pour améliorer la précision de l'estimation au fur et à mesure de la migration. La qualité de prédiction de la fin d'une migration est ainsi adaptée en temps réel lors la migration. La fréquence de mise à jour varie entre 2 secondes et 0.25 secondes pour plus de précision, cette valeur minimale à été définie pour éviter de consommer trop de ressources CPU et ralentir la migration. De plus, Pacer a la capacité d'agir directement sur le processus de migration via une modification de l'implémentation de l'algorithme de pré-copie. Il offre ainsi la possibilité d'ajuster la vitesse de transfert des pages mémoires vers l'hôte de destination en cours de migration. En pratique, la fréquence d'ajustement est fixée à 5 secondes pour éviter de ralentir le processus de migration. Cette capacité d'ajustement de la vitesse de migration couplée à une prédiction adaptative de la durée de migration permet ainsi à Pacer de contrôler précisément le temps de fin des migrations. Le contrôle de la durée de migration en temps réel offre la possibilité à l'administrateur de définir une date de fin précise pour chaque migration qu'il doit effectuer. La garantie de contrôle du temps de fin de migration est cependant de type *best-effort*. En effet, la vitesse de migration peut varier de manière inattendue en fonction du trafic réseau concurrent et de l'activité mémoire variable de certains types d'application. Dans ce cas, les fréquences restreintes d'ajustement de la vitesse de migration (5 secondes) et de la prédiction de durée (jusqu'à 2 secondes) peuvent entrainer une déviation par rapport au temps de fin de migration désiré. Les nombreuses expérimentations effectuées montrent toutefois une déviation acceptable des temps de fin variant de 1 à 5 secondes dans le pire cas pour une migration de plus de 4 minutes. Le contrôle de la vitesse de migration permet par exemple de ralentir une migration pour la terminer à un moment plus opportun. Par exemple lorsque l'activité mémoire de la VM est au plus faible afin de réduire la durée d'interruption de la VM et donc l'impact sur les performances de l'application. Le contrôle du temps de fin des migrations introduit

aussi la possibilité de coordination des migrations. Cela permet par exemple de minimiser les problèmes liés aux migrations de VMs inter-communicantes (les VMs appartenant à une même VC par exemple) tel que décrit en section 3.2.2. En effet, la quantité de trafic inter-VMs qui transite temporairement entre la source et la destination des VMs est due à une mauvaise synchronisation de la fin des migrations. Une bonne coordination des migrations permet alors de minimiser ce type de trafic non désiré qui peut consommer beaucoup de ressources réseau et dégrader fortement la performance des applications. Cependant, malgré de bonnes avancées en termes de contrôle et de coordination des migrations, seules les migrations issues d'un unique hôte physique sont considérées ce qui limite l'intérêt de la solution à des VMs colocalisées uniquement. De plus, le problème de la parallélisation des migrations n'est pas abordé. En effet, coordonner la fin de plusieurs migrations revient à les paralléliser, il est alors nécessaire de disposer d'une bande passante suffisante pour pouvoir garantir la terminaison de l'ensemble des VMs concernées. Des décisions de parallélisation et de groupement des migrations doivent alors être considérées.

Dans la continuité de leurs travaux, Zheng et al. ont ensuite proposé COMMA [Zhe+14], une version améliorée de Pacer offrant plus de possibilités de coordination et d'ordonnancement des migrations. À la différence de Pacer, COMMA bénéficie d'une architecture centralisée permettant de coordonner les migrations depuis différents hyperviseurs KVM et donc en provenances de plusieurs hôtes physiques. L'intérêt de COMMA est entièrement porté vers la migration de VMs inter-communicantes. L'objectif principal consiste alors à minimiser l'impact occasionné par ce type de migration, sans pour autant minimiser leur durée. Pour déterminer les possibilités de parallélisation des migrations, COMMA commence par mesurer la bande passante réseau disponible sur chaque chemin de migration et la quantité de trafic échangé entre les VMs. Dès lors, si la parallélisation de l'ensemble des migrations n'est pas possible, des groupes de migrations parallélisables sont définis de manière à assurer la terminaison de l'ensemble des migrations d'un groupe. En pratique, la règle de groupement des migrations consiste à s'assurer que les taux de modification des pages mémoire de l'ensemble des VMs d'un groupe soient inférieurs à la bande passante disponible sur les chemins de migration. De plus, pour minimiser au maximum le trafic inter-VMs entre les différents groupes (trafic inter-groupes), les groupements sont réalisés de manière à ce que les VMs qui communiquent le plus entre elles soient placées au sein d'un même groupe. L'ordre dans lequel sont réalisées les migrations des groupes de VMs est ensuite déterminé de manière à minimiser le trafic inter-groupes. COMMA dispose d'un contrôle accru de l'implémentation de l'algorithme de pré-copie dans KVM. Il est ainsi capable d'isoler la phase de démarrage, qui (tel que

décrit au chapitre 2) consiste à transférer l'ensemble des pages mémoire utilisées par la VM, de la phase de pré-copie itérative. La phase de démarrage y est identifiée comme statique car la quantité de données à envoyer est fixe et connue avant le début de la migration. Dès lors, afin de minimiser le délai entre la fin de chaque groupe de migrations (et donc minimiser le trafic inter-groupes), COMMA commence par migrer l'ensemble des VMs, tous groupes confondus, simultanément en coordonnant la fin de leur phase de démarrage. En effet, la quantité de données à transférer pendant la phase de démarrage étant statique, aucune bande passante minimale n'est requise pour assurer la terminaison du transfert. La vitesse de transmission de chaque migration est ainsi contrôlée et ajustée pour coordonner précisément la fin de leur phase de démarrage respective. Ainsi, l'ordonnancement intra-groupe et inter-groupes ne commence que lorsque la phase de démarrage de toutes les migrations est terminée. De manière générale, l'ordonnancement inter-groupes est déterminé de façon à minimiser la quantité de trafic échangé entre les différents groupes de VMs. L'ordonnancement intra-groupe, quant à lui, est réalisé de manière à distribuer la bande passante disponible entre les différentes migrations pour qu'elles terminent en même temps et si nécessaire au moment désiré. Ainsi, beaucoup d'efforts ont été entrepris avec COMMA afin d'optimiser et automatiser la migration de VMs inter-communicantes. Les expérimentations réalisées montrent une amélioration significative des performances des applications par rapport à des stratégies standard de migrations parallèles et séquentielles. Tout comme Pacer, le code source n'a cependant pas été rendu disponible. De plus, la nécessité de modifier en profondeur l'hyperviseur KVM et les mises à jours fréquentes de ce dernier, notamment au niveau de l'implémentation de la migration à chaud, rendent sa maintenabilité difficile.

3.4 Discussion

Dans ce chapitre, nous avons introduit le problème d'ordonnancement des migrations de VMs et présenté les différents facteurs à considérer pour prendre des décisions d'ordonnancement efficaces.

Après avoir analysé différents scénarios amenant à ordonnancer un ensemble de migrations, nous avons observé que les critères qualitatifs de l'ordonnancement restent identiques. Des contraintes particulières peuvent toutefois s'ajouter en fonction des besoins spécifiques des administrateurs ou de certaines VMs, tel que pour les VMs inter-communicantes. Il faut ainsi pouvoir définir un objectif commun à tout ordonnancement de migrations, tout en considérant la possibilité de les spécialiser par l'ajout de contraintes annexes et spécifiques à chacun.

Nous avons discuté des particularités de l'infrastructure réseau d'un centre de données et de manière plus générale de son importance dans le choix de parallélisation des migrations. En ayant passé en revue l'état de l'art relatif aux migrations multiples de VMs, nous avons constaté que les décisions de parallélisation des migrations sont souvent négligées. On assiste principalement à une sur-parallélisation des migrations. En effet, lorsque le réseau est ignoré, on observe une parallélisation aveugle et massive des migrations. Lorsque, au contraire, la topologie et les capacités du réseau sont considérées, le niveau de parallélisme est simplement maximisé de façon à assurer la terminaison des migrations. Les durées de migration ne sont alors plus garanties minimales, ce qui est pourtant le critère de performance principal de la migration à chaud. Il est donc nécessaire de définir des règles de parallélisation qui garantissent à la fois une utilisation maximale du réseau et une durée minimale pour chaque migration.

Pour pouvoir calculer des ordonnancements efficaces, il est nécessaire de modéliser l'infrastructure réseau des centres de données en incluant la topologie mais aussi les capacités des liens et des équipements réseau. En effet, le modèle réseau doit permettre de représenter des architectures hiérarchiques mais aussi d'autres types d'architectures plus complexes ainsi que des équipements réseaux bloquants. Dans le chapitre 2, nous avons discuté de la nécessité de disposer d'un modèle efficace de prédiction des performances de la migration. Dès lors, l'utilisation conjointe d'un modèle de performance précis et d'un modèle réseau réaliste permettrait de prendre des décisions d'ordonnancement efficaces exploitant au maximum les capacités du réseau et maximisant les performances des migrations.

Deuxième partie

♢ ♢ ♢

Contributions

Chapitre 4

Modèle d'ordonnancement

Sommaire

4.1 Modèle de migration à chaud 47
4.1.1 Consommation mémoire des hôtes 48
4.1.2 Activité mémoire des VMs 49
4.2 Modèle réseau 55
4.2.1 Description du modèle 55
4.2.2 Bande passante et parallélisation 56
4.3 Résolution du problème d'ordonnancement 59
4.3.1 Gestion de projet à contraintes de ressources 59
4.3.2 Multi-mode RCPSP 59
4.4 Conclusion 60

Ordonnancer un ensemble de migrations consiste à déterminer, pour chacune d'entre elles, le moment de son démarrage et la bande passante à lui allouer. Nous avons vu que les décisions d'ordonnancement doivent s'appuyer sur un modèle de prédiction des performances de migration mais aussi sur un modèle qui puisse représenter l'architecture et les capacités du réseau.

Dans ce chapitre, nous présentons notre modélisation du problème d'ordonnancement de migrations à chaud de VMs. Dans un premier temps, nous décrivons le modèle de migration sous-jacent et notre méthode d'analyse de l'activité mémoire des VMs. Nous décrivons ensuite notre modèle réseau et les différentes analyses qui nous ont guidé vers des règles de parallélisation qui n'affectent pas la performance des migrations. Finalement, nous classifions le problème d'ordonnancement et déduisons la complexité de sa résolution.

4.1 Modèle de migration à chaud

Comme discuté au chapitre 2, établir des prédictions rapides et précises des durées de migration à chaud nécessite la définition d'un modèle analytique approprié qui puisse quantifier l'activité mémoire des VMs. Nous avons ainsi analysé l'activité mémoire de différentes VMs au sein de l'hyperviseur KVM et

défini un modèle de migration permettant d'estimer la durée d'une migration à chaud mais aussi sa faisabilité en fonction de la durée d'interruption maximale désirée.

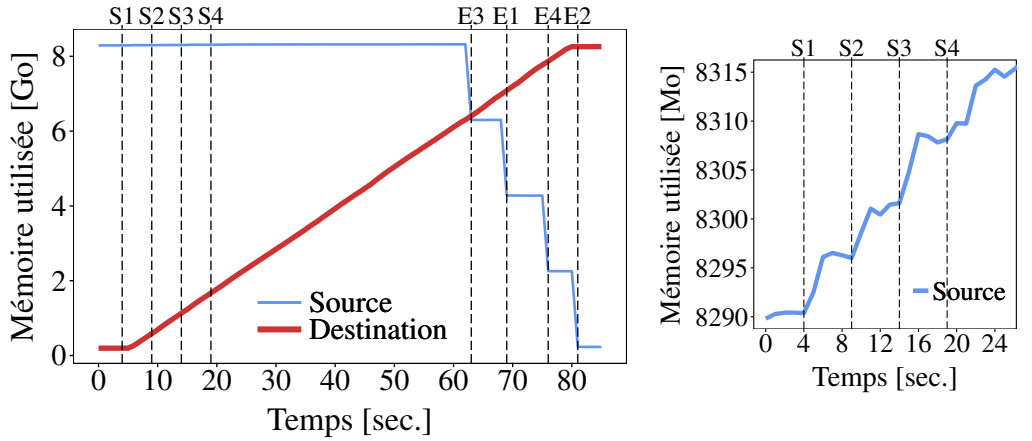
4.1.1 Consommation mémoire des hôtes

La mémoire est la ressource la plus significative pour la modélisation de la migration à chaud. On s'intéresse ici plus particulièrement à la consommation mémoire de l'algorithme de migration et à son impact sur la disponibilité des ressources des hôtes physiques. En effet, pour connaître et contrôler la consommation mémoire de chaque hôte dans le temps, il faut pouvoir modéliser le comportement de la migration de la mémoire entre les hôtes. Il est ainsi d'abord nécessaire d'analyser comment et à quel moment la mémoire de la VM est placée dans la mémoire physique de l'hôte destination et libérée sur son hôte source.

Pour analyser la consommation mémoire des migrations et le comportement du transfert de mémoire entre les hôtes, nous avons réalisé 4 migrations simultanées entre deux hôtes et analysé leur consommation respective. Pour pouvoir observer l'utilisation mémoire de chaque migration séparément, nous les avons démarrées à 5 secondes d'intervalles. L'infrastructure de test est composée de deux hôtes identiques ayant chacun 16 Go de mémoire et 2 CPUs de 4 cœurs chacun. Les VMs ont un unique CPU virtuel et consomment 2 Go de mémoire chacune. Le système d'exploitation des VMs est basé sur la distribution Ubuntu 14.10. Pour éviter de prolonger les durées de migration inutilement, les VMs sont migrées avec une faible activité mémoire et CPU.

La Figure 4.1a représente la consommation mémoire des deux hôtes pendant toute la durée de l'expérimentation. Les migrations sont représentées par les lignes verticales en pointillé où *S1* à *S4* représentent les temps de démarrage et *E1* à *E4* les temps de fin des quatre migrations respectives. Comme nous pouvons le voir, l'hôte source libère l'ensemble de la mémoire utilisée par une VM immédiatement après la terminaison de sa migration. Ce comportement est directement lié à l'implémentation de l'algorithme de pré-copie dans l'hyperviseur KVM. De l'autre côté, le serveur de destination reçoit les pages mémoire de la VM à la vitesse du réseau (1 Gbps) et les place directement en mémoire. Sa propre consommation mémoire augmente ainsi linéairement pendant toute la durée de migration. Par conséquent, aucun délai additionnel n'est requis en fin de migration pour placer la mémoire de la VM dans la mémoire physique de l'hôte.

L'utilisation mémoire des processus de migration est trop faible pour être observée en Figure 4.1a, nous avons ainsi représenté le début des migrations sur une échelle plus réduite en Figure 4.1b. Nous pouvons alors observer que



(a) Consommation mémoire des hôtes (b) Mémoire utilisée pour migrer

FIGURE 4.1 – Aperçu de la consommation mémoire de deux hôtes physique (source et destination) lors de la migration parallèle de 4 VMs. La Figure (b) est une vue rapprochée de la consommation mémoire de l'hôte source lorsque les 4 migrations débutent.

l'utilisation mémoire induite par chaque migration est d'environ 6 Mo. En comparaison à la quantité de mémoire disponible sur les hôtes ou utilisée par les VMs, cette consommation supplémentaire est négligeable. En effet, une quantité de mémoire suffisante (généralement plusieurs centaines de Mo) est laissée disponible sur les hôtes pour les besoins de leur propre système d'exploitation ou encore l'exécution d'outils de gestion locaux par exemple. Cette réservation de mémoire représente un faible pourcentage de la mémoire disponible sur les hôtes et n'affecte donc pas directement la capacité d'hébergement de ces derniers. Par conséquent, l'utilisation mémoire additionnelle générée par le processus de migration est suffisamment faible pour être ignorée dans le modèle de migration.

4.1.2 Activité mémoire des VMs

4.1.2.1 Méthode d'analyse

Pour capturer et analyser l'activité mémoire des VMs, nous avons apporté une modification dans l'espace utilisateur de l'hyperviseur KVM. Inspirée de l'implémentation de l'algorithme de pré-copie, la nouvelle fonctionnalité consiste à démarrer une tâche dédiée pour sonder, synchroniser et réinitialiser l'état des pages mémoires de la VM à des intervalles de temps pré-définis. Dans KVM, une VM est exécutée dans un processus multi-tâches où chaque CPU virtuel ainsi que les entrées/sorties réseaux et disques par exemple sont gérés

par une tâche dédiée. Tout comme lors de la migration d'une VM, la création d'une tâche dédiée par processus permet d'éviter de bloquer l'exécution de la VM. La structure de données existante et représentant l'état des pages mémoire d'une VM est une matrice de bits partagée entre les différentes tâches d'un processus KVM. En activant le traçage de la mémoire d'une VM, cette structure est automatiquement maintenue à jour depuis des appels système périodiques vers l'espace noyau de KVM qui gère la mémoire virtuelle de la VM. La capture doit être réalisée dans des conditions normales d'exécution pour être la plus représentative possible de l'activité mémoire réelle de la VM. Ainsi, la fréquence des captures doit être suffisamment élevée pour observer la progression de l'écriture des pages mémoires de la VM sans toutefois ralentir son exécution. La demande de démarrage de la tâche et les paramètres de capture de l'activité mémoire d'une VM sont transmis à l'hyperviseur au moyen d'une nouvelle commande implémentée pour l'occasion via le *protocole de supervision Qemu* (QMP). La commande QMP est quant à elle directement envoyée depuis l'interface en ligne de commande de l'outil de gestion *virsh*.

4.1.2.2 Faisabilité des migrations

En s'appuyant sur l'analyse de l'activité mémoire d'une VM, il est possible de déterminer si sa migration est réalisable en fonction de la bande passante disponible et la durée d'indisponibilité envisagée. Notre approche consiste à estimer la quantité de pages mémoire transmises pendant la dernière étape de pré-copie pour en déduire la durée d'indisponibilité. Pour cela nous considérons que le nombre de pages mémoire transférées pendant la durée d'indisponibilité d'une VM est égal au nombre de pages modifiées pendant cette même durée. En effet, dans le chapitre 2 nous avons vu que la quantité de données envoyée à chaque itération de pré-copie diminue et converge vers une quantité suffisamment faible pour être transmise pendant la durée d'indisponibilité. Les cycles de synchronisation et de transmission des pages mémoire s'accélèrent alors progressivement pour conserver une vitesse de migration maximale. En pratique, plus la durée d'indisponibilité est faible et plus les cycles sont rapprochés. Ainsi, la durée de transmission précédant la dernière étape d'arrêt et copie est très proche de la durée d'indisponibilité de la VM. C'est pourquoi, avec des durées d'indisponibilité de l'ordre de quelques millisecondes, nous pouvons considérer ces deux durées égales et obtenir des prédictions fiables.

Notre méthode consiste à capturer la quantité de pages mémoire modifiées par la VM sur un intervalle de temps égal à la durée d'indisponibilité désirée. En sachant que chaque page contient 4 Ko de données, nous pouvons directement déduire la quantité de données à transmettre pendant la dernière étape de l'algorithme de pré-copie. Finalement, en connaissant la bande passante

disponible pour la migration, nous pouvons en déduire la durée de transfert et vérifier que celle-ci est bien inférieure à la durée d'indisponibilité maximale envisagée. Ces informations permettent également de déduire la bande passante minimale nécessaire à la faisabilité d'une migration, ou encore la durée d'interruption minimale permettant d'assurer sa terminaison.

Pour simuler une activité mémoire représentative d'une application courante, nous avons utilisé l'outil d'évaluation de serveurs HTTP *httperf*. Notre configuration de httperf consiste à récupérer de manière répétitive une page web statique depuis un serveur HTTP *Apache* local à la VM. La Figure 4.2 représente la quantité de mémoire modifiée par httperf en 30 ms en fonction du nombre de requêtes par seconde généré par httperf. La bande passante requise correspond à la bande passante minimale permettant d'assurer la terminaison de la migration avec une durée d'indisponibilité de 30 ms. Pour représenter la variabilité de l'activité mémoire, chaque point représente la moyenne sur 50 captures successives et l'intervalle de confiance représenté est calculé à plus ou moins l'écart type.

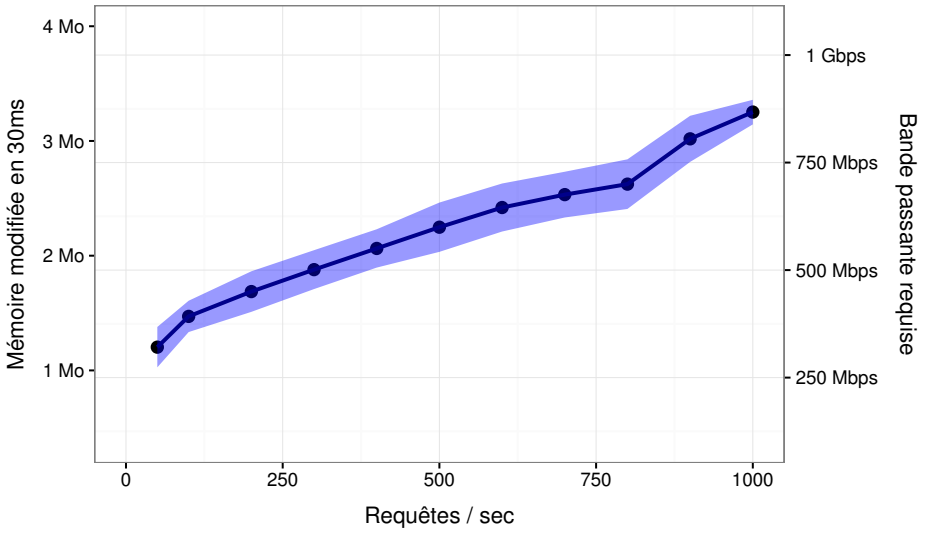


FIGURE 4.2 – Activité mémoire générée par *httperf* en 30 ms.

De manière générale, on observe que plus la fréquence des requêtes générées par httperf est élevée et plus la bande passante nécessaire à la migration est importante. L'analyse de la bande passante requise permet de prendre des décisions de parallélisation par exemple. En effet, on remarque qu'avec un taux supérieur à 300 requêtes par seconde la migration nécessite une bande passante supérieure à 500 Mbps ce qui limite strictement la parallélisation de ce type de VM sur un lien réseau 1 Gbps. A l'inverse, un taux inférieur permet de migrer 2 VMs simultanément, et jusqu'à 3 pour un taux de 50

requêtes par secondes (320 Mbps requis). Ces déductions ont été vérifiées et validées en pratique via une série d'expérimentations entre deux hyperviseurs KVM. L'utilisation mémoire des VMs à été fixée à 2 Go et la durée limite de migration à 10 minutes, ce qui est largement suffisant pour déduire avec certitude que des migrations concurrentes sont irréalisables (de durée infinie).

4.1.2.3 Modélisation de l'activité mémoire d'une VM

D'après la majorité des activités mémoire observées sur des applications de type différentes [Ako+10 ; Liu+11], l'évolution du taux de mémoire modifié peut être séparé en deux phases consécutives. En effet, les pages chaudes sont constamment modifiées à une vitesse très élevée (première phase) tandis que les autres pages mémoire utilisées par la VM sont modifiées à un rythme moins soutenu (seconde phase), nous appelons ces dernières *pages froides*.

La Figure 4.3 représente l'évolution du nombre de pages mémoire uniques modifiées par une VM après une réinitialisation de son état mémoire (matrice de bits à 0) au sein de l'hyperviseur KVM. L'activité mémoire de la VM est générée via l'outil officiel d'évaluation du serveur HTTP Apache nommé *ab* qui permet de simuler plusieurs utilisateurs effectuant des requêtes sur le serveur de façon concurrents. Le nombre d'utilisateurs est fixé à 50 et 100 respectivement.

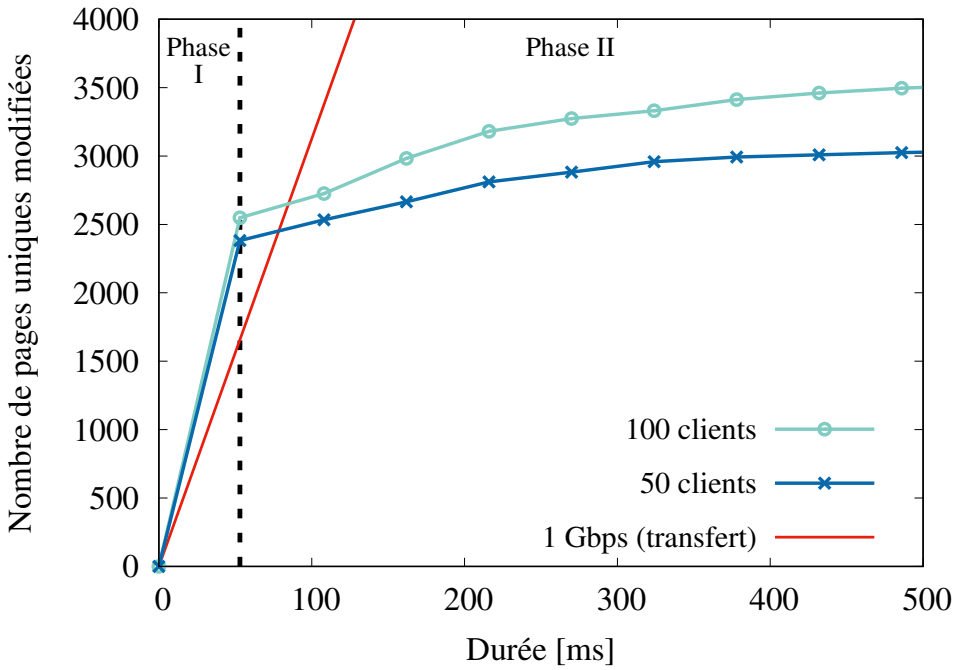


FIGURE 4.3 – Activité mémoire générée par *ab*.

Comme nous pouvons le constater sur la Figure 4.3, la seconde phase est

distinguable par une plus faible variation du taux d'écriture des pages mémoire des VMs. La précision de la mesure est cependant limitée par la fréquence des captures réalisées (50 ms), on estime alors que les pages chaudes sont toutes modifiées au moins une fois dans l'intervalle $[0, 50]$ ms. Dans cette exemple, nous avons limité la fréquence de capture à 50 ms au vu du taux d'écriture élevé des pages mémoire généré par *ab*. En effet, en prenant l'exemple des 100 utilisateurs simultanés, la quantité de pages chaudes est d'environ 2500 pages. En considérant que ces pages (contenant chacune 4 Ko de données) seront transmises à une vitesse de 1 Gbps par exemple, la durée nécessaire à leur transfert est de : $\frac{2500 \times 4 \text{ Ko}}{1 \text{ Gbps}} = 80$ ms. On constate ainsi que la durée requise pour envoyer ces pages mémoire (80 ms) est supérieure à la durée de leur modification (50 ms) ce qui signifie qu'elles devront nécessairement être transférées lors de la dernière étape d'*arrêt et copie* de l'algorithme de pré-copie. Il est alors certain que la durée d'interruption de la VM sera d'au moins 80 ms et il n'est donc pas nécessaire d'effectuer de captures inférieures à 50 ms. Sur la Figure 4.3, la droite rouge représente le temps requis pour transférer les pages mémoires modifiées avec une bande passante de 1 Gbps. Dès lors, la durée d'interruption de la VM est représentée par l'intersection de cette droite avec les courbes d'activités mémoire, soit environ 77 ms d'interruption pour l'activité générée par 50 utilisateurs et 86 ms pour 100 utilisateurs. Ainsi, la migration n'est pas réalisable avec la configuration par défaut de KVM par exemple qui limite la durée d'interruption à 30 ms. La fréquence élevée de modification des pages mémoire s'explique par la forte quantité de requêtes générées par *ab*. En effet, chaque utilisateur simulé effectue des requêtes en continu (une nouvelle requête étant envoyée dès que la réponse de la précédente est réceptionnée), la très faible latence induite par la connexion locale entre *ab* et le serveur HTTP produit alors une charge considérable au sein de la VM.

Bien que la fréquence d'écriture des pages mémoire varie en fonction du type d'application et de son utilisation, les fréquences d'écriture des pages chaudes et froides doivent être considérées constantes pour pouvoir réaliser des prédictions rapides des durées de migration. Ainsi, la progression de l'écriture des pages chaudes et froides sont considérées linéaires. Comme illustré précédemment, la quantité de pages chaudes notée PC_s en Mo et le temps nécessaire à leur écriture noté PC_d en secondes donnent un bon aperçu de la bande passante minimale nécessaire pour assurer la terminaison d'une migration. En pratique, prédire la faisabilité d'une migration consiste à mesurer PC_s sur une période de temps équivalente à la durée d'interruption maximale désirée D et de vérifier que $\frac{PC_s}{D}$ est inférieure à la bande passante disponible sur le chemin de migration. Le taux d'écriture des *pages froides* noté PF_r en Mo/s peut être mesuré après $t = PC_d$. Bien que souvent très faible, PF_r est tout de

même dépendant du type d'application s'exécutant sur la VM et de l'activité mémoire qu'elle génère. Tout comme les pages chaudes, les pages froides sont considérées modifiées dès le début du processus de migration. Ainsi, le taux d'écriture des pages chaudes s'écrit : $PC_r = \frac{PC_s}{PC_d} - PF_r$. Étant donné une migration $m \in \mathcal{M}$, avec $mu(m)$ la quantité de pages mémoire utilisée par la VM en Mo et $bw(m)$ la bande passante allouée pour la migration en Mo/s, la durée minimale de migration d_{min} en secondes s'écrit : $d_{min}(m) = \frac{mu(m)}{bw(m)}$. Dès lors, si l'on considère que la quantité totale de pages froides modifiées pendant le processus de migration notée PF_s en Mo est toujours inférieure à $mu(m)$, alors PF_s s'écrit : $PF_s = d_{min}(m) \times PF_r$. De manière générale, pour transférer une quantité de mémoire X avec une bande passante Y et un taux d'écriture mémoire Z, la durée de transfert peut s'écrire $\frac{X}{Y-Z}$. Le temps passé à envoyer les pages froides d_{PF} s'écrit alors :

$$d_{PF}(m) = \frac{PF_s}{bw(m) - PF_r} \tag{4.1}$$

Et le temps passé à envoyer les pages chaudes d_{PC} s'écrit :

$$d_{PC}(m) = \frac{PC_s}{bw(m)} + \frac{PC_s - \big(D \times bw(m)\big)}{bw(m) - PC_r} \tag{4.2}$$

Où $\frac{PC_s}{bw(m)}$ représente la première transmission des pages chaudes et $D \times bw(m)$ correspond à la quantité de données à envoyer après l'arrêt de la VM sur l'hôte source. Si cette valeur est supérieure à la quantité de pages chaudes mesurée ($D \times bw(m) > PC_s$), alors il ne sera pas nécessaire d'envoyer itérativement de pages chaudes pour respecter la durée d'interruption désirée et le calcul est alors simplifié :

$$d_{PF}(m) + d_{PC}(m) = \frac{PF_s - \Big(\big(D \times bw(m)\big) - PC_s\Big)}{bw(m) - PF_r} + \frac{PC_s}{bw(m)} \tag{4.3}$$

Finalement, la durée de migration $d(m)$ s'écrit :

$$d(m) = d_{min}(m) + d_{PF}(m) + d_{PC}(m) + D \tag{4.4}$$

La durée d_{min} est le facteur dominant la durée totale de migration $d(m)$. En effet, d_{min} est généralement exprimée en dizaines de secondes ou en minutes alors que $d_{PF}(m)$ et $d_{PC}(m)$ sont de l'ordre de quelques secondes. Finalement,

la durée d'interruption D à un poids très faible dans la durée de migration car elle est généralement exprimée en millisecondes (30 ms par défaut dans KVM). La durée d'interruption peut ainsi être ignorée dans le calcul de la durée de migration qui est généralement arrondie à la seconde près.

4.2 Modèle réseau

La migration consiste à transférer une VM d'un hôte physique vers un autre au travers du réseau. Pour des raisons principalement économiques, nous avons vu que les réseaux sont rarement non-bloquants. Notre modèle réseau représente ainsi le trafic généré par chaque migration au cours du temps, au travers d'un ensemble d'éléments réseaux aux capacités limitées. La notion de parallélisation étant inhérente au problème d'ordonnancement de migrations, nous avons également défini des règles de parallélisation qui garantissent des performances de migration optimales.

4.2.1 Description du modèle

Étant donné que le prochain placement des VMs est connu, le modèle considère qu'une VM migre de son hôte source vers sa destination au travers d'une route prédéfinie. La bande passante allouée à une migration est supposée constante et les liens réseaux sont considérés en duplex intégral (*full-duplex*). Finalement, le modèle ignore la latence du réseau, ce qui signifie qu'une migration occupe simultanément tout les éléments réseau de son chemin. Cette hypothèse est cohérente car la précision de notre modèle est de l'ordre de la seconde alors que la latence induite entre deux hôtes d'un centre de données est de l'ordre de la microseconde.

Le modèle réseau considère un ensemble de migrations $\mathcal{M} \subseteq \mathcal{A}$ à réaliser au travers d'un ensemble d'éléments réseau $\mathcal{N}$ (interfaces réseau, commutateurs, etc.). Pour tout élément $n \in \mathcal{N}$, $capa(n)$ désigne sa capacité maximale de traitement en Mbps. Pour toute migration $m \in \mathcal{M}$, $path(m) \subseteq \mathcal{N}$ indique les éléments réseau traversés par la migration (hôtes source et destination inclus), $bw(m)$ désigne la bande passante allouée, $st(m)$ et $ed(m)$ indiquent respectivement le début et la fin de l'opération. L'équation (4.5) modélise finalement le partage de la capacité d'un élément réseau entre les différentes migrations qui le traversent :

$$\sum_{\substack{m \in \mathcal{M},\ n \in path(m),\\ t \in [st(m);\ ed(m)]}} bw(m) \quad < \quad capa(n) \tag{4.5}$$

Le modèle réseau reste ainsi très générique et permet de modéliser tout type d'architecture réseau qu'elle soit hiérarchique ou non. Il permet de maîtriser la bande passante disponible sur chaque chemin de migration en gérant le partage des capacités des éléments réseaux entre les migrations. Néanmoins, ce modèle réseau seul n'est pas suffisant pour décider de la parallélisation des migrations par exemple. C'est pourquoi nous avons étudié plus en détail le comportement des migrations en fonction de la bande passante allouée et proposé des règles de parallélisation permettant d'assurer de bonnes performances de migration.

4.2.2 Bande passante et parallélisation

L'analyse de l'activité mémoire des VMs couplée à la modélisation du réseau permet de prendre des décisions de parallélisation qui garantissent la faisabilité des migrations tel qu'illustré en section 4.1.2.2. Il est également possible de rajouter des marges d'erreurs aux calculs des bandes passantes minimales requises par les migrations pour augmenter la fiabilité de l'approche par exemple. Toutefois, nous allons voir que la parallélisation des migrations ne doit pas uniquement s'appuyer sur la faisabilité de ces dernières mais aussi sur leurs performances respectives.

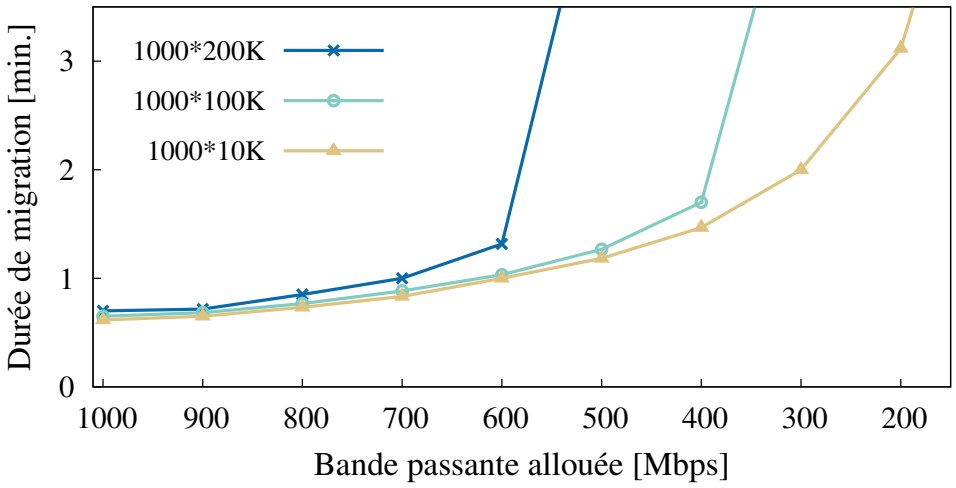


FIGURE 4.4 – Durées de migration à chaud entre 2 hyperviseurs KVM.

La Figure 4.4 représente la durée de migration d'une VM en fonction de la bande passante qui lui est allouée. La VM utilise 4 Go de mémoire et la durée d'indisponibilité maximale pour sa migration est fixée à 30 ms. Trois activités mémoire différentes ont été simulées à l'aide de la commande *stress*, le paramètre *1000*10K* indique par exemple l'exécution concurrente

de 1000 tâches dont chacune alloue puis libère continuellement 10 Ko de mémoire. On constate premièrement que la durée de migration augmente de façon exponentielle lorsque la bande passante diminue de façon linéaire. Deuxièmement, plus l'activité mémoire de la VM est élevée et plus l'augmentation de la durée est significative. En effet, tout comme l'activité mémoire des VMs, la bande passante disponible pour les migrations impacte le comportement itératif du processus de migration à chaud. Plus précisément, lors d'une itération de pré-copie, c'est la durée de transmission des pages modifiées qui détermine indirectement la quantité de pages à envoyer lors de la prochaine itération. Ainsi, en réduisant la bande passante d'une migration, on augmente par effet de bord la durée de transmission des pages modifiées et donc la quantité de données à envoyer à travers le réseau. En outre, plus l'activité mémoire de la VM est intense et plus cet effet de bord est important.

Pour conserver de bonnes performances de migration, les décisions de parallélisation doivent donc être prises de manière à conserver une bande passante maximale pour chaque migration. Ainsi, les opportunités de parallélisation concernent généralement des VMs en provenance et à destination d'hôtes distincts. En effet, paralléliser revient à partager la capacité d'un lien entre plusieurs migrations. Ce lien doit alors au moins disposer d'une capacité égale à la somme des bandes passantes maximales disponibles sur chaque chemin de migration. Ainsi, les VMs en provenance ou à destination d'un même hôte ne doivent être parallélisées que si la capacité de transfert de l'hôte le permet (capacité n fois supérieur au lien le plus faible, n étant le nombre de migrations concurrentes), ce qui est très rarement le cas dans les réseaux hiérarchiques actuels (voir chapitre 3).

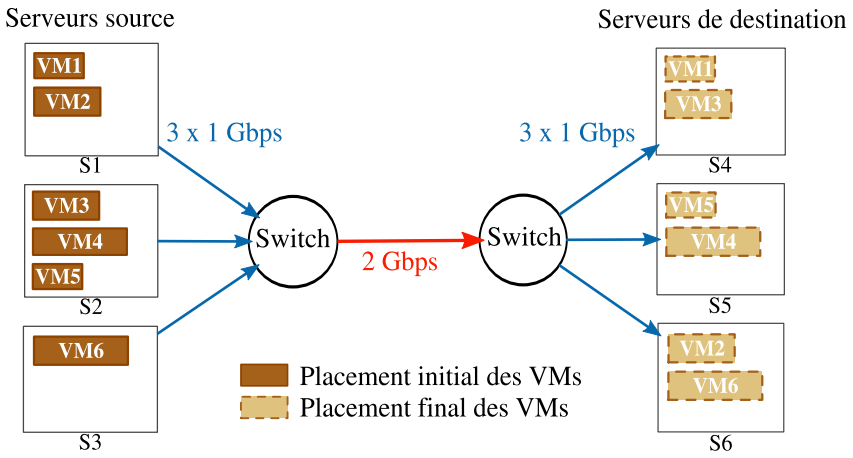


FIGURE 4.5 – Exemple d'un scénario de décommissionnement.

Prenons l'exemple du scénario de décommissionnement représenté en Figure 4.5. Ce dernier consiste à migrer 6 VMs depuis 3 hôtes sources vers 3

hôtes de destination au travers d'un réseau bloquant. Les hôtes sources et destinations sont connectés à leur commutateur ToR respectifs via des liens 1 Gbps, et les deux commutateurs sont directement reliés entre eux via une connexion de 2 Gbps. Le diagramme de Gantt en Figure 4.6 représente un plan d'ordonnancement idéal pour ce scénario. En effet, en migrant les VMs

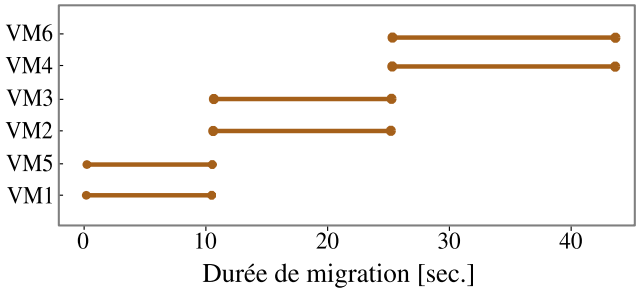


FIGURE 4.6 – Ordonnancement idéal pour le scénario en Figure 4.5.

2 par 2, l'ordonnancement représenté parallélise les migrations de façon à utiliser la pleine capacité du lien inter-commutateurs tout en conservant une bande passante maximale pour chaque migration. On constate d'abord que les opportunités de parallélisation tendent à apparaître principalement sur des couches supérieures de l'architecture réseau dont les liens ont des capacités de transfert plus importantes. De plus, les groupes de VMs migrées en parallèle sont sélectionnés avec précaution depuis deux hôtes distincts et à destination de deux hôtes distincts pour éviter le partage de la capacité d'un lien faible de 1 Gbps entre plusieurs migrations et conserver des vitesses de migration maximales. Finalement, on observe que les migrations sont réalisées par groupes de durées identiques. En effet, grouper les migrations par durée permet de maintenir une parallélisation maximale du début jusqu'à la fin de l'opération et ainsi garantir une durée totale de migration minimale. Ainsi, les migrations parallèles doivent être sélectionnées en fonction de la topologie et des capacités du réseau pour minimiser la durée de chaque migration, mais aussi en fonction de leurs durées respectives pour minimiser la durée totale de l'opération. Néanmoins, dans le cas particulier de VMs fortement inter-communicantes par exemple, tel que présenté au chapitre 3, il peut être préférable de paralléliser massivement les migrations jusqu'à la limite de leur faisabilité. Certaines contraintes particulières peuvent ainsi venir invalider les règles de parallélisation décrites ci-dessus, c'est pourquoi il est nécessaire de disposer d'une approche flexible qui puisse prendre des décisions d'ordonnancement efficaces tout en intégrant les contraintes spécifiques des opérateurs.

4.3 Résolution du problème d'ordonnancement

Calculer le moment de démarrage de chaque migration sans excéder les capacités des éléments réseaux réfère au problème bien connu de *gestion de projet à contraintes de ressources* (RCPSP) [BLK83], où chaque migration est une activité à réaliser et un élément réseau est une ressource partageable et renouvelable. L'objectif est de trouver un ordonnancement, c'est-à-dire un ensemble de dates de démarrage pour chaque activité, qui respecte toutes les contraintes et dont la durée totale est minimum (la durée d'un ordonnancement étant définie par la différence entre la plus grande des dates de fin et la plus petite des dates de début).

4.3.1 Gestion de projet à contraintes de ressources

Le RCPSP est un problème d'optimisation très combinatoire qui recouvre un grand nombre de situations d'ordonnancement. Le problème est NP-difficile au sens fort et ne peut donc être résolu optimalement que par des algorithmes de complexité exponentielle. Depuis plus de 30 ans, de nombreuses études ont été consacrées au RCPSP et diverses méthodes de résolution ont été proposées :

- Des *heuristiques* permettant de calculer rapidement des solutions réalisables mais sans être nécessairement optimales.
- Des *métaheuristiques* essayant d'approximer la meilleure solution, les algorithmes génétiques figurant parmi les métaheuristiques les plus performantes pour la résolution de ce problème.
- Des méthodes d'*optimisation linéaire* où la fonction à minimiser mais aussi les contraintes sont décrites par des fonctions linéaires.
- Des algorithmes de *séparation-évaluation* consistant à diviser le RCPSP en plusieurs sous-problèmes pour les résoudre individuellement.
- La *programmation par contraintes* (PPC) est également employée pour résoudre ce type de problème d'ordonnancement. En PPC, on sépare la partie modélisation de la partie résolution à l'aide de *problèmes de satisfaction de contraintes* (CSPs). La particularité de cette approche réside dans l'utilisation active des contraintes du problème pour réduire la taille de l'espace des solutions à parcourir (on parle alors de propagation de contraintes).

4.3.2 Multi-mode RCPSP

Le problème d'ordonnancement de migrations, à la différence d'un RCPSP classique, réside dans le lien entre la durée d'une migration et la bande pas-

sante qui lui est allouée. En effet, la durée d'une migration varie en fonction de la bande passante disponible sur son chemin de migration. Cependant, dans un RCPSP classique, la durée d'une activité est considérée constante et donc indépendante de la capacité des ressources qu'elle utilise.

L'ordonnancement de migrations réfère finalement à une variante du RCPSP appelée *Multi-mode RCPSP* (MRCPSP) où chaque activité dispose d'un ensemble de modes disponibles, on parle alors d'activités multi-mode. Ainsi, pour une activité donnée, chaque mode définit une durée propre et une demande en ressources spécifique, le choix d'un mode pour l'exécution d'une activité fait alors entièrement partie du problème d'ordonnancement. Dans le contexte d'ordonnancement de migrations, chaque mode d'une migration correspond alors à un couple de bande passante allouée et de la durée de migration associée. Finalement, que ce soit en RCPSP classique ou multi-mode, minimiser le temps de fin de l'opération est connu pour être un problème NP-difficile dans le sens fort du terme [KSD95]. Le MRCPSP augmente encore la complexité du problème et il est clair que le temps de calcul d'une solution augmentera exponentiellement avec le nombre de VMs et d'éléments réseau. Il est ainsi nécessaire de réduire la complexité du problème en minimisant par exemple le nombre de modes disponibles par activité et en proposant des heuristiques qui permettent de trouver rapidement des solutions de bonnes qualités.

4.4 Conclusion

Le modèle d'ordonnancement que nous proposons est basé à la fois sur un modèle de migration à chaud permettant de prédire rapidement la durée des migrations et sur un modèle réseau permettant de gérer le partage des ressources réseaux dans le temps.

Nous avons analysé le comportement des migrations et défini des règles de parallélisation. Premièrement, nous avons observé que le partage de la bande passante maximale disponible pour une migration entraîne inévitablement une perte de performance quelle que soit l'activité mémoire de la VM. Ainsi, des migrations sont parallélisables efficacement si et seulement si la bande passante maximale disponible sur chaque chemin de migration n'est pas réduite. Deuxièmement, nous avons constaté que les opportunités de partage des ressources réseaux apparaissent principalement au niveau des couches hautes et intermédiaires de l'architecture réseau, ce qui limite drastiquement la parallélisation de migrations entre deux hôtes distincts. Troisièmement, lorsque la parallélisation des migrations est possible, c'est à dire sans affecter la performance de ces dernières, le groupement des migrations par durées est un

moyen sûr et efficace pour minimiser le temps total de migration. Cependant, il faut tout de même rester flexible et permettre d'intégrer des contraintes particulières en cas de situations spécifiques qui nécessitent par exemple une parallélisation massives des migrations.

Finalement, le problème d'ordonnancement de migrations se réfère au MRCPSP, un problème d'optimisation combinatoire NP-difficile dont nous avons présenté brièvement les différentes méthodes de résolution proposées dans l'état de l'art. Nous avons ainsi besoin de trouver des optimisations et définir des heuristiques qui nous permettent de trouver rapidement des solutions de qualités qui soient les plus proches possibles de l'optimale.

Chapitre 5

Implémentation dans BtrPlace

Sommaire

5.1 BtrPlace 64
5.1.1 La programmation par contraintes 64
5.1.2 Modélisation du problème de reconfiguration 65
5.1.3 Objectif de la résolution 65
5.2 Implémentation de l'ordonnancement 66
5.2.1 Modèle réseau 66
5.2.2 Modèle de migration 68
5.2.3 Configuration et application de l'ordonnancement . . . 70
5.3 Optimisations 71
5.3.1 Bande passante maximale 71
5.3.2 Heuristique 72
5.4 Extensions 73
5.4.1 Algorithme de post-copie 73
5.4.2 Contrôle temporel 74
5.4.3 Modèle énergétique 76
5.5 Conclusion 78

Notre modèle d'ordonnancement des migrations a été intégralement implémenté en Java au sein du gestionnaire de VMs *open-source* BtrPlace. Ce dernier permet de calculer la meilleure séquence de migrations ainsi que de n'importe quelles actions nécessaires à la reconfiguration d'un centre de données tout en satisfaisant des contraintes en continu.

Dans ce chapitre, nous présentons d'abord BtrPlace et son approche à base de programmation par contraintes. Nous décrivons ensuite l'implémentation de notre modèle d'ordonnancement et les améliorations que nous lui avons apportées pour améliorer ses performances. Finalement, nous détaillons les extensions développées pour par exemple contrôler l'ordonnancement via des contraintes temporelles ou encore l'intégration d'un modèle énergétique.

5.1 BtrPlace

BtrPlace [HLM13] est un gestionnaire de VMs *open source* écrit en Java qui peut être configuré dynamiquement via un ensemble de contraintes prédéfinies. Son but est de maintenir un centre de données dans une configuration viable, dans laquelle le placement des VMs et l'état des différents hôtes respecte l'ensemble des contraintes fournies. Initialement basé sur Entropy [Her+09], BtrPlace bénéficie d'une réécriture complète de l'algorithme de reconfiguration et offre de meilleures performances et une flexibilité accrue. Il comporte également son propre langage de script pour exprimer les contraintes de manière déclarative. L'utilisation de scripts de configuration permet par exemple de simplifier le travail des administrateurs qui peuvent alors se concentrer sur la définition d'un placement viable des VMs plutôt que sur la manière d'y parvenir. BtrPlace fournit également une API REST au format JSON qui facilite son utilisation et sa portabilité.

BtrPlace consiste à calculer, en fonction de contraintes et objectifs spécifiques, le prochain placement d'un ensemble de VMs, l'état futur des hôtes et le plan d'actions qui permet d'y conduire. BtrPlace utilise la programmation par contraintes [RVBW06] et repose plus particulièrement sur la bibliothèque Java nommée Choco [JRL08] pour modéliser le placement des VMs, la planification des actions et résoudre le problème associé.

5.1.1 La programmation par contraintes

La *programmation par contraintes* (PPC) est une approche permettant de modéliser et de résoudre des problèmes combinatoires représentés par des relations logiques qui doivent être satisfaites par la solution calculée. Une caractéristique fondamentale de la PPC est la séparation de la partie modélisation de la partie résolution. L'algorithme de résolution est ainsi indépendant des contraintes qui composent le problème et de l'ordre dans lequel elles sont fournies. En PPC, un problème est modélisé comme un CSP comprenant un ensemble de *variables de décision* et de *contraintes*. Une contrainte représente une relation entre une ou plusieurs variables qui limite les valeurs que peuvent prendre simultanément chacune d'entre elles.

Les algorithmes de recherche de solution, appelés *solveurs* de contraintes, s'appuient sur la *propagation de contraintes* pour réduire le nombre de solutions à explorer, ainsi que sur une recherche exhaustive des différentes affectations possibles pour chaque variable. Ces algorithmes garantissent ainsi de trouver une solution lorsqu'elle existe mais aussi de prouver qu'il n'en existe pas si aucune solution n'a été trouvée à la fin de la recherche. Un solveur calcule une solution pour un CSP en assignant une valeur à chaque variable de

manière à satisfaire simultanément l'ensemble des contraintes. Un CSP peut être également complété avec un objectif représenté par une variable dont la valeur doit être maximisée ou minimisée.

5.1.2 Modélisation du problème de reconfiguration

BtrPlace prend en entrée la configuration du centre de données, incluant le placement des VMs et l'état des hôtes, ainsi qu'un ensemble de contraintes à satisfaire. À partir du placement des VMs et de l'état des hôtes, BtrPlace modélise d'abord un *problème de reconfiguration* (RP) cœur qui consiste en un algorithme de placement et d'ordonnancement minimal qui manipule les hôtes et les VMs au travers d'actions. Chaque action est modélisée en fonction de sa propre nature (démarrage, migration ou arrêt d'une VM, démarrage et arrêt d'un hôte). Une action $a \in \mathcal{A}$ inclut au moins les variables $st(a)$ et $ed(a)$ qui dénotent respectivement son moment de démarrage et de fin. Ces moments sont exprimés en secondes et sont relatifs à l'instant de démarrage de la reconfiguration ($t = 0$). Une fois le RP cœur généré, BtrPlace le personnalise à l'aide de toutes les contraintes définies ainsi que l'objectif s'il est fourni. Le RP spécialisé qui en résulte est ensuite résolu pour générer un plan d'action à appliquer.

La PPC étant composable, il est possible de brancher des modèles externes au dessus du RP cœur pour, par exemple, supporter des éléments supplémentaires d'un centre de données tels que les éléments qui composent le réseau, ou encore des préoccupations additionnelles telles que la consommation énergétique produite par l'exécution de chaque action. Il est ainsi possible d'utiliser un modèle alternatif pour chaque type d'action.

5.1.3 Objectif de la résolution

La définition d'un objectif de résolution n'est pas obligatoire, le solveur de contraintes se contente alors de retourner la première solution trouvée qui satisfasse l'ensemble des contraintes définies. L'objectif intervient lorsque l'on cherche à calculer une solution optimale à un problème donné. Il est alors représenté par une variable (généralement appelée variable de coût) dont il s'agit de maximiser ou minimiser la valeur. Ainsi, en alternant algorithmes de filtrage par contraintes et mécanismes de recherche, Choco calcule de manière itérative des solutions de qualités croissantes par rapport à l'objectif fixé et finit par retourner la solution optimale. En effet, pour minimiser (resp. maximiser) une variable K, Choco travaille de manière incrémentale : à chaque fois qu'une solution avec un coût associé k est calculé, Choco ajoute automatiquement la contrainte $K < k$ (resp. $K > k$) et essaie de calculer une nouvelle

solution. La contrainte ajoutée assure ainsi que la prochaine solution aura une meilleure valeur d'objectif. Ce processus se répète jusqu'à ce que Choco ait parcouru l'ensemble de l'espace de recherche ou atteigne un délai maximal donné. Le solveur retourne finalement la dernière (et donc la meilleure) solution trouvée.

BtrPlace définit un objectif global nommé *MinMTTR* dont l'intuition consiste à réaliser des actions de courte durée et à les exécuter le plus tôt possible pour générer des ordonnancements rapides. Il est implémenté de la façon suivante :

$$\min\left(\sum_{a\in\mathcal{A}} ed(a)\right)$$

Ainsi, *MinMTTR* consiste à minimiser la somme des temps de fin de l'ensemble des actions à réaliser. Pour minimiser cette somme, le solveur va chercher à démarrer les actions au plus tôt en instanciant les variables $st(a)$ à leurs valeurs les plus faibles. De plus, si la durée d'une action (représentée par $ed(a) - st(a)$) n'est pas constante, le solveur va commencer par instancier les différentes variables du problème à des valeurs permettant d'obtenir des durées d'actions minimales. L'objectif permet ainsi à l'utilisateur de statuer sur sa propre définition d'une solution optimale et de guider le solveur vers cette dernière.

5.2 Implémentation de l'ordonnancement

L'implémentation de notre modèle d'ordonnancement de migrations au sein de BtrPlace inclut la définition d'un modèle réseau générique auparavant inexistant et l'amélioration du modèle de migration permettant d'estimer les durées de migration à chaud. Nous avons ainsi introduit ces modèles à l'intérieur du RP cœur de BtrPlace.

Dans cette section, nous présentons d'abord les détails d'implémentation du modèle d'ordonnancement présenté au chapitre 4. Nous décrivons ensuite les différentes étapes de configuration du nouveau modèle ainsi que notre méthode d'exécution du plan d'actions généré par BtrPlace.

5.2.1 Modèle réseau

Le modèle réseau a pour principal objectif de déterminer la bande passante utilisable par les migrations via la représentation de l'architecture réseau et des capacités de transfert du centre de données. L'allocation de bande passante aux migrations est régie par deux principes basiques. Premièrement, la bande

passante maximale disponible pour une migration est limitée par l'élément réseau traversé ayant la plus faible capacité. Deuxièmement, lorsque plusieurs migrations concurrentes traversent un même élément réseau, la bande passante disponible sur ce dernier doit être partagée de manière équitable entre les migrations.

Nous avons choisi de modéliser le partage de bande passante entre migrations à l'aide de contraintes cumulatives [AB93]. Une contrainte cumulative consiste à placer un ensemble de tâches sur des ressources bornées. Chaque tâche agrège trois variables : une hauteur, une durée et un moment de démarrage. La contrainte assure alors que, à tout moment, la hauteur cumulée des tâches placées sur une ressource n'excède pas la hauteur de la ressource. Ainsi, dans notre modèle d'ordonnancement, une tâche représente une migration dont la hauteur est la bande passante allouée et une ressource correspond à un élément réseau dont la hauteur représente sa capacité maximale.

Nous avons défini deux types d'éléments distincts qui composent le modèle réseau. Le premier est le *lien réseau*. Pour représenter à la fois des liens de canaux *semi duplex* et *duplex intégral*, nous avons utilisé respectivement une et deux contraintes cumulatives pour représenter la capacité du lien en fonction de son mode de transmission. En effet, avec un canal de communication semi duplex, la transmission de données est bidirectionnelle mais elle ne peut s'effectuer simultanément que dans une seule direction. Ainsi, la capacité d'un lien *semi duplex* représente la bande passante globale atteignable sur le lien indépendamment du sens de communication, et ne requiert ainsi qu'une seule contrainte cumulative. A l'opposé, un lien en *duplex intégral* requiert deux contraintes cumulatives séparées pour représenter la bande passante maximale atteignable dans ses deux sens de communication qui peuvent être empruntées simultanément par différentes migrations. Le second élément est le *commutateur réseau* utilisé pour connecter les hôtes et les liens entre eux. Dans le cas d'un commutateur bloquant uniquement, sa capacité de commutation de fond de panier maximum est représenté par une simple contrainte cumulative. La quantité de liens et d'hôtes que peut interconnecter un même commutateur n'est pas limité et reste totalement indépendant de sa capacité de commutation.

Modéliser son réseau revient à déclarer l'ensemble des commutateurs puis à les interconnecter de manière à représenter l'architecture désirée. Pour reproduire un réseau existant le plus précisément possible, il est alors nécessaire de disposer d'une bonne connaissance de l'architecture et des caractéristiques des équipements qui le composent. Ainsi, pour faciliter la description de réseaux, nous avons ajouté la possibilité d'importer ces informations via le format de

description de la plateforme SimGrid [1] [Cas+14]. L'ensemble des informations nécessaires à la modélisation du réseau peut alors être récupéré à l'aide d'outils d'analyse et de surveillance du réseau puis converti au format SimGrid pour être importé dans BtrPlace. Finalement, si aucun réseau n'est spécifié, un modèle générique est automatiquement défini et consiste en un unique commutateur non-bloquant qui interconnecte l'ensemble des hôtes via des liens à 1 Gbps.

Pour pouvoir utiliser le modèle réseau, nous devons cependant considérer que le placement futur des VMs a déjà été calculé en amont, par un algorithme de placement par exemple, puis fourni en entrée sous forme de contraintes. En effet, par défaut, lorsqu'une contrainte quelconque est violée par le placement initial des VMs, BtrPlace peut décider de réaliser une ou plusieurs migrations qui n'étaient peut-être pas prévue par l'administrateur. Dans ce cas, BtrPlace sélectionne un nouvel hôte pour chaque VM concernée de manière à satisfaire l'ensemble des contraintes définies. Lorsque plusieurs hôtes candidats permettent d'héberger une VM, BtrPlace en sélectionne un de manière aléatoire et instancie la variable représentant le placement futur de la VM. Cependant, la définition du modèle réseau nécessite de connaître à l'avance l'hôte de destination des VMs qui nécessitent une migration. En effet, la variable représentant le placement futur d'une VM doit être instanciée pour pouvoir déterminer son chemin de migration et ainsi placer la tâche correspondante sur les contraintes cumulatives représentant les éléments réseaux traversés. Pour ce faire, une contrainte de placement existante nommée *Fence* qui consiste à déterminer le futur placement d'une VM doit être spécifiée pour chaque VM nécessitant une migration. Toutefois, pour conserver la sélection automatique des hôtes de destination, nous avons introduit une résolution facultative en deux phases dans BtrPlace. La première phase consiste à résoudre le problème en retirant le modèle réseau pour obtenir un placement final viable de manière automatique. La seconde phase analyse le placement calculé, insère les contraintes *Fence* correspondantes dans le problème initial, réintroduit le modèle réseau puis calcule finalement l'ordonnancement en disposant alors de l'ensemble des contraintes de placement nécessaires à l'utilisation du modèle réseau.

5.2.2 Modèle de migration

Nous avons implémenté le modèle de migration tel que décrit au chapitre 4. L'estimation de la durée d'une migration nécessite 4 nouvelles informations qui doivent être définies pour chaque VM à migrer. Ces informations sont

1. `http://simgrid.gforge.inria.fr/simgrid/3.9/doc/platform.html`

spécifiées dans le modèle de BtrPlace sous la forme de paramètres dédiés directement attachés aux VMs.

Premièrement, la mesure de l'utilisation mémoire effective de la VM est essentielle pour obtenir une durée de migration réaliste. Cette information nous permet de déterminer la durée de migration minimale et influe grandement sur le nombre d'itérations de l'algorithme de pré-copie. Néanmoins, la récupération de cette information n'est pas toujours triviale. En effet, l'auto-ballonnement dynamique de mémoire, présenté au chapitre 2, alloue la mémoire physique à la VM par morceau et à la demande. Ainsi, lorsqu'une VM utilise 2 Go de mémoire par exemple, la quantité de mémoire réellement allouée par l'hôte est légèrement supérieure et dépend de la taille des morceaux de mémoire utilisée pour l'allocation dynamique. Lorsque l'auto-ballonnement est activé, la mémoire réellement allouée à la VM doit obligatoirement être récupérée depuis l'hyperviseur qui est le seul à en connaître la quantité exacte. L'outil *virsh* permet d'effectuer la requête directement auprès de l'hyperviseur via le paramètre *dommemstat*, la valeur à relever apparaît sous le champ *RSS* (*Resident Set Size*).

Deuxièmement, la représentation de l'activité mémoire de la VM nécessite 3 différents paramètres qui peuvent être récupérés depuis la nouvelle fonctionnalité que nous avons introduite dans KVM et décrite au chapitre 4. Notre modélisation de l'activité mémoire d'une VM repose ainsi sur la mesure des pages chaudes et froides à fréquence d'écriture différentes. Les deux premiers paramètres à fournir concernent les pages chaudes de la VM et représentent respectivement la quantité moyenne de pages chaudes et la durée moyenne nécessaire à leur réécriture. Ces paramètres permettent principalement de déterminer la durée d'interruption de la VM. Enfin, le dernier paramètre correspond au taux moyen de modification des pages froides. En effet, les pages froides sont modifiées à un rythme plus soutenu après l'écriture des pages chaudes et nous considérons que leur taux de réécriture est constant jusqu'à la fin de la migration. Ainsi, notre modèle de migration requiert un unique paramètre qui est la fréquence de modification moyenne des pages froides. Néanmoins, ces informations relatives à l'activité mémoire de la VM ne sont pas obligatoires. Si elles ne sont pas spécifiées, le modèle considère alors que les VMs sont migrées avec une activité mémoire très faible. Nous avons ainsi défini des valeurs par défaut pour ces paramètres depuis la moyenne de plusieurs mesures effectuées sur différentes VMs inutilisées.

Le modèle de migration implémenté établit le lien entre la durée d'une migration, représentée par la longueur de la tâche dans la contrainte cumulative, et la bande passante à allouer, représentée par la hauteur de la tâche. Ainsi, BtrPlace connaît la bande passante minimale requise pour assurer la terminaison d'une migration et sait qu'une bande passante élevée réduit de

façon exponentielle la durée de migration, ce qui lui permet de prendre des décisions d'ordonnancement efficaces.

5.2.3 Configuration et application de l'ordonnancement

Le modèle d'ordonnancement de migrations implémenté dans BtrPlace requiert trois types d'informations en entrée : la configuration du centre de données, les caractéristiques des VMs et les contraintes d'ordonnancement. La configuration du centre de données concerne essentiellement le réseau et inclut l'architecture et la capacité des commutateurs réseaux. Les caractéristiques des VMs incluent leur placement initial, leur utilisation en ressources, mais aussi leur utilisation mémoire réelle et les taux de modifications des pages mémoire chaudes et froides. Les contraintes indiquent les différents besoins et attentes de l'opérateur et doivent obligatoirement être satisfaites par l'ordonnancement calculé. Les nouvelles contraintes d'ordonnancement que nous avons implémentées sont décrites en section 5.4. Ces dernières permettent d'exprimer des restrictions supplémentaires relatives à l'ordonnancement tel que le besoin de synchroniser le début ou la terminaison d'un ensemble de migrations par exemple. À l'aide de ces entrées, BtrPlace calcule finalement un *plan de reconfiguration* qui contient l'ordonnancement des actions à effectuer. Pour chaque action de migration, BtrPlace indique alors le moment de démarrage de l'action, sa durée estimée et la quantité de bande passante à lui allouer.

Un module nommé *Executor* est finalement chargé d'appliquer le plan de reconfiguration en effectuant l'ensemble des actions requises. Le module en question est entièrement personnalisable, il permet par défaut d'exécuter un script spécifique pour chaque action à réaliser en passant les paramètres de l'action (bande passante, durée estimée, etc.) en argument. De plus, pour éviter l'accumulation d'erreurs de prédiction, l'ordonnancement des actions ne doit pas uniquement se baser sur les temps de démarrage. En effet, les décalages dus à l'accumulation d'erreurs, même minimes, peuvent conduire à des violations de SLA, des consommations énergétiques excessives ou à des limitations techniques tels que la migration d'une VM vers un serveur qui n'est pas encore démarré par exemple. Ainsi, pour les besoins d'ordonnancement de migrations, le module déduit les relations de précédence et les besoins de synchronisation entre les migrations en fonction de l'horloge virtuelle globale du plan de reconfiguration généré par BtrPlace.

5.3 Optimisations

Nous avons défini deux stratégies différentes, basées sur notre nouveau modèle d'ordonnancement, pour optimiser le processus de résolution de BtrPlace. Notre première stratégie simplifie grandement le problème d'ordonnancement et nous permet de pré-calculer les durées de migration de manière statique. La seconde est une heuristique spécifique qui guide le solveur vers un plan de reconfiguration rapide.

5.3.1 Bande passante maximale

Comme expliqué dans le chapitre 4, il y a un intérêt limité à paralléliser les migrations jusqu'au partage de la bande passante maximale disponible sur leur chemin de migration. En effet, en réduisant la bande passante maximale disponible pour une migration, on augmente par effet de bord la quantité de données à transférer sur le réseau ce qui se traduit nécessairement par une perte de performance de la migration. Par conséquent, nous avons défini une première stratégie d'optimisation nommée *MaxBandwidth* qui consiste à forcer l'allocation de la bande passante maximale disponible par migration. Ainsi, la bande passante à allouer pour chaque migration peut être directement déduite du modèle réseau par la simple connaissance du chemin de migration. Dès lors, avec une seule bande passante disponible par migration, cette simplification nous permet également de pré-calculer la durée des migrations. *MaxBandwidth* réduit ainsi la quantité de variables du problème aux seules variables représentant le moment de démarrage des migrations. Par conséquent, en précalculant la bande passante et la durée de chaque migration, nous réduisons alors la complexité du problème à un RCPSP classique où chaque activité ne dispose que d'un seul et unique mode tel qu'expliqué au chapitre 4.

La seule limitation de l'optimisation *MaxBandwidth* concerne les possibilités de parallélisation des migrations. En effet, bien que la réduction de bande passante réduit inévitablement la performance des migrations, certaines situations spécifiques tel que la migration de VMs fortement communicantes nécessitent une parallélisation massive des migrations pour éviter d'autres effets de bord plus importants. Nous avons ainsi implémenté des contraintes spécifiques, décrites en section 5.4, qui nous permettent d'adresser ce problème en forçant la parallélisation des migrations jusqu'à la limite de leur faisabilité.

5.3.2 Heuristique

Notre seconde stratégie est une heuristique qui indique au solveur les variables qu'il doit instancier en priorité ainsi que les valeurs à essayer pour ces variables. De manière générale, l'intuition est de guider le solveur vers l'instanciation des variables d'intérêt. L'heuristique que nous avions implémentée dans un précédent travail de recherche [KMH15c] était trop spécialisée et seulement efficace pour résoudre des scénarios de décommissionnement. Cette première heuristique empêchait le solveur de résoudre des problèmes d'ordonnancement dans lesquels les migrations étaient moins ordonnées, sujets à dépendances ou lorsque certains hôtes doivent émettre et recevoir des VMs en même temps.

La nouvelle heuristique que nous avons implémentée établit trois groupes ordonnés de variables représentant le début des actions à réaliser : le démarrage et l'extinction d'hôtes, et le démarrage des migrations. L'heuristique demande ensuite au solveur d'instancier ces variables groupe par groupe. Pour les deux groupes de variables relatives aux hôtes, l'heuristique demande au solveur de se concentrer sur les actions les plus restrictives à planifier, c'est-à-dire celles ayant le plus petit domaine. En effet, plus le domaine d'une variable est petit et moins le solveur essaiera de valeurs différentes pour cette variable. Dès lors, les variables les plus contraintes sont instanciées en priorité pour laisser le solveur se concentrer sur les actions offrant un plus large choix d'ordonnancement différents et donc plus d'opportunités de solutions viables. Pour le groupe de variables relatives aux migrations, l'heuristique considère les migrations dans un graphe où les hôtes correspondent aux sommets et les migrations aux arcs. L'heuristique oblige d'abord le solveur à se concentrer sur les migrations dont l'hôte de destination est uniquement sujet à des migrations entrantes. Ensuite, les migrations dont les hôtes de destination ont la quantité la plus faible de migrations sortantes sont sélectionnés. Ce processus est répété jusqu'à ce que tous les instants de démarrage des migrations soient ordonnés. Cette méthode permet de réduire les risques de *retour sur trace* (*backtracking*) du solveur en limitant les problèmes de dépendances entre les migrations. En effet, en commençant par instancier les migrations entrantes d'un hôte qui n'ont pas ou peu de chances d'être dépendantes d'autres migrations sortantes sur ce même hôte, on réduit ainsi les risques de décisions entraînant un *backtracking* du solveur. De plus, à chaque fois qu'un moment de démarrage est sélectionné par l'heuristique de recherche, le solveur est forcé d'essayer en premier la valeur la plus faible pour démarrer les actions le plus tôt possible. Dès lors, la ou les premières solutions trouvées par le solveur bénéficieront d'une meilleure qualité par rapport à l'objectif qui vise à obtenir des ordonnancements rapides. Le passage à l'échelle de notre solution incluant l'heuristique et l'optimisation *MaxBandwidth* est évaluée au chapitre 6.

Il est important de rappeler que l'heuristique est uniquement un guide, elle ne change pas la définition du problème et continue de conduire le solveur vers une solution viable. En effet, le solveur prévient toute instanciation qui contredit une contrainte et le mécanisme de *backtracking* permet de revenir sur une instanciation initiale qui s'avère être invalide plus loin dans l'arbre de recherche.

5.4 Extensions

Dans cette section, nous présentons les extensions développées au sein de BtrPlace pour contrôler l'ordonnancement des migrations. Toutes ces extensions utilisent l'API de BtrPlace et reposent sur des variables fournies par le modèle d'ordonnancement.

En s'appuyant sur les variables de notre modèle de migration actuel, nous avons d'abord implémenté l'algorithme de post-copie pour pouvoir en estimer la durée des migrations. Nous avons également développé trois contraintes temporelles distinctes permettant de contrôler l'ordre d'exécution et la parallélisation des migrations. Finalement, en s'appuyant sur notre modèle de migration, nous avons implémenté un modèle énergétique permettant d'estimer la consommation d'une migration ainsi qu'une contrainte de puissance instantanée et un objectif de réduction énergétique.

5.4.1 Algorithme de post-copie

L'algorithme de post-copie, présenté au chapitre 2, est une approche très différente de la pré-copie mais n'est pas encore disponible dans les hyperviseurs de production actuels malgré son efficacité. Néanmoins les possibilités d'extensibilité offertes par BtrPlace nous permettent de supporter ce modèle de migration à la place de celui présenté an chapitre 4.

La post-copie consiste à réveiller la VM sur son hôte de destination en début de migration, sans aucune donnée mémoire, puis de récupérer les pages mémoire manquantes par la suite à la demande. Bien que ce processus dégrade les performances de la VM, il garantit que chaque page mémoire n'est transmise qu'une seule fois vers l'hôte de destination. Ainsi, en comparaison à l'algorithme de pré-copie, l'algorithme de post-copie tend à réduire la durée de migration en supprimant le besoin de retransmission des pages mémoire déjà envoyées. En s'appuyant sur les variables existantes du modèle de migration de BtrPlace, nous avons alors implémenté une version basique de cet

algorithme en statuant :

$$d(m) = \frac{mu(m)}{bw(m)} + D \tag{5.1}$$

Nous sommes ainsi capables d'estimer la durée de migration à chaud d'une VM réalisée via l'algorithme de post-copie. Cette simple version du modèle de post-copie suppose cependant que les pages mémoire envoyées en fin de migration sont transmises sans interruption et à la vitesse maximale du réseau. Cette hypothèse peut être mise à mal en fonction de l'utilisation faite des différentes techniques de récupération des pages mémoire par l'implémentation du protocole (techniques décrites au chapitre 2) mais elle reste réaliste.

5.4.2 Contrôle temporel

Au dessus du RP cœur de BtrPlace, nous avons défini des contraintes additionnelles permettant de contrôler l'ordonnancement d'un point de vue temporel. Ces contraintes peuvent être fournies au travers de scripts de configuration ou directement via l'API de BtrPlace.

La première contrainte définie consiste à synchroniser le début ou la fin d'un ensemble de migrations pour forcer leur parallélisation par exemple. Les deux autres contraintes consistent à assurer la séquentialisation d'un ensemble de migrations et à établir des relations de précédences strictes entre elles.

5.4.2.1 Parallélisation et synchronisation

Pour pouvoir agir sur la parallélisation des migrations, nous avons défini une première contrainte nommée *Sync*. Cette dernière prend en paramètre un ensemble de VMs à migrer et consiste à forcer la synchronisation de leurs migrations.

Notre implémentation de cette contrainte supporte deux approches différentes qui consistent à synchroniser soit le moment de démarrage des migrations, soit leur terminaison. En pratique, *Sync* force les variables dénotant les moments de démarrage ou de terminaison des migrations à être égales. La synchronisation des temps de démarrage peut être alors utilisée pour définir manuellement des groupes de migrations à paralléliser au lieu de laisser le solveur en décider. En effet, par défaut BtrPlace tend à grouper les migrations par durées pour minimiser la durée de reconfiguration. *Sync* permet ainsi de contrôler ces groupes en fonction de ses propres besoins.

En plus de contrôler les groupes de migrations, *Sync* permet aussi de synchroniser le réveil des VMs sur leur hôte de destination respectif. Cette fonctionnalité est inspirée de COMMA [Zhe+14], présenté au chapitre 3, où

plusieurs VMs fortement inter-communicantes doivent être migrées vers un hôte distant. La synchronisation de leur terminaison est utilisée pour limiter la perte de performance induite par le passage de trafic inter-VMs au travers d'un lien faible. Cependant, COMMA considère uniquement l'algorithme de pré-copie. En effet, la synchronisation de la terminaison des migrations n'est efficace qu'avec l'algorithme de pré-copie qui réveille les VMs sur leur hôte de destination en fin de migration. L'algorithme de post-copie quant à lui réveille les VMs sur leur hôte de destination en début de migration. Pour synchroniser leur réveil, il est alors nécessaire de synchroniser leur temps de démarrage. Ainsi, *Sync* permet de gérer naturellement les migrations de VMs inter-communicantes en utilisant l'algorithme de pré-copie ou de post-copie. De plus, notre implémentation de la contrainte *Sync* permet également de synchroniser le réveil de VMs migrées via des algorithmes différents. Par exemple, pour synchroniser une migration m_1 utilisant l'algorithme de pré-copie avec une migration m_2 qui utilise l'algorithme de post-copie, la contrainte force alors l'égalité suivante :

$$ed(m_1) = st(m_2)$$

Dans le cas particulier de COMMA, les migrations doivent être parallélisées massivement car le gain de performance obtenu par leur synchronisation vient compenser la perte de performance provoquée par leur sur-parallélisation. Cependant, nous avons vu que notre optimisation MaxBandwidth restreint les possibilités de parallélisme en forçant l'instanciation de la bande passante maximale disponible par migration. Ainsi, la contrainte *Sync* nous permet également de forcer au besoin la parallélisation massive d'un ensemble de migrations malgré les limitations du modèle d'ordonnancement.

5.4.2.2 Séquentialisation et précédence

A l'opposé de la contrainte *Sync*, utilisée pour paralléliser un ensemble de migrations, nous avons ensuite défini la contrainte *Seq* pour forcer la séquentialisation des migrations d'un ensemble de VMs passé en paramètre. *Seq* assure alors que toutes les migrations concernées soient ordonnancées de façon séquentielle. Cette contrainte permet par exemple à un administrateur de minimiser les conséquences en cas de défaillance matérielle lors de l'exécution du plan d'ordonnancement en s'assurant qu'une seule migration est active à la fois. De plus, combinée à la contrainte *Sync*, il est possible de limiter la taille de plusieurs groupes de n migrations parallèles en forçant leur séquentialisation pour s'assurer qu'à tout instant seulement n migrations soient actives. *Seq* est implémentée à l'aide d'une contrainte cumulative dont la capacité est

fixée à 1 tout comme la hauteur des migrations concernées. Bien que l'utilisation d'une contrainte *disjonctive* [Car82] serait préférable, cette dernière n'est pas encore implémentée dans Choco.

La contrainte *Seq* ne force cependant pas l'ordre des migrations et laisse le solveur décider de l'ordre le plus profitable en fonction des autres contraintes définies. Nous avons ainsi développé la contrainte *Before* qui permet soit d'établir une règle de précédence stricte entre deux migrations, soit d'imposer un temps de fin maximal pour une migration. Elle offre par exemple la capacité à un administrateur de spécifier des priorités dans une opération de maintenance, ou encore d'en assurer la terminaison avant les heures de travail par exemple pour éviter de perturber les utilisateurs du centre de données. Pour établir une règle de précédence entre deux migrations m_1 et m_2 (m_1 doit être migrée avant m_2), la contrainte force la relation suivante :

$$ed(m_1) \leq st(m_2)$$

La définition d'un temps de fin maximal pour une migration est implémentée de façon similaire. Nous avons toutefois considéré deux types de temps différents. Le premier correspond à une durée en secondes relative au début de la reconfiguration, et le second correspond à un temps absolu représentant un moment précis de la journée (représentation numérique de l'heure selon la norme internationale ISO 8601 [2]).

5.4.3 Modèle énergétique

À l'intérieur du RP cœur de BtrPlace, nous avons implanté un modèle permettant d'estimer la consommation énergétique d'une migration. BtrPlace embarque déjà un modèle énergétique pour les actions d'allumage et d'arrêt des hôtes et des VMs. Nous décrivons ici le modèle énergétique utilisé pour la migration à chaud de VMs et dérivé des travaux de Liu et al. [Liu+11]. En effet, les auteurs proposent et valident un modèle où l'énergie consommée par une migration augmente linéairement avec la quantité de données à transmettre. L'équation 5.2 exprime, à l'aide des variables de notre modèle de migration, la consommation énergétique d'une migration entre deux hôtes considérés homogènes :

$$\forall m \in \mathcal{M}, \quad E(m) = \alpha \times bw(m) \times d(m) + \beta \tag{5.2}$$

La consommation énergétique d'une migration est ainsi directement liée à la quantité totale de données transmises sur le réseau ($bw(m) \times d(m)$). Les

2. http://www.iso.org/iso/fr/home/standards/iso8601.htm

variables α et β sont les deux paramètres à déduire depuis des mesures énergétiques réelles à effectuer lors de migrations. Ces variables sont en effet dépendantes de la consommation énergétique des hôtes du centre de données et nécessitent ainsi une *phase d'entraînement* pour être déterminées de façon représentative.

Le plan de reconfiguration calculé par BtrPlace est composé d'un ensemble d'actions à exécuter. Dans une opération de maintenance d'hôtes par exemple, des VMs doivent être migrées mais des hôtes doivent aussi être éteints ou allumés. L'ensemble de ces actions doivent alors être considérées pour calculer un ordonnancement qui minimise la consommation énergétique du centre de données par exemple ou encore pour éviter de dépasser un certain budget énergétique [WW11]. Nous avons ainsi implémenté un objectif spécifique qui minimise la consommation énergétique d'un centre de données pendant la période de reconfiguration ainsi qu'une contrainte permettant de limiter la puissance instantanée consommée.

La nouvelle contrainte nommée *PowerBudget* est chargée de limiter la puissance instantanée consommée par l'infrastructure et prend en paramètres la période de temps concernée ainsi que le seuil de puissance désiré. Lorsque la période de temps n'est pas spécifiée, la limitation de puissance est appliquée pendant toute la durée de reconfiguration. Cette contrainte permet par exemple d'éviter les surchauffes [WW11], de ne pas consommer plus que la quantité d'énergie renouvelable disponible ou encore de respecter les restrictions énergétiques imposées par une *Smart City authority*. Pour respecter la contrainte de *PowerBudget*, BtrPlace est alors capable de reporter des migrations ainsi que n'importe quelles actions en fonction de leurs consommations énergétique. Tout comme pour le modèle réseau ou la contrainte *Seq*, *PowerBudget* est implémentée à l'aide d'une contrainte cumulative. La capacité de la ressource représente alors la puissance maximale autorisée pendant la période de temps désirée. Chaque action est modélisée comme une tâche dont la hauteur correspond à sa consommation de puissance instantanée. Lorsque le budget varie pendant la période de reconfiguration, plusieurs *PowerBudget* peuvent être définis. Des tâches additionnelles sont alors insérées sur les périodes de temps requises pour s'aligner sur les exigences de budget. De plus, tout comme la contrainte *Before*, les périodes désirées peuvent être spécifiées en temps absolu ou relatif au début de la reconfiguration.

L'objectif *MinEnergy* que nous avons défini consiste à minimiser l'énergie totale consommée pendant toute la durée de reconfiguration. La variable de coût à minimiser est définie comme la somme de la puissance instantanée consommée par toutes actions ($a \in \mathcal{A}$), serveurs ($s \in \mathcal{S}$) et VMs ($v \in \mathcal{V}$) à chaque seconde de la reconfiguration. En effet, l'énergie consommée E (représentée en joules (J)) par une action correspond à la somme de la puissance

instantanée consommée P (représentée en watt (W)) par l'action à chaque seconde de sa durée :

$$\forall a \in \mathcal{A}, \quad E(a) = \sum_{t \in [st(a), ed(a)]} P(a, t) \tag{5.3}$$

L'objectif est alors implémenté de la façon suivante :

$$\min \left(\sum_{a \in \mathcal{A}} E(a) + \sum_{s \in \mathcal{S}} E(s) + \sum_{v \in \mathcal{V}} E(v) \right) \tag{5.4}$$

5.5 Conclusion

Nous avons implémenté notre modèle d'ordonnancement au sein du gestionnaire de VMs BtrPlace qui se sert de la programmation par contraintes pour calculer des plans de reconfiguration de centres de données. L'implémentation des différents modèles sous-jacents nous a fait découvrir une optimisation majeure liée à l'allocation de bande passante et nous a permis de précalculer la durée des migrations à chaud en ramenant ainsi la complexité à un *problème de gestion de projet à contraintes de ressources* classique. Nous avons également développé une heuristique que nous avons ensuite améliorée pour guider rapidement le solveur vers des solutions de qualité. En s'appuyant sur le nouveau modèle d'ordonnancement, nous avons défini de nouvelles contraintes permettant de contrôler finement l'ordonnancement des migrations et nous avons également modélisé l'algorithme de post-copie pour en estimer les durées de migration. Les capacités d'extensibilité de BtrPlace nous ont également permis d'intégrer un modèle énergétique permettant d'estimer la consommation des migrations, une contrainte de limitation de la puissance instantanée ainsi qu'un objectif permettant de minimiser la consommation énergétique globale du centre de données pendant la période de reconfiguration.

L'implémentation dans son ensemble est succincte, chaque contrainte représente approximativement une centaine de lignes de code Java, et les objectifs définis nécessitent en moyenne 200 lignes de code chacun. Au total, l'implémentation représente environ 1600 lignes de code Java.

Chapitre 6

Évaluation

Sommaire

6.1 Configuration expérimentale 80
6.2 Décisions d'ordonnancement 81
6.2.1 Parallélisation optimale 81
6.2.2 Efficacité du modèle d'ordonnancement 84
6.3 Précision du modèle de migration 89
6.3.1 Modèles de comparaison 89
6.3.2 Analyse des erreurs de prédiction 90
6.3.3 Analyse de la déviation des migrations 92
6.4 Efficacité énergétique 93
6.4.1 Économie d'énergie 94
6.4.2 Évaluation de la limitation de puissance 96
6.5 Passage à l'échelle de l'ordonnanceur 97
6.5.1 Configuration expérimentale 97
6.5.2 Optimisation 98
6.5.3 Surcoût du nouveau modèle d'ordonnancement 101
6.6 Conclusion 103

Dans ce chapitre, nous évaluons l'intérêt pratique de notre modèle d'ordonnancement au travers d'expérimentations réalisées en environnements réels et simulés. Nous commençons par analyser la précision de notre modèle de migration et nous comparons les résultats obtenus avec d'autres modèles de l'état de l'art. Nous évaluons ensuite la qualité des ordonnancements calculés par BtrPlace au regard de l'infrastructure réseau et de son estimation des durées de migration. Par la suite, nous validons notre modèle énergétique et la capacité de BtrPlace à limiter la consommation de puissance pendant la phase de reconfiguration d'un centre de données. Nous évaluons également sa capacité à adapter l'ordonnancement pour satisfaire des contraintes énergétiques. Finalement, nous évaluons son passage à l'échelle et sa robustesse en calculant des plans d'ordonnancement de tailles variables à partir de scénarios générés aléatoirement.

6.1 Configuration expérimentale

Toutes les expérimentations réelles ont été réalisées sur la plateforme Grid'5000 [Bol+]. L'environnement d'exécution est composé de différents sites distribués géographiquement et disposant de plusieurs baies d'hôtes hétérogènes. Notre configuration du réseau et de l'environnement d'exécution décrit ci-dessous est identique sur l'ensemble des sites utilisés, seule la puissance des hôtes varie. Les hôtes d'une même baie sont raccordés à leur commutateur *ToR* via des interfaces Ethernet à 1 Gbps. Les commutateurs ToR sont ensuite connectés ensemble sur un même commutateur d'agrégation via des interfaces Ethernet à 10 Gbps. Les hôtes sont également connectés à un réseau Infiniband à 20 Gbps que nous utilisons pour partager les images disques des VMs entre les hôtes via un serveur NFS dédié. Pour éviter toute interférence avec le trafic issu des VMs, le réseau Ethernet des hôtes est exclusivement dédié aux migrations.

Le système d'exploitation installé sur les hôtes repose sur la distribution Linux *Debian Jessie* avec un noyau GNU/Linux 3.16.0-4-amd64. Au niveau de la virtualisation, nous utilisons l'hyperviseur KVM (Qemu) dans sa version 2.2.50. Cette version de KVM a été compilée par nos soins et inclut notre patch, décrit au chapitre 5, pour analyser l'activité mémoire des VMs. La bibliothèque de virtualisation *libvirt* est utilisée pour configurer les VMs et réaliser les migrations. Afin d'obtenir des durées de migration minimales et une meilleure réactivité de l'environnement, nous avons désactivé le chiffrement des communications de *libvirt*. La quantité de connexions parallèles autorisées par *libvirt* a également été augmentée au maximum pour éviter de restreindre la parallélisation des migrations. Le système d'exploitation des VMs repose sur la distribution Linux *Ubuntu Utopic Unicorn (14.10)* en version bureau que nous avons configuré pour représenter un environnement réel et représentatif d'une utilisation quotidienne. Un unique CPU virtuel est paramétré par VM et la durée d'interruption maximale des migrations est configurée par défaut à 30 ms. L'activité mémoire des VMs est simulée à l'aide de l'outil en ligne de commande *stress* que nous configurons pour reproduire l'activité mémoire observé sur des applications courantes. Finalement, nous utilisons la commande *tc* pour émuler des réseaux bloquants en limitant le débit des interfaces réseaux désirées.

L'ensemble des expérimentations décrites dans ce chapitre sont reproductibles, tout les éléments et informations nécessaires à leur reproduction sont disponibles dans un dépôt publique du projet GitHub officiel de BtrPlace [1].

1. `https://github.com/btrplace/kherbache-thesis`

6.2 Décisions d'ordonnancement

Dans cette section, nous évaluons la qualité des décisions d'ordonnancement réalisées par notre nouvelle version de BtrPlace en les exécutant sur un environnement réel. Nous analysons d'abord en détail le comportement de notre modèle d'ordonnancement sur un scénario de décommissionnement et nous comparons les résultats obtenus par rapport à la version originale de BtrPlace. Nous validons ensuite l'intérêt pratique de notre solution via l'exécution de scénarios générés aléatoirement et nous comparons son efficacité par rapport à l'ordonnanceur de *Memory Buddies* [Woo+09].

6.2.1 Parallélisation optimale

Pour analyser précisément les décisions de parallélisation prises par notre modèle d'ordonnancement, nous avons réalisé une micro-expérimentation représentant un scénario de décommissionnement. L'expérimentation consiste à remplacer 4 anciens hôtes, hébergeant chacun 2 VMs, par un nouveau plus puissant. La mémoire utilisée par les VMs varie de 1 à 2 Go et le nouvel hôte dispose de suffisamment de ressources pour héberger l'ensemble des 8 VMs. Pour créer des opportunités de parallélisation tout en profitant de notre optimisation *MaxBandwidth*, nous avons émulé un réseau hétérogène où l'hôte de destination dispose d'une bande passante deux fois plus élevée que les hôtes source. Pour ce faire, la commande *tc* limite la bande passante des hôtes source à 500 Mbps. Le scénario est représenté en Figure 6.1.

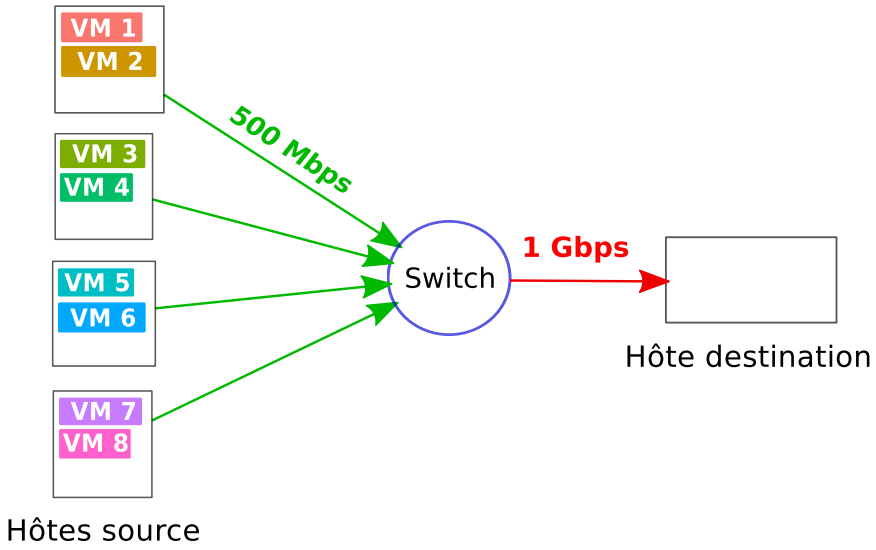


FIGURE 6.1 – Configuration expérimentale.

Pour évaluer et comparer les résultats obtenus, nous avons d'abord exé-

cuté l'ordonnancement calculé par la version originale de BtrPlace. Les durées de migration obtenues sont représentées par le diagramme de Gantt en Figure 6.2. On constate que les migrations sont toutes réalisées en parallèle et affichent une durée moyenne de 84 secondes. Ce comportement s'explique par les lacunes du modèle d'ordonnancement original de BtrPlace qui ne tient pas compte des caractéristiques du réseau et se contente de migrer massivement les VMs si aucune dépendance n'est requise. En effet, par défaut, l'ordonnanceur d'origine de BtrPlace parallélise l'ensemble des migrations. Des décisions de séquentialisation ne sont prises que lorsque des contraintes spécifiques le lui obligent tels que par exemple des contraintes sur la disponibilité des ressources nécessaires à l'hébergement des VMs.

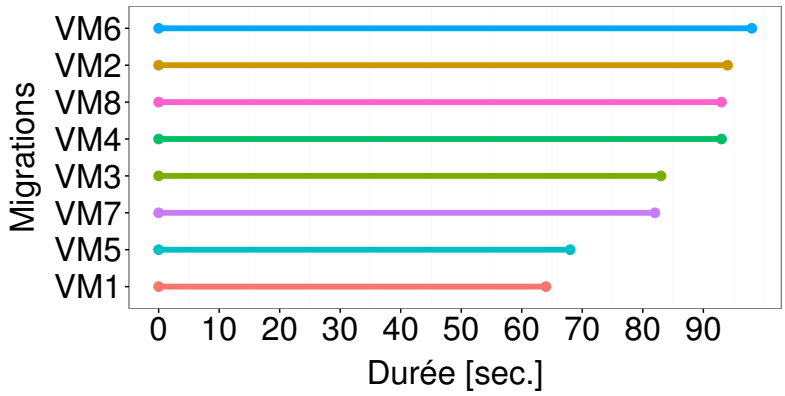


FIGURE 6.2 – Ordonnancement utilisant l'ancien modèle.

L'exécution de l'ordonnancement calculé par notre nouvelle version de BtrPlace est représentée en Figure 6.3. On constate d'abord que les migrations sont toutes parallélisées 2 par 2. En se référant ensuite au placement initial des VMs représenté en Figure 6.1, on remarque que les migrations parallèles sont sélectionnées depuis des hôtes source différents. Cet ordonnancement alloue ainsi une bande passante maximale de 500 Mbps à chaque migration en utilisant le lien de destination de 1 Gbps à sa capacité maximale. On remarque ensuite que les migrations sont groupées par durée. En effet, les prédictions réalisées par notre modèle de migration ont permis à BtrPlace de grouper les migrations ayant des durées similaires de façon à conserver une parallélisation optimale le plus longtemps possible. Ainsi, l'utilisation du lien réseau de destination (1 Gbps) est maximisée pendant toute la durée de reconfiguration ce qui garantit un temps de complétion minimal. De manière générale, la précision moyenne du modèle de migration est de 93,86% avec une déviation de seulement 1,5 secondes. En comparaison à l'ordonnancement calculé par la version originale de BtrPlace, les durées de migration sont en moyenne 3,4 fois plus faibles avec une durée de migration moyenne de 24,7 secondes.

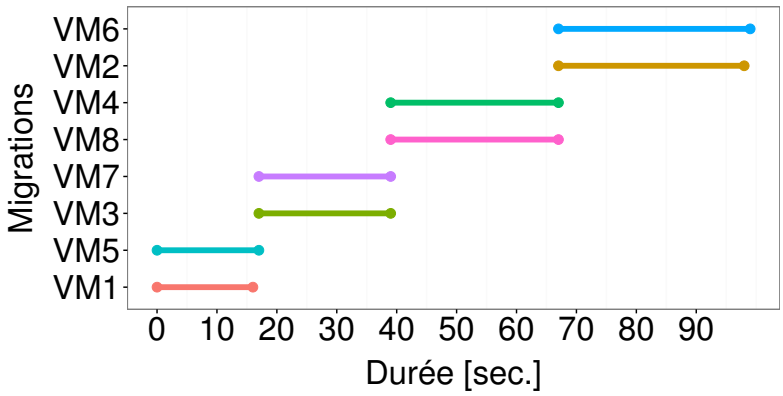


FIGURE 6.3 – Ordonnancement utilisant le nouveau modèle.

Malgré les meilleures performances de notre modèle d'ordonnancement, le temps de complétion reste le même que pour l'ordonnancement parallèle généré par la version originale de BtrPlace. Le gain en temps de complétion n'est pas visible sur cette expérimentation pour plusieurs raisons. Premièrement, nous avons été contraint de minimiser l'activité mémoire des VMs pour pouvoir terminer l'ensemble des migrations avec la version originale de BtrPlace. En effet, lors de la parallélisation massive des migrations représenté en Figure 6.2, la bande passante maximale allouable à chaque migration est de 125 Mbps. Comme expliqué au chapitre 2, lorsque l'activité mémoire d'une VM est trop élevée par rapport à la bande passante disponible pour sa migration, on assiste alors à une retransmission cyclique des pages mémoire modifiées qui rend la migration interminable. Nous avons ainsi été contraint de réduire l'activité mémoire des VMs pour permettre la terminaison des migrations avec seulement 125 Mbps de bande passante. Dès lors, le temps passé à retransmettre les pages mémoire modifiées est très faible et le temps complétion obtenu se rapproche de l'ordonnancement optimal généré par notre nouveau modèle. Cependant, le temps de complétion devrait être légèrement plus élevé que pour l'ordonnancement réalisé par notre nouveau modèle ce qui soulève un second problème. En effet, pour chaque action de migration, le temps passé à contacter les deux hyperviseurs, initialiser la migration et enfin attendre qu'elle atteigne sa vitesse de transfert maximale introduit une légère latence. Bien que cette latence soit faible (moins de 2 secondes), elle se cumule lorsque les migrations sont séquentialisées et peut devenir significative sur des scénarios de courtes durées. Sur des scénarios à plus grande échelle, tel que celui réalisé en section 6.4, le temps de complétion obtenu avec notre nouveau modèle est plus court que pour la version originale de BtrPlace.

6.2.2 Efficacité du modèle d'ordonnancement

Pour évaluer l'efficacité des décisions d'ordonnancement prises par notre nouvelle version de BtrPlace, nous réalisons une nouvelle expérimentation qui consiste à migrer 10 VMs, ayant différentes utilisations mémoire, entre 4 hôtes d'une même baie. Chaque hôte possède 2 processeurs Intel Xeon L5420 de quatre cœurs chacun et 16 Go de mémoire vive. La description du réseau et la configuration de l'environnement sont décrits en section 6.1. Pour émuler un réseau bloquant, la commande *tc* limite la bande passante de la moitié des serveurs à 500 Mbps. La mémoire utilisée par les VMs est fixée à 2 Go pour la première moitié des VMs et à 3 Go pour la seconde moitié. Cette quantité représente la mémoire réellement allouée à la VM par l'auto-ballonnement dynamique de KVM et représente ainsi la quantité exacte de données à transférer lors de la première étape de l'algorithme de pré-copie. L'activité mémoire de chaque VM est générée via l'exécution de 1000 tâches qui allouent puis libèrent chacune 70 Ko de mémoire de façon continue. Le placement des VMs est réalisé de manière aléatoire comme décrit ci-dessous.

Dans cette première expérimentation, nous comparons les ordonnancements calculés par notre nouvelle version de BtrPlace à un ordonnanceur qui reproduit les décisions de *Memory Buddies* (MB). Étant donné que la version originale de BtrPlace ne génère que des plans de migration entièrement parallélisés qui peuvent empêcher les migrations de se terminer, nous avons choisi d'écarter la version originale de BtrPlace de cette expérimentation. De façon similaire à BtrPlace, MB contrôle la parallélisation des migration. Toutefois, le niveau de parallélisme est statique et doit être défini à l'avance. En pratique, nous le configurons de trois manières différentes, dénommées *MB-2*, *MB-3* et *MB-4*, où le niveau de parallélisme varie respectivement de 2 à 4. Ces configurations sont suffisantes pour exploiter la pleine capacité des liens en duplex intégral tout en limitant les risques de saturation. Pour réaliser une expérimentation robuste qui couvre un large spectre de scénarios différents, nous avons pré-calculé 50 différentes exécutions où les hôtes source et destination de chaque VM sont choisis aléatoirement. Cela nous permet d'éviter tout biais lié à une configuration particulière de l'expérimentation. Pour évaluer la qualité des résultats, nous avons comparé pour chaque ordonnanceur les durées de migration et les temps de complétion obtenus par rapport à un ordonnancement purement séquentiel réalisé dans un environnement virtuel considéré parfait. Cette comparaison exhibe les avantages et inconvénients des décisions de parallélisation via la représentation de l'accélération des temps de complétion et des ralentissements potentiels des migrations par rapport à un ordonnancement séquentiel. Enfin, pour chaque ordonnanceur, les valeurs représentées correspondent à la moyenne obtenue sur trois exécutions

séparées.

6.2.2.1 Durées de migration

Tableau 6.1 – Durées de migration absolues et ralentissement relatif à un ordonnancement séquentiel.

Ordonnanceur	BtrPlace	MB-2	MB-3	MB-4
Durées moyennes de migration (sec.)	45,55	57,22	113,2	168,6
Ralentissement moyen observé (%)	7,35%	29,69%	141,3%	259,2%

Le Tableau 6.1 indique les durées de migration moyennes observées pour chaque ordonnanceur ainsi que le ralentissement par rapport à des durées de migration minimales obtenues lors d'un ordonnancement séquentiel. Nous observons d'abord que la nouvelle version de BtrPlace obtient de meilleurs résultats que toutes les configurations de MB. En effet, les ordonnancements calculés par BtrPlace terminent en moyenne 20,4% plus rapidement que ceux calculés par MB-2 qui est la meilleure configuration de MB dans ce scénario. Par rapport aux durées de migration minimales, on observe en moyenne 7,35% de ralentissement pour BtrPlace et au moins 4 fois plus pour MB-2.

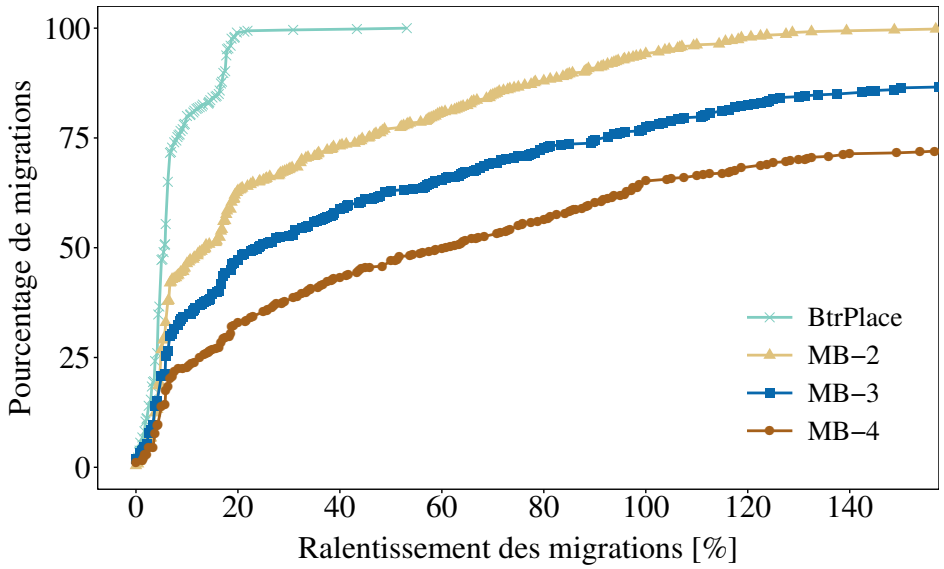


FIGURE 6.4 – Durées de migration.

Les *fonctions de distribution cumulative* (CDFs) en Figure 6.4 représentent, pour les 4 ordonnanceurs sélectionnés, la distribution du ralentissement des migrations en pourcentage par rapport à un ordonnancement séquentiel. On observe que 88% des migrations ordonnancées par BtrPlace ont un

ralentissement inférieur à 5 secondes, contre seulement 52,8% pour MB-2. On remarque également que la distribution des ralentissements pour BtrPlace est regroupée alors qu'elle est dispersée pour MB et empire avec l'augmentation du niveau de parallélisme. Ces améliorations par rapport à MB s'expliquent par de meilleures décisions de parallélisation. En effet, MB parallélise les migrations statiquement ce qui peut produire une sous-utilisation des capacités du réseau ainsi que des abus de parallélisation sur des liens où la migration concurrente de VMs n'est pas désirée. Cette sur-parallélisation réduit alors la bande passante disponible pour les migrations ce qui entraîne une augmentation des transmissions de pages mémoire modifiées et ainsi des durées de migration plus élevées. De l'autre côté, BtrPlace déduit la quantité adéquate de migrations concurrentes par lien réseau au cours du temps à partir de sa connaissance de l'architecture et des capacités du réseau. Par conséquent, BtrPlace migre chaque VM à sa vitesse maximale et parallélise les migrations en maximisant l'utilisation de la capacité du réseau. En pratique, nous avons observé une variation du parallélisme de 2 à 5 migrations concurrentes au cours de l'exécution. On observe globalement que BtrPlace prend des décisions de parallélisation plus élevées que la configuration la plus agressive de MB tout en produisant de plus faibles ralentissements que sa configuration la moins agressive.

Finalement, on observe trois migrations distinctes anormalement longues sur l'ordonnancement de BtrPlace. Une analyse de l'environnement d'exécution a révélée que ces anomalies sont causées par les limitations techniques de notre configuration expérimentale et plus spécifiquement à la configuration de la limitation de trafic sur certaines interfaces réseau. En effet, lorsqu'un hôte reçoit et envoie simultanément des migrations au travers d'une interface limitée à 500 Mbps, alors la régulation de trafic effectuée via la commande *tc* n'est pas toujours équitable. Il s'avère que le mécanisme de file d'attente employé pour la régulation de trafic engendre des irrégularités périodiques sur l'allocation de bande passante aux flux concurrents (entrant et sortant) des interfaces réseaux concernées. En effet, pour limiter le trafic d'une interface réseau dans ses deux sens de communications, plusieurs files d'attentes sont mises en place par *tc*, dont une file globale qui traite l'ensemble des flux entrants et sortants de l'interface. Nous avons constaté que ces mécanismes de filtrages superposés ne sont pas parfaits et induisent des déséquilibres et des latences dans le traitement des flux. Nous avons pu reproduire cette perturbation à l'aide de l'outil de mesure de performances réseau *iperf* et nous avons mesuré des ralentissements occasionnels variants de 100 à 200 Mbps dans le pire des cas observés. Ce problème intervient aussi lors des expérimentations réalisées avec MB car les chances d'utiliser des interfaces restreintes dans ses deux sens de communication augmente lorsque la parallélisation est plus éle-

vée. Malheureusement, à notre connaissance, aucune solution fiable idéale de limitation de trafic bidirectionnel n'existe actuellement.

6.2.2.2 Temps de complétion

Tableau 6.2 – Temps de complétion absolus et accélération relative à un ordonnancement séquentiel.

Ordonnanceur	BtrPlace	MB-2	MB-3	MB-4
Temps de complétion moyen (sec.)	212,8	295,9	394,6	479,4
Accélération moyenne (%)	54,18%	36,42%	15,94%	-2,64%

Le Tableau 6.2 indique les temps de complétion obtenus pour chaque ordonnanceur ainsi que leur accélération par rapport à un ordonnancement purement séquentiel. On observe d'abord que BtrPlace produit des temps de complétion plus courts que toutes les configurations de MB. En effet, les exécutions se sont terminées en moyenne 83,1 secondes (soit 28,1%) plus tôt qu'avec MB-2 qui reste la meilleure configuration de MB. Pour évaluer la qualité de ces résultats, tout comme pour les durées de migration, nous avons comparé les temps de complétion obtenus aux valeurs prédites par un ordonnancement séquentiel. On observe alors une accélération moyenne des temps de complétion de 54,18% pour BtrPlace, 1,49 fois plus forte que pour MB-2 qui affiche une accélération de 36,36%.

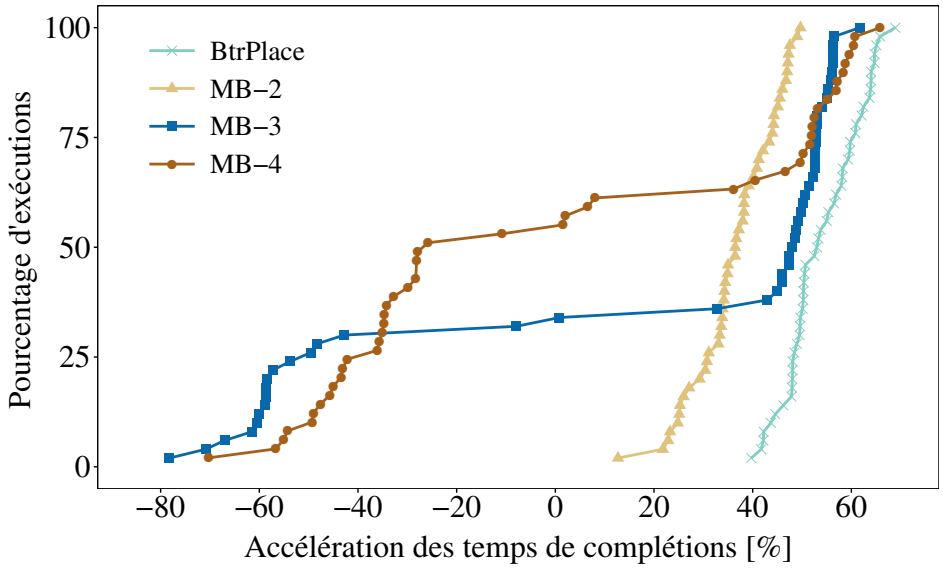


FIGURE 6.5 – Temps de complétion.

La Figure 6.5 représente les CDFs de l'accélération des temps de complétion obtenue pour BtrPlace ainsi que les 3 configurations de MB. À nouveau, BtrPlace surpasse toutes les configurations de MB avec une amélioration des temps de complétion de 54,3% en moyenne contre 36,4% pour MB-2. Cette amélioration globale est due aux nouvelles décisions de parallélisation et de groupement prises par BtrPlace. Comme expliqué précédemment, BtrPlace optimise le parallélisme en fonction des chemins de migration et des bandes passantes disponibles alors que MB restreint le parallélisme par une constante. Par ailleurs, contrairement à MB, BtrPlace déduit comment grouper les migrations en fonction de la prédiction des durées de migration. Le groupement des migrations est réalisé de manière à maximiser l'utilisation globale du réseau et ainsi minimiser le temps de complétion. De manière générale, les performances observées valident la fiabilité et l'efficacité de notre modèle d'ordonnancement implémenté dans BtrPlace. En conclusion, cette expérimentation confirme que le groupement des migrations par la prédiction des durées de migration et l'adaptation dynamique du niveaux de parallélisme est la clef pour réaliser des ordonnancements de qualité.

Du coté de MB, alors que toutes les exécutions de MB-2 sont plus courtes que l'ordonnancement séquentiel, on observe que la répartition des accélérations de MB-3 et MB-4 est très dispersée avec des temps de complétion qui sont soit très longs soit très courts. Par exemple pour MB-3, 15 des 50 exécutions affichent un temps de complétion au moins 40% plus long que pour l'ordonnancement séquentiel alors que 33 exécutions ont un temps de complétion au moins 40% plus court. Ces résultats illustrent parfaitement les avantages et inconvénients de la parallélisation des migrations. En effet, avec un niveau de parallélisation statique, certaines exécutions bénéficient d'une meilleure utilisation du réseau alors que d'autres pâtissent d'une sur-parallélisation qui diminue les bandes passantes allouées aux migrations et augmente le temps de complétion. De plus, malgré l'accélération moyenne supérieure de MB-2, plus de la moitié des exécutions de MB-3 affichent une meilleure accélération que MB-2 et profitent ainsi d'une parallélisation plus élevée. Cependant, plus la parallélisation augmente et plus cette proportion diminue car seules 8 exécutions sur 50 de MB-4 surpassent MB-3 et profitent réellement d'une parallélisation de 4 migrations à la fois.

Bien que notre approche soit précise et permette de représenter l'activité mémoire d'applications courantes [KMH14 ; Ako+10 ; Liu+11], certaines VMs peuvent toujours exhiber une activité mémoire instable. Dans ce cas, l'estimation des durées de migration peut être imprécise et amener BtrPlace à produire des ordonnancements irréalistes. En effet, ces imprécisions peuvent biaiser les décisions de groupement réalisées par BtrPlace en créant des périodes de sous-utilisation du réseau et augmenter le temps de complétion. Il

est alors important de déduire les relations de précédences entre les migrations pour appliquer le plan d'ordonnancement calculé. Cela permet de conserver des décisions de parallélisation optimales malgré les erreurs d'estimation des durées de migration. Ainsi, contrairement à MB, aucune parallélisation excessive ne peut survenir et par conséquent les durées de migration restent minimales.

6.3 Précision du modèle de migration

Pour évaluer la précision de notre modèle de migration, nous nous appuyons sur les résultats expérimentaux de la précédente section 6.2.2. Nous reprenons ainsi les plans d'ordonnancement calculés par notre nouvelle version de BtrPlace et nous les exécutons sur différents outils de simulation et modèles représentatifs existants. Nous déduisons alors la précision des différents modèles en comparant les durées de migration estimées avec les durées d'exécution réelles obtenues.

6.3.1 Modèles de comparaison

Le premier modèle de migration que nous avons choisi est représentatif de la plupart des simulateurs existants qui ignorent à la fois l'architecture interne du réseau et l'activité mémoire des VMs. Par exemple, le standard de facto CloudSim [Cal+11], mais aussi Entropy ou encore la version originale de BtrPlace ne tiennent pas compte ni de la topologie complète du réseau (seulement les interfaces réseaux des hôtes) ni de l'activité mémoire générée par les applications s'exécutant sur les VMs. Nous nous référons à cette famille de modèles sous le nom *NoShare*. Le second modèle de migration sélectionné, nommé *NoDP*, considère toute l'architecture réseau mais ignore l'activité mémoire des VMs. Ainsi, ce modèle est uniquement consacré à étudier l'importance de l'activité mémoire des VMs dans le modèle de migration en analysant la déviation par rapport à notre modèle d'ordonnancement complet. Le troisième modèle que nous avons sélectionné, nommé *SimGrid*, correspond au modèle actuellement implémenté dans le simulateur SimGrid [Cas+14]. Ce dernier repose sur une modélisation réaliste du réseau basé sur les flux de communications avec un mécanisme de file d'attente stochastique (SFQ). La latence du réseau est considérée ainsi que les effets de la concurrence de trafics et leurs impacts sur les vitesses de migration. Le modèle de migration de SimGrid considère également l'activité mémoire des VMs [HLP13]. Elle est représentée par une intensité d'écriture des pages mémoire et modélisée comme un pourcentage de la bande passante disponible tempérée par l'utilisation CPU de la VM. En

considérant un unique taux constant d'écriture des pages mémoire, nommé DP_r, la durée de migration $d(m)$ est modélisée par SimGrid via l'équation suivante :

$$d(m) = \frac{mu(m)}{bw(m) - DP_r} \tag{6.1}$$

Pour rappel, $mu(m)$ représente la mémoire utilisée par la VM et $bw(m)$ la bande passante allouée pour sa migration. L'intensité de modification des pages mémoire notée DP_i est donnée par l'équation (6.2) où $cu(m)$ est l'utilisation CPU de la VM à migrer en pourcentage :

$$DP_i = \frac{DP_r}{(cu(m) \times bw(m))} \tag{6.2}$$

La durée de migration $d(m)$ peut alors s'écrire :

$$d(m) = \frac{mu(m)}{bw(m) - (DP_i \times cu(m) \times bw(m))} \tag{6.3}$$

Pour configurer SimGrid avec des paramètres représentatifs de notre configuration réelle, l'utilisation CPU $cu(m)$ est définie à 100% pour correspondre à la charge CPU générée par l'outil *stress* utilisée pour simuler des activités mémoire diverses. Pour déterminer l'intensité de modification des pages mémoire DP_i correspondant à notre configuration de l'outil *stress*, nous avons d'abord calculé à l'aide de notre modèle de migration la quantité totale de pages modifiées pendant la plus courte et la plus longue des migrations observées (respectivement 20 et 61 secondes). Nous avons ensuite déduit les deux taux constants de modifications des pages mémoire correspondants aux quantités totales de pages calculées. Finalement, nous avons déduit l'intensité de modification des pages mémoire DP_i depuis l'équation 6.2. Nous avons obtenu des intensités variant de 5 à 7% et sélectionné la moyenne de 6% comme intensité globale de modification des pages mémoire pour l'ensemble des VMs. En effet, étant donné que la charge générée par notre configuration de l'outil *stress* est la même pour chaque VM, une même intensité à alors été définie pour l'ensemble des VMs.

6.3.2 Analyse des erreurs de prédiction

Les CDFs en Figure 6.6 montrent la distribution de la déviation normalisée des durées de migration par rapport à leur exécution réelle pour chacun des modèles utilisés. Les valeurs exactes sont également fournies dans le Tableau 6.3. On observe que *NoShare* produit les plus longues durées de

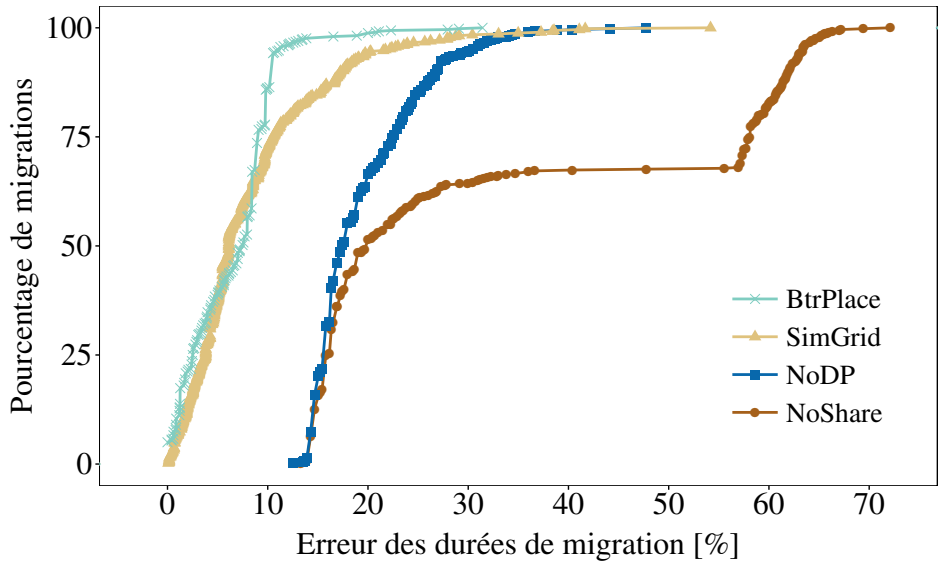


FIGURE 6.6 – Déviation normalisée des durées de migration par rapport aux exécutions réelles.

migration en atteignant 32,28% de déviation moyenne. Le CDF correspondant peut être séparé en deux parties où 68% des migrations sont prolongées par la simple ignorance de l'activité mémoire des VMs et les 32% restantes souffrent également d'une parallélisation excessive occasionnée par l'ignorance de l'architecture réseau. Dans ce dernier et pire cas, la déviation des durées de migration est comprise entre 57 et 72% ce qui indique de graves problèmes d'ordonnancement. *NoDP* engendre des déviations plus courtes avec une moyenne de 19,54% mais cette déviation reste toutefois trois fois plus élevée que pour BtrPlace. De plus, la meilleure prédiction issue du modèle *NoDP* entraîne tout de même une déviation de 10% par rapport à la durée réelle de migration observée. Cette déviation généralisée souligne ainsi l'importance de l'activité mémoire des VMs pour l'estimation des durées de migration. Le modèle de SimGrid affiche de meilleurs résultats avec une déviation moyenne de 8,31%. Bien que la déviation est similaire à celle de BtrPlace pour environ 70% des migrations, on observe une plus grande dispersion de la distribution avec un écart type 63,27% plus élevé que pour BtrPlace. Ces résultats démontrent que le modèle de migration implémenté dans BtrPlace, basé sur deux taux distincts de modification des pages mémoire (PC_r et PF_r), est plus précis que le simple taux constant DP_r considéré par SimGrid.

Tableau 6.3 – Déviation normalisée des durées de migration par rapport aux exécutions réelles.

Modèle de migration	NoShare	NoDP	SimGrid	BtrPlace
Moyenne (%)	32,28	19,54	8,31	6,47
Écart type (%)	20,17	5,34	7,2	4,41
Premier quartile (%)	16,16	15,79	3,82	2,55
Médiane (%)	20	17,42	6,12	7,49
Troisième quartile (%)	58,17	22,58	10,55	9,09

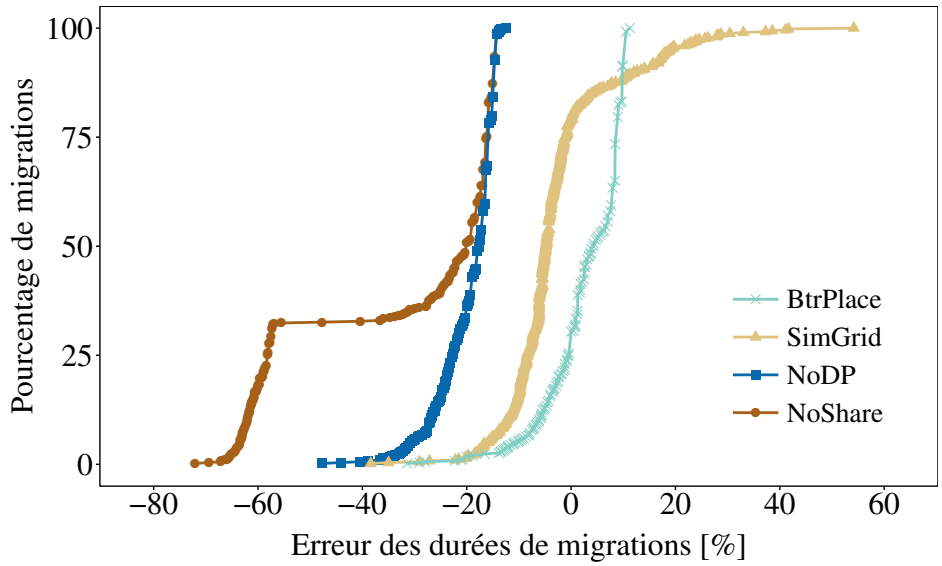


FIGURE 6.7 – Déviation absolue des durées de migration par rapport aux exécutions réelles.

6.3.3 Analyse de la déviation des migrations

Pour analyser plus en détails la déviation des durées de migration, les CDFs en Figure 6.7 montrent la distribution de la déviation absolue entre sous-estimation et sur-estimation des durées de migration. Nous remarquons d'abord que BtrPlace génère les résultats les plus équilibrés entre sous- et sur-estimation. Au niveau de SimGrid, on observe une tendance à la sous-estimation des durées de migration avec 80% des estimations sous-évaluées. Ces résultats reflètent le modèle de migration utilisé par SimGrid basé sur une représentation linéaire de l'activité mémoire des VMs et qui tend à sous-évaluer la durée de migration additionnelle induite par les différences de fréquence d'écriture entre les pages mémoire chaudes et froides des VMs. En effet, en dégradant notre modèle vers un unique taux de modification des pages mémoire identique pour chaque VM, nous réduisons indéniablement

la précision par rapport à notre modèle d'origine par l'approximation d'une valeur commune qui convienne le mieux à l'ensemble des migrations.

Par ailleurs, malgré un modèle réseau robuste, SimGrid a tendance à fortement sur-estimer certaines durées de migration en atteignant 55% de déviation dans le pire cas et seulement 10% avec BtrPlace. En pratique, ces sur-estimations se produisent dans les scénarios impliquant l'utilisation bidirectionnelle des liens de façon simultanée. Nous supposons alors que l'estimation des effets indésirables causés par la concurrence de trafics sur les deux sens de communication d'un même lien est sur-estimé dans le modèle réseau actuel de SimGrid.

On observe également 5% de sous-estimations extrêmes pour l'ensemble des modèles utilisés. Ces cas particuliers sont liés à la limitation de trafic utilisée sur certaines interfaces réseau tel que décrit en section 6.2.2.1. Ces 5% de valeurs aberrantes proviennent ainsi de notre configuration expérimentale et peuvent être ignorées pour la comparaison de la précision des modèles de migration.

6.4 Efficacité énergétique

Dans cette section, nous validons l'intérêt pratique du modèle énergétique en analysant la consommation énergétique produite par l'exécution des plans d'ordonnancement calculés par BtrPlace. Nous analysons d'abord les décisions d'ordonnancement prises par BtrPlace pour minimiser la consommation énergétique lors de la reconfiguration d'un centre de données. Dans un second temps, nous évaluons la capacité de BtrPlace à répondre à des besoins énergétiques spécifiques et nous validons la qualité des plans d'ordonnancement.

L'expérimentation consiste à exécuter un scénario de décommissionnement à grande échelle, calculée avec différents ordonnanceurs, puis d'observer la consommation énergétique générée. Contrairement à BtrPlace, l'ordonnanceur de MB n'est pas destiné à planifier d'autres actions que les migrations (comme l'allumage ou l'extinction d'hôtes par exemple). En conséquence, nous utilisons la version originale de BtrPlace comme référence pour cette expérimentation. L'environnement expérimental est composé de 3 baies de 24 hôtes chacune. Chaque hôte dispose d'un processeur Intel Xeon X3440 à quatre cœurs cadencés à 2,53 GHz et de 16 Go de mémoire vive. Le réseau et la configuration de l'environnement sont décrits en section 6.1. Le scénario de décommissionnement consiste à remplacer deux anciennes baies par une nouvelle en y migrant l'ensemble des VMs hébergées. Chaque hôte source héberge 2 VMs, l'expérimentation revient donc à migrer un total de 96 VMs depuis 48 hôtes source et à destination de 24 hôtes. Chaque VM utilise un unique

CPU virtuel et la mémoire allouée est définie à 2 Go pour la première moitié des VMs et 4 Go pour la seconde moitié. Étant donné que la version originale de BtrPlace sur-parallélise les migrations, nous avons minimisé l'activité mémoire des VMs pour éviter les risques de non-terminaison des migrations.

Pour calibrer notre modèle énergétique avec des valeurs réalistes, les consommations énergétiques des hôtes (exécution, allumage, hébergement de VMs) ont été directement mesurées depuis le *contrôleur de gestion de la carte mère* (BMC). Au niveau des migrations de VMs, nous avons réutilisé les valeurs expérimentales de références du modèle énergétique de migration [Liu+11]. Le Tableau 6.4 décrit en détail le calibrage du modèle énergétique.

Tableau 6.4 – Calibration du modèle énergétique

Élément du modèle	**Modélisation énergétique**
Exécution d'un hôte (*idle*)	$110\ W \times$ *durée d'exécution*
Démarrage d'un hôte	$20\ W \times$ *durée du démarrage*
Hébergement d'une VM	$16\% \times 110\ W \times$ *durée d'hébergement*
Migration d'une VM	$0.512 \times$ *données transférées* $+ 20.165$

6.4.1 Économie d'énergie

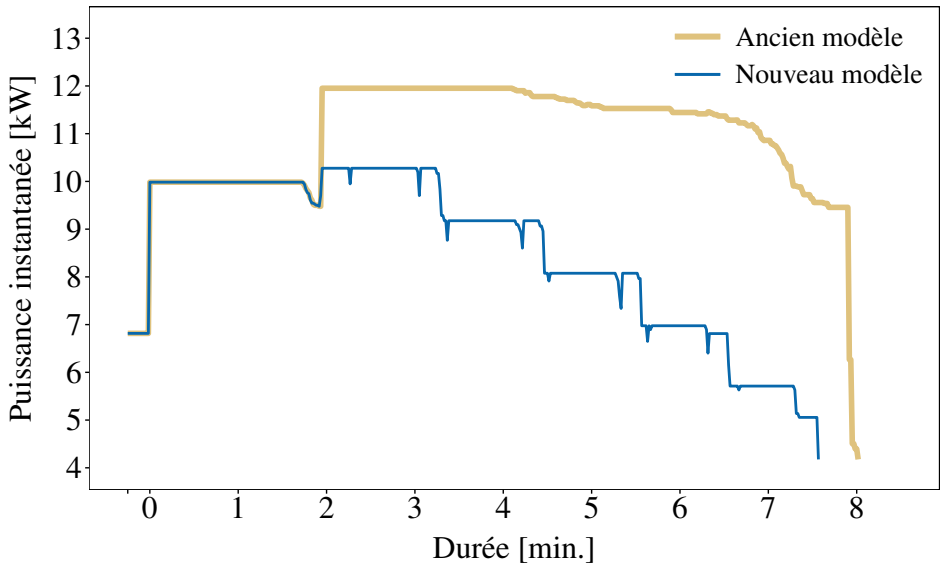


FIGURE 6.8 – Puissance instantanée consommée.

La Figure 6.8 compare la puissance instantanée consommée durant la période de reconfiguration entre l'ordonnancement calculé par l'ancien modèle

de BtrPlace et le nouveau. Étant donné que notre plateforme expérimentale ne nous permet pas de mesurer la consommation de puissance de l'ensemble des hôtes utilisés, les valeurs sont calculés depuis notre modèle énergétique. Nous avons calculé que le nouveau modèle d'ordonnancement économise au total 2350 Joules par rapport à l'ancien modèle de BtrPlace, soit une réduction de 21,5% de la consommation énergétique. Cette différence est principalement expliquée par l'extinction des hôtes sources qui est réalisée bien plus tôt par le nouveau modèle. Pour pouvoir réaliser les actions d'extinctions des hôtes source au plus tôt, le nouveau modèle implémenté dans BtrPlace ordonne les groupes de migrations de manière à pouvoir libérer les hôtes sources progressivement et les éteindre dès que possible. Au début de l'expérimentation, la consommation de puissance instantanée augmente rapidement de 7 à 10 kW pour les deux modèles d'ordonnancement. Cette augmentation est expliquée par le démarrage simultané des 24 hôtes de destination pendant environ 2 minutes. Une fois les hôtes allumés, l'ancien modèle de BtrPlace exécute toute les migrations en parallèle, ce qui entraîne de très longues durées de migration. Comme la plupart des migrations terminent en effet cascade au bout de 8 minutes d'expérimentations, il est alors impossible d'éteindre un seul hôte source avant cette période. En effet, en conséquence de la parallélisation massive des migrations, au moins une migration provenant de chaque hôte source est toujours en cours d'exécution au bout de 8 minutes d'expérimentation. Avec le nouveau modèle de migration, les migrations sont mieux parallélisées et terminent plus vite ce qui permet de libérer et d'éteindre des hôtes source dès la $3^{ème}$ minute d'expérimentation. Ce comportement est observé en Figure 6.8 par les réductions successives de la puissance consommée à partir de la $3^{ème}$ minute.

En pratique, les migrations ont été réalisées 10 par 10. Comme décrit au chapitre 4, l'optimisation *MaxBandwidth* force la bande passante maximale de 1 Gbps par migration, la parallélisation 10 par 10 des migrations utilise ainsi toute la capacité du réseau Ethernet 10 Gbps. De plus, pour obtenir un flux inter-commutateurs de 10 Gbps, les groupes de migrations sont choisis entre 10 hôtes sources et destinations différents et groupés par leur durée estimée. Avec le nouveau modèle d'ordonnancement, on observe également de courtes baisses de consommation de puissance représentés par les pics descendants sur la Figure 6.8. Ces pics correspondent aux instants entre la terminaison d'un groupe de migrations et au commencement du suivant. En théorie, les groupes de migrations se succèdent parfaitement sans aucune durée de transition. Cependant, en pratique, les légères erreurs de prédictions des durées de migration (autour de 7%) impliquent la synchronisation des groupes de migrations pour maintenir l'ordonnancement initial et éviter les chevauchements indésirables de certaines actions. A la fin de l'expérimentation, le temps de complétion ex-

cède l'estimation de BtrPlace de seulement 5 secondes soit 1,09% de la durée totale d'exécution.

6.4.2 Évaluation de la limitation de puissance

La contrainte *PowerBudget* à pour but de limiter la puissance instantanée consommée par l'infrastructure pendant le processus de reconfiguration. Celle-ci nous a été initialement inspirée des besoins du projet européen DC4Cities [Dc4] qui vise à adapter la charge dans les centres de données en fonction de la disponibilité de l'énergie renouvelable. De manière générale, la limitation de puissance est globalement utilisée dans les centres de données pour réduire les couts d'exploitation (électricité, refroidissement, etc.) [Bha+13]. En restreignant la consommation au cours du temps, la contrainte *PowerBudget* peut différer certaines migrations ou n'importe quelle action en fonction de la puissance qu'elle consomme. Pour valider l'effet de la contrainte sur les décisions d'ordonnancement, nous avons exécuté le même scénario de décommissionnement que précédemment en ajoutant une contrainte *PowerBudget* pour limiter la consommation de puissance instantanée à 9 kW.

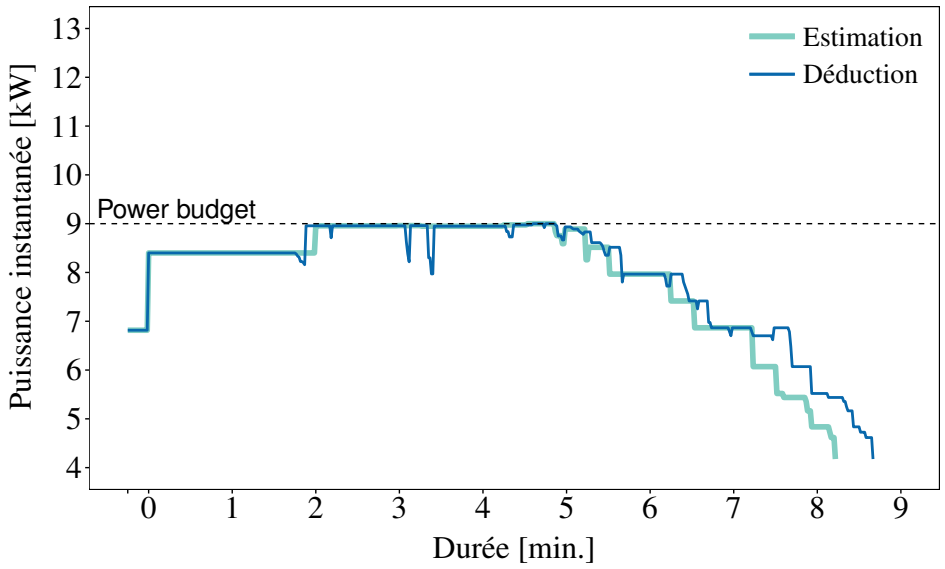


FIGURE 6.9 – Limitation de la puissance instantanée consommée.

La Figure 6.9 représente la consommation de puissance prédite et déduite de l'infrastructure pendant la reconfiguration avec la contrainte *PowerBudget*. On observe d'abord que, par rapport à l'exécution sans contrainte de puissance représentée en Figure 6.8, l'ordonnancement calculé par BtrPlace réduit avec succès l'ensemble des pics de consommation de puissance pour

rester en dessous de la limite fixée. En pratique, au lieu de démarrer les hôtes de destination simultanément, la contrainte force la répartition des actions de démarrage d'hôtes durant les 5 premières minutes de l'exécution pour ne pas dépasser les 9 kW de puissance instantanée. Un premier ensemble d'actions de démarrage d'hôtes est exécuté au début de l'expérimentation et termine à la $2^{ème}$ minute d'exécution. Ensuite, les actions restantes (démarrage et extinction d'hôtes, migrations de VMs) sont ordonnancés plus tard par petits groupes qui se recouvrent partiellement. En effet, entre la $2^{ème}$ et la $5^{ème}$ minute, on remarque que la consommation de puissance est très proche de la limite de 9 kW. Les groupes d'actions exécutés pendant cette période sont sélectionnés avec soin par BtrPlace, par exemple la parallélisation des migrations est restreinte pour ne pas dépasser le budget de puissance. Au final, l'exécution de l'ordonnancement a nécessité 90 secondes additionnelles par rapport à l'exécution sans contrainte représentée en Figure 6.8.

Malgré une précision d'estimation des durées de migration de 93% en moyenne, on observe tout de même un temps de complétion plus élevé avec 32 secondes supplémentaires par rapport à la prédiction de BtrPlace. Ce délai supplémentaire est principalement dû à la grande quantité de points de synchronisation inséré par notre module d'exécution pour éviter l'accumulation d'erreurs de prédictions et maintenir l'ordonnancement calculé. De plus, tel que pour l'expérimentation en section 6.2.1, la latence accumulée pour contacter les hyperviseurs et démarrer les migrations vient également prolonger le temps de complétion.

6.5 Passage à l'échelle de l'ordonnanceur

Calculer le moment de démarrage de chaque migration en fonction de l'allocation de bande passante est un problème NP-difficile. En pratique, le temps nécessaire à BtrPlace pour calculer des ordonnancements dépend de la quantité de VMs à migrer, du nombre d'éléments réseaux et de leurs capacités. Nous avons évalué successivement l'accélération des temps de résolution de BtrPlace avec l'optimisation *MaxBandwidth* et le délai supplémentaire induit par notre nouveau modèle par rapport au modèle d'origine de BtrPlace.

6.5.1 Configuration expérimentale

Pour évaluer le passage à l'échelle de notre modèle d'ordonnancement, nous avons repris la même configuration expérimentale que pour l'évaluation énergétique précédente en section 6.4. Nous l'avons ensuite agrandit jusqu'à 18 fois sa taille d'origine selon deux facteurs d'échelle différents.

Le premier facteur est directement lié à la taille de l'infrastructure et concerne la capacité du commutateur d'agrégation et le nombre de baies d'hôtes. À la plus large échelle (×10), chaque instance consiste à ordonnancer les migrations de 960 VMs depuis 20 baies et à destination de 10 nouvelles baies, chacune d'entre elles comprenant 24 hôtes. Alors que l'ensemble des hôtes sont connectés à leur commutateur ToR via des interfaces réseaux à 1 Gbps, les commutateurs d'agrégation fournissent une bande passante de 100 Gbps que nous considérons comme élevé pour un centre de données. Au niveau du fonctionnement interne de BtrPlace, cette expérimentation évalue l'impact de l'ajout de contraintes cumulatives avec une quantité de tâches fixe.

Le second facteur d'échelle concerne l'augmentation du nombre de VMs et de la capacité d'hébergement des hôtes (ressources mémoire et CPU). À la plus large échelle (×18), chaque instance consiste à calculer l'ordonnancement de 1728 migrations depuis 2 baies de 24 hôtes chacun et à destination d'une baie unique. La consolidation atteint 72 VMs par hôte de destination et nous considérons que cette valeur est suffisamment élevée pour représenter les centres de données actuels les plus performants. Par exemple, lorsque chaque VM nécessite 4 cœurs virtuels, ce placement peut suffire à saturer la dernière génération de serveurs Bull, le *bullion S*, équipé d'un total de 288 cœurs répartis sur 16 processeurs Intel E7 v3. Au niveau du fonctionnement interne de BtrPlace, cette expérimentation évalue les conséquences de l'augmentation du nombre de tâches sur les contraintes cumulatives existantes.

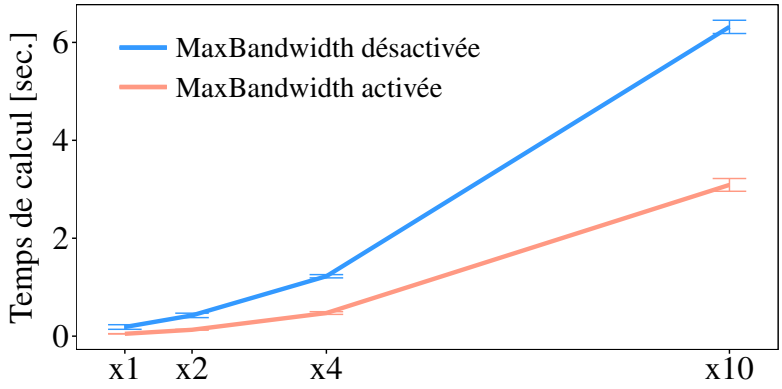
Pour obtenir des temps de complétions représentatifs, nous avons généré aléatoirement 100 instances différentes pour chaque échelle. Les résultats donnés représentent la moyenne de 10 exécutions consécutives pour chaque instance.

6.5.2 Optimisation

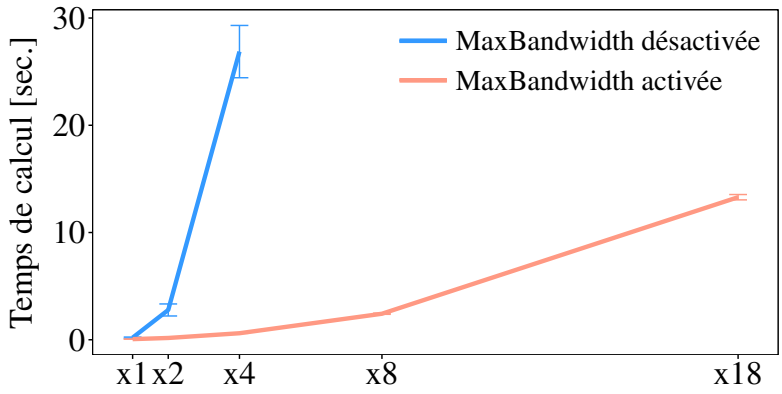
Dans cette section, nous comparons les temps de calculs de BtrPlace avec et sans l'optimisation *MaxBandwidth*. Nous évaluons d'abord l'intérêt de l'optimisation *MaxBandwidth* en comparant les temps de calculs obtenus avec et sans son utilisation. Comme expliqué au chapitre 4, l'optimisation *MaxBandwidth* consiste à forcer une allocation de bande passante maximale par migration et ainsi à pré-calculer les durées de migration. L'optimisation ramène le problème d'ordonnancement à un RCPSP classique à simple mode ce qui en réduit la complexité de résolution.

Pour ces expérimentations, chaque hôte peut héberger jusqu'à 4 VMs, la quantité maximale de migrations entrantes ou sortantes d'un hôte est ainsi limitée à 4. Pour évaluer BtrPlace sans l'optimisation *MaxBandwidth*, nous

avons dû autoriser toute possibilité de parallélisation des migrations (1, 2, 3 et 4 à la fois). Pour ce faire, nous avons alors configuré BtrPlace pour accepter 4 bandes passantes différentes par migration (1 Gbps, 500 Mbps, 333 Mbps et 250 Mbps) en ramenant ainsi le problème à un MRCPSP à 4 modes. Les instances sont générées en sélectionnant aléatoirement un placement initial et final pour chaque VM tout en respectant les capacités d'hébergement des hôtes. Ainsi, à la seule différence des placements aléatoires, l'expérimentation reste la même que celle réalisée en section 6.4 pour l'évaluation énergétique de notre modèle d'ordonnancement.



(a) Passage à l'échelle de l'infrastructure.



(b) Passage à l'échelle du nombre de VMs.

FIGURE 6.10 – Comparaison des durées de résolution de BtrPlace en fonction de l'activation de l'optimisation *MaxBandwidth* et du facteur d'échelle. Lorsque l'optimisation est désactivée, 4 bandes passantes différentes sont disponibles par migration.

La Figure 6.10 représente les temps de calcul moyens accompagnés des intervalles de confiance (95%) pour chaque modèle et facteur d'échelle utilisé. Pour alléger la représentation des données, nous avons écarté les instances

Tableau 6.5 – Pourcentages d'instances résolues en moins d'une minute en fonction de l'activation de l'optimisation *MaxBandwidth* et du facteur d'échelle.

Échelle	**Option *MaxBandwidth*** désactivée	activée
x1	65%	100%
x2	22%	100%
x4	83%	100%
x10	62%	100%

(a) Passage à l'échelle de l'infrastructure.

Échelle	**Option *MaxBandwidth*** désactivée	activée
x1	66%	100%
x2	57%	100%
x4	47%	100%
x8	0%	100%
x18	0%	100%

(b) Passage à l'échelle du nombre de VMs.

nécessitant un temps de calcul supérieur à une minute. Les statistiques de résolution sont toutefois représentées dans le Tableau 6.5.

Nous avons calculé que l'activation de l'optimisation *MaxBandwidth* réduit les temps de calculs jusqu'à 74% pour le premier facteur d'échelle, et augmente également la quantité d'instances résolues en moins d'une minute. Lorsque la taille de l'infrastructure augmente (voir Figure 6.10a), le temps de calcul augmente de façon exponentiel pour les deux modèles sans toutefois dévier significativement. À l'échelle la plus grande, l'activation de l'optimisation réduit les temps de complétion de 3,2 secondes en moyenne ce qui représente une accélération de 51% par rapport à la version non optimisée du modèle. Lorsque l'optimisation est désactivée et que la quantité de VMs augmente (voir Figure 6.10b), seules quelques instances d'échelles $\times 1$ à $\times 4$ ont été résolues en moins d'une minute. Ces résultats expliquent ainsi la dispersion des temps de calculs (grande intervalle de confiance) et l'augmentation exponentielle de la durée de résolution en fonction de l'échelle utilisée. A l'opposé, l'activation de l'optimisation a permis de résoudre l'ensemble des instances en moins d'une minute avec des temps de calcul en moyenne 43 fois plus faibles à l'échelle $\times 4$.

Le Tableau 6.5 montre que la quantité d'instances résolues ne diminue pas nécessairement lorsque la taille de l'infrastructure augmente. En effet, lorsque l'optimisation est désactivée, seules 22% des instances ont été résolues en moins d'une minute à l'échelle $\times 2$ alors que 83% sont résolues à l'échelle $\times 4$. Cette irrégularité s'explique par les placements aléatoires des VMs. En effet, certains placements réalisés à faible échelle peuvent alors générer une quantité plus élevée de contraintes cumulatives que d'autres placements réalisés à plus grande échelle. De plus, avec des tâches à hauteurs et durées variables, l'accumulation de contraintes cumulatives peut facilement entraîner le solveur vers

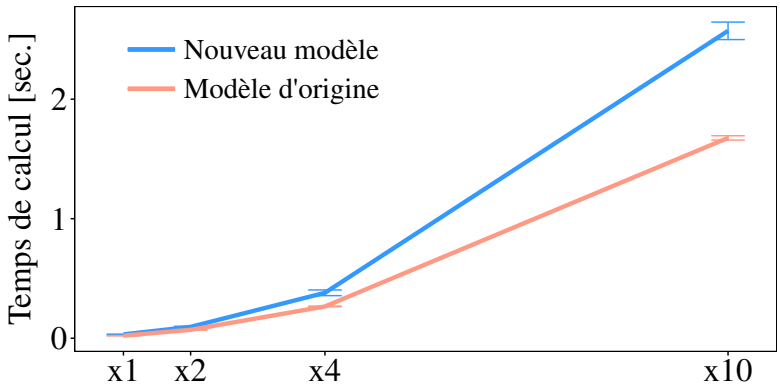
de mauvaises décisions d'instanciations des variables et augmenter de façon conséquente les temps de calculs.

En conclusion, cette expérimentation démontre que l'optimisation *Max-Bandwidth*, qui permet le pré-calcul des durées de migration, améliore significativement les performances de BtrPlace sans affecter la qualité de l'ordonnancement.

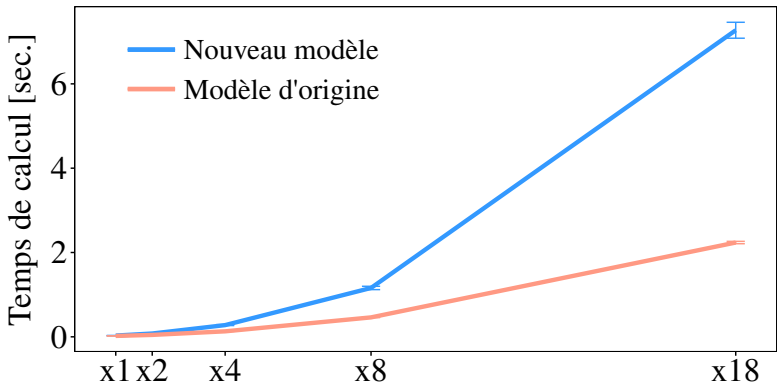
6.5.3 Surcoût du nouveau modèle d'ordonnancement

Pour cette expérimentation, nous comparons les durées de résolution du nouveau modèle d'ordonnancement de BtrPlace avec sa version d'origine sur des instances générées aléatoirement. Nous avons réalisé une expérimentation préliminaire dans une étude précédente [KMH15c], cependant les instances générées étaient trop spécialisées et représentaient uniquement des scénarios de décommissionnement symétriques. Nous avons constaté que l'heuristique initiale employée était trop spécifique et empêchait BtrPlace de résoudre certaines instances aléatoires en moins d'une minute. Les optimisations présentées au chapitre 5 ont permis de corriger les problèmes de robustesse de notre précédente heuristique. Toutes les instances, générées aléatoirement, sont désormais résolues par BtrPlace en moins d'une minute. De plus, pour fournir une analyse robuste de notre modèle d'ordonnancement qui ne repose pas que sur des scénarios de décommissionnement, les placements de VMs sont désormais réalisés aléatoirement parmi l'ensemble des hôtes de l'infrastructure. La Figure 6.11 montre les résultats obtenus en faisant varier la taille de l'infrastructure et la quantité de VMs. À la plus petite échelle, notre nouveau modèle d'ordonnancement nécessite en moyenne 9,4 ms supplémentaires pour trouver une solution, une augmentation de 48% par rapport au modèle d'origine de BtrPlace. Lorsque la taille de l'infrastructure est soumise à un facteur d'échelle $\times 10$ (voir Figure 6.11a), l'augmentation du temps de calcul du nouveau modèle atteint 82%. Malgré cette augmentation, le temps de calcul moyen du nouveau modèle reste très faible : seules 2,6 secondes sont requises pour calculer l'ordonnancement de 960 migrations entre 720 hôtes. Lorsque la quantité de VMs augmente (voir Figure 6.11b), l'augmentation du temps de calcul du nouveau modèle atteint 231,8% pour un facteur d'échelle $\times 18$. La durée moyenne de résolution reste à nouveau relativement faible et plafonne à 7,3 secondes pour l'ordonnancement de 1728 migrations entre 72 hôtes.

L'augmentation des durées de résolution est expliquée en pratique par les calculs additionnels effectués par le nouveau modèle pour fournir des ordonnancements fiables, rapides et économes en énergie. Même si l'augmentation relative à l'ancien modèle est significative, la durée de résolution reste négligeable par rapport à la durée d'exécution des ordonnancements. En effet, en



(a) Passage à l'échelle de l'infrastructure.



(b) Passage à l'échelle du nombre de VMs.

FIGURE 6.11 – Comparaison des durées de résolution du nouveau modèle d'ordonnancement de BtrPlace par rapport à son modèle d'origine et du facteur d'échelle.

augmentant le nombre de VMs par un facteur $\times 10$ les durées d'exécution prédites par BtrPlace sont comprises entre 152 et 224 secondes, le temps de calcul de BtrPLace ne représente alors en moyenne que 3,9% du temps d'exécution de l'ordonnancement. Lorsque la taille de l'infrastructure est augmentée par un facteur $\times 18$, les durées d'exécution sont estimées de 22 à 26 minutes pour l'ordonnancement le plus long. Le temps de calcul de BtrPlace ne représente alors en moyenne que 0,2% du temps d'exécution estimé.

On observe finalement, à facteurs d'échelle équivalents, que le temps de calcul supplémentaire requis par le nouveau modèle d'ordonnancement est plus élevé lorsque la quantité de VMs augmente plutôt que la taille de l'infrastructure. Cette différence s'explique par l'implémentation du modèle réseau. En effet, une contrainte cumulative a une complexité temporelle de $\mathcal{O}(n^2)$ où n est le nombre de tâches placées sur les ressources de la contrainte. Cela indique

alors que l'ajout de tâches sur une même contrainte cumulative (représenté par l'ajout de migrations au travers d'un même élément réseau) requiert plus de temps de calcul que l'ajout de contraintes cumulatives avec une même quantité de tâches.

On observe également des temps de calculs plus faibles que pour les scénarios de décommissionnement représentés en Figure 6.10. Cette différence s'explique par l'utilisation des liens réseaux. En effet, alors que la quantité de migrations est identique dans les deux configurations, la dissolution des groupes d'hôtes source et destination réalisée dans cette dernière expérimentation entraîne une consolidation par hôte plus faible. Dès lors, les interfaces réseaux des hôtes sont utilisées de manière moins intensive ce qui réduit par conséquent le nombre de tâches par contrainte cumulative. De plus, contrairement aux scénarios de décommissionnement, l'ensemble des liens réseaux peuvent également être utilisés en duplex-intégral. L'utilisation des deux sens de communication des liens se traduit par plus de contraintes cumulatives (1 pour chaque sens de communication) avec moins de tâches ce qui réduit le temps de calcul global.

À une très grande échelle, la durée de résolution de BtrPlace peut tout de même devenir significative par rapport au temps d'exécution des ordonnancements calculés. Pour surpasser cette limitation, une solution simple consiste à diviser l'opération de reconfiguration en plusieurs étapes. En effet, lorsque la bande passante utilisée pour migrer les VMs excède la capacité du commutateur d'agrégation par exemple, BtrPlace décide alors d'ordonnancer les migrations par groupes. Par conséquent, avec une capacité d'agrégation limitée à 100 Gbps, demander à BtrPlace de calculer soit un ordonnancement de 960 VMs, soit deux ordonnancements consécutifs de 480 VMs, conduirait à un résultat globalement similaire au vu des limitations de parallélisation induites par le réseau.

6.6 Conclusion

Les décisions d'ordonnancement de BtrPlace ont été validées au travers d'expérimentations représentatives réalisées en environnement réel. Les résultats obtenus ont été comparés à la version originale de BtrPlace et à un ordonnanceur qui reproduit les décisions de *Memory Buddies* (MB). De nombreuses micro-expérimentations ont démontré que notre nouveau modèle d'ordonnancement surpasse toutes les configurations des deux ordonnanceurs concurrents. Sur 50 plans de migration générés aléatoirement, les migrations exécutées depuis notre modèle d'ordonnancement ont terminé en moyenne 20,4% plus rapidement qu'avec la meilleure configuration de MB et les temps de complétion

ont été réduits de 28,1%. Contrairement à la parallélisation statique de MB, notre ordonnanceur s'avère toujours plus performant que des exécutions purement séquentielles avec une accélération moyenne des temps de complétion de 54,18%. De plus, les durées de migration sont proches de l'optimum avec un ralentissement moyen de seulement 7,35%, soit 4,5 fois plus faible qu'avec MB.

La précision du modèle de migration a été validé en comparant l'exécution de plans de migration aléatoires dans un simulateur et dans un environnement réel. Nous avons comparé les prédictions de notre modèle par rapport au simulateur SimGrid ainsi que deux autres modèles de migration représentatifs des solutions d'ordonnancement actuelles. Les résultats montrent que notre modèle de migration est plus précis que les simulateurs existants avec une précision moyenne de 93,9%.

Des expérimentations à plus large échelle ont été réalisées et ont permis d'exhiber l'intérêt pratique de BtrPlace à répondre à des besoins énergétiques. Dans un scénario de décommissionnement impliquant 96 migrations entre 72 hôtes, notre modèle d'ordonnancement à réduit la consommation énergétique de l'opération de 21,5% par rapport à la version originale de BtrPlace. Nous avons également validé la capacité de contrôle énergétique de notre ordonnanceur en limitant la consommation de puissance instantanée pendant toute la période de reconfiguration d'un centre de données. En fonction d'un budget de puissance pré-défini, BtrPlace est désormais capable de reporter des actions de reconfiguration (migration, démarrage et extinction d'hôtes) en fonction de leur consommation de puissance respective pour garantir le respect du budget donné.

Finalement, l'évaluation du passage à l'échelle de notre modèle a révélé une capacité de calcul confortable en générant l'ordonnancement de milliers de migrations aléatoires en quelques secondes seulement. En faisant varier le taux de consolidation et la taille de l'infrastructure, nous avons observé que le temps de calcul requis pour un ordonnancement représente moins de 1% de sa durée d'exécution.

Chapitre 7

Conclusion

Sommaire

7.1 Résumé 105
7.2 Perspectives 107
7.2.1 Extensions réseaux 108
7.2.2 Convergence du placement et de l'ordonnancement . . 109
7.2.3 Viabilité de l'approche à long terme 109

7.1 Résumé

La migration à chaud de machines virtuelles est utilisé quotidiennement par les algorithmes de consolidation et les opérateurs de centres de données pour gérer les VMs sur les hôtes en production. Nous avons constaté que les gestionnaires de VMs actuels calculent des placements de qualité mais négligent l'ordonnancement des migrations générées. En effet, en reposant sur des hypothèses peu réalistes, les ordonnancements mènent à de longues et coûteuses migrations qui viennent réduire les bénéfices escomptés par la réorganisation des VMs.

Dans cette thèse, nous avons d'abord dressé un état de l'art relatif aux migrations à chaud dans les centres de données virtualisés ainsi qu'aux techniques d'ordonnancement de migrations multiples. Nous avons ensuite proposé, implémenté puis évalué un nouvel ordonnanceur de migrations au sein du gestionnaire de VMs BtrPlace. Pour répondre aux problèmes et lacunes des ordonnanceurs existants, notre modèle d'ordonnancement s'appuie désormais sur l'architecture réseau et l'analyse de l'activité mémoire des VMs pour déduire le meilleur moment de démarrage des migrations et la quantité adéquate de bande passante à leur allouer.

L'analyse de l'état de l'art sur les migrations à chaud a révélé l'impact non négligeable de l'activité mémoire des VMs sur le comportement du processus de migration. Nous avons d'abord proposé une méthode d'analyse rapide et efficace de l'activité mémoire des VMs que nous avons implémenté dans l'hyperviseur KVM. A partir de l'analyse de l'activité mémoire des VMs, nous

avons défini un modèle de migration à chaud reposant sur l'algorithme de pré-copie ainsi que les optimisations communément réalisées par les hyperviseurs existants pour réaliser des prédictions précises des performances des migrations. Plusieurs expérimentations ont évalué la précision moyenne de notre modèle de migration à 93,9%, offrant de meilleurs résultats que les algorithmes de placement et d'ordonnancement actuels ainsi que le modèle de migration implémenté dans le simulateur SimGrid.

L'analyse de l'état de l'art sur l'ordonnancement de migrations multiples ainsi que l'étude de différent scénarios de migration ont révélé l'importance de l'analyse et de la gestion du réseau pour garantir des ressources stables aux migrations et offrir des opportunités de parallélisation efficaces. Nous avons alors défini un modèle réseau générique que nous avons implémenté dans BtrPlace. Le modèle en question permet de définir l'architecture d'un centre de données tout en y représentant les capacités des différents éléments bloquants qui composent le réseau. Les ordonnancements calculés profitent ainsi d'une parallélisation dynamique des migrations adaptée à la topologie et aux capacités du réseau. Des expérimentations effectuées depuis des plans de migration générés aléatoirement ont montré une accélération des durées de migration d'au moins 20,4% par rapport aux stratégies de parallélisation statique. De manière générale, l'utilisation conjointe du modèle de migration et du modèle réseau ont réduit les durées d'exécution des ordonnancements calculés de 28,1% en moyenne par rapport à la meilleure stratégie de parallélisation statique réalisable. Notre approche bénéficie également d'une meilleure robustesse en garantissant la terminaison de l'ensemble des migrations. Les problèmes de faisabilité des migrations ordonnancées par le modèle original de BtrPlace ont ainsi été entièrement résolus ce qui améliore grandement la fiabilité du gestionnaire de VMs.

Une particularité fondamentale de notre approche repose sur l'utilisation de la programmation par contraintes pour la modélisation et la résolution du problème d'ordonnancement. L'expressivité et la composabilité de la PPC nous ont permis de contrôler finement l'ordonnancement des migrations en définissant des contraintes spécifiques qui viennent s'intégrer naturellement au problème d'ordonnancement pour influer sur les décisions de l'ordonnanceur. Nous avons validé cet aspect en développant 3 contraintes d'ordonnancement pour forcer la parallélisation ou la séquentialisation d'un ensemble de migrations ainsi que pour agir sur leur ordre d'exécution. Ces contraintes suffisent à contrôler pleinement l'aspect temporel de l'ordonnancement et sont facilement utilisables par le biais de scripts de configuration ou directement via l'API de BtrPlace. La composabilité de notre approche permet également d'étendre aisément le modèle d'ordonnancement en définissant par exemple de nouveaux protocoles réseau si besoin, différents modèles de migration, ou encore des

objectifs personnalisés à atteindre (minimiser ou maximiser une variable spécifique). En s'appuyant sur les possibilités d'extensions offertes par BtrPlace, nous avons intégré un modèle énergétique existant pour estimer la consommation énergétique des migrations. Nous avons ensuite complété le modèle énergétique par des mesures effectuées sur notre infrastructure expérimentale pour être en mesure d'estimer précisément la consommation énergétique des différentes actions de reconfiguration générées par BtrPlace. Ainsi, la définition d'une stratégie de minimisation énergétique a réduit l'énergie consommée par un scénario de décommissionnement de 21,5% par rapport à la version originale de BtrPlace. De plus, le développement d'une contrainte de limitation de puissance instantanée a validé la capacité de contrôle énergétique de notre ordonnanceur qui est désormais capable de reporter des actions de reconfiguration pour respecter un budget énergétique donné.

L'analyse de la performance des migrations et des opportunités de parallélisation des migrations sur différents types de réseaux nous a conduit vers la réalisation d'une heuristique générique efficace et vers la définition d'une optimisation globale basée sur des règles de parallélisation fondamentales. Ces optimisations ont réduit la complexité initiale de notre problème d'ordonnancement tout en conservant des décisions d'ordonnancement efficaces qui assurent des durées de migration et des temps de complétion minimaux. Finalement, les temps de calcul de notre ordonnanceur représentent approximativement 1% des durées d'exécution produites par les ordonnancements calculés.

7.2 Perspectives

Tout au long de cette thèse, nous avons présenté nos analyses et contributions dans le domaine de l'ordonnancement de migrations à chaud de VMs. Bien que ces travaux aient fait l'objet de mures réflexions et de validations expérimentales rigoureuses, nous discutons dans cette section des limitations de notre approche, des améliorations à y apporter et de sa viabilité à plus long terme. Nous discutons d'abord des limites actuelles de notre modèle réseau et des possibilités d'extensions de ce dernier. Nous apportons ensuite des pistes de réflexions sur l'évolution générale de BtrPlace en soulignant le lien particulier entre les décisions de placement et d'ordonnancement des VMs. Finalement, nous analysons et posons des questions ouvertes sur la viabilité de notre approche à plus long terme et essayons d'y répondre.

7.2.1 Extensions réseaux

La configuration réseau de notre environnement d'exécution étant relativement basique, nous avons implémenté un modèle réseau minimal composé de liens et de commutateurs. Bien que nous ayons ajouté la possibilité de construire son réseau manuellement depuis un ensemble de routes statiques, aucun protocole de routage n'est implémenté pour le moment. Ainsi lorsque plusieurs chemins sont disponibles pour une migration, le modèle réseau actuel se contente de sélectionner la première route trouvée. L'implémentation d'un protocole de routage minimal tel que RIP ou encore OSPF permettrait de gérer des réseaux plus complexes et de bénéficier des plus courts chemins de migration ou encore de répartir la charge entre les différentes routes disponibles. La définition de réseaux logiques ou virtuels (VLAN) peuvent également être implémentées pour identifier et séparer les flux représentant différents types de migrations par exemple. L'implémentation d'un protocole d'agrégation de liens tel que LACP est également envisageable mais nécessite des connaissances approfondies des configurations des équipements réseaux pour reproduire à l'identique le comportement des négociations LACP réalisées. Au cours de cette thèse, nous avons réalisé avec succès des expérimentations à grande échelle sur une infrastructure disposant d'une agrégation de plusieurs liens réseau à 1 Gbps [KMH14]. N'ayant pas eu accès à la configuration du commutateur réseau, nous avons tout de même été capable de reproduire l'infrastructure expérimentale depuis notre modèle réseau via une analyse approfondie de la topologie avec l'outil de mesure de bande passante *iperf*. Ainsi, lorsque l'agrégation de liens est configuré de manière à générer des routes statiques, notre modèle actuel est suffisant pour reproduire ce type de réseau en automatisant par exemple le sondage de la topologie et du choix des liens d'agrégation puis en important les routes statiques observées.

Lorsque l'optimisation actuelle *MaxBandwidth* est activée, la bande passante disponible la plus élevée est automatiquement assignée aux migrations. Lorsque les moyens techniques ne permettent pas de dédier un réseau à la migration de VMs, les opérateurs des centres de données ne peuvent pas garantir une bande passante fixe aux migrations et les décisions de parallélisation doivent être adaptées en conséquence. Il est alors intéressant de considérer une nouvelle contrainte d'ordonnancement qui consisterait à ajuster la capacité de parallélisation des migrations par lien en diminuant par exemple la bande passante disponible (déclaration de flux concurrents). En effet, avec la connaissance de l'utilisation du réseau, un opérateur peut décider de réduire la bande passante utilisable (et donc la parallélisation) sur certains liens qui sont soumis à des trafics concurrents. De plus, comme pour la contrainte de limitation de puissance que nous avons développé, cette limitation pourrait

être limitée dans le temps et effective uniquement pendant une période de temps pré-définie. Par exemple, lors de la réalisation d'une tâche de maintenance récurrente qui peut être quantifiée, telle que la mise à jour périodique des systèmes d'exploitation où une grande quantité de données doit alors être transférée depuis un serveur de stockage à destination des hôtes.

7.2.2 Convergence du placement et de l'ordonnancement

À plus long terme, il serait intéressant de fusionner le modèle de placement de BtrPlace avec notre modèle d'ordonnancement pour converger vers un gestionnaire de VMs qui puisse prendre des décisions conjointes de placement et d'ordonnancement. Actuellement, si aucun placement spécifique n'est imposé et que BtrPlace décide de migrer plusieurs VMs, le nouveau placement des VMs est calculé indépendamment de l'ordonnancement des migrations. En effet, tel que décrit en fin de section 5.2.1, nous avons réalisé une résolution en deux phases qui consiste à calculer consécutivement et de façon séparée le nouveau placement des VMs puis l'ordonnancement des migrations générées. En conséquence, certains ordonnancements peuvent être considérés sous-optimaux par rapport aux attentes de l'algorithme de placement en terme de réactivité. Il serait alors intéressant de prendre des décisions de placement qui tiennent compte des problématiques d'ordonnancement sous-jacentes. En effet, les décisions de placement s'appuient essentiellement sur les besoins en ressources des VMs et les contraintes de placement définies par l'utilisateur. Dès lors, si plusieurs hôtes valident l'ensemble des contraintes permettant l'hébergement d'une même VM par exemple, l'hôte final est alors sélectionné aléatoirement parmi les hôtes candidats. Dans une version future du gestionnaire de VMs, le choix de l'hôte pourrait s'appuyer sur les contraintes d'ordonnancement de l'utilisateur ainsi que sur le modèle réseau par exemple pour minimiser la durée des migrations et le temps d'exécution de la reconfiguration. Ainsi, les contraintes d'ordonnancement définies dans BtrPlace pourraient également impacter ses décisions de placement. Par exemple, l'ordonnanceur peut actuellement introduire un délai sur une migration comme conséquence d'un mauvais choix du serveur de destination. Avec un couplage fort entre les deux modèles, l'algorithme de placement serait alors capable de réviser ses décisions en fonction des performances de l'ordonnancement des migrations.

7.2.3 Viabilité de l'approche à long terme

L'augmentation considérable du volume de trafic échangé au sein des centres de données et l'évolution des équipements réseaux pousse les opérateurs de centres de données à disposer de bandes passantes toujours plus

élevées. De nos jours, il est courant de voir des serveurs équipés de plusieurs interfaces allant de 10 Gbps à 40 Gbps, et jusqu'à 100 Gbps pour le cœur de réseau. Ainsi, en considérant l'évolution future des centres de données, l'exploitation d'interfaces à 100 Gbps par serveur pourrait-elle avoir une incidence sur le modèle d'ordonnancement présenté dans cette thèse ? En effet, nous avons vu qu'une bande passante plus élevée permet de réduire non seulement les durées de migration mais aussi les durées d'interruption des VMs. Nous avons également constaté que la durée d'interruption maximale définie par défaut dans KVM (30 ms) est un bon compromis pour la plupart des migrations à chaud disposant d'une bande passante de 1 Gbps. Cette durée d'interruption permet en effet de conserver un état cohérent de la VM entre les hôtes sans prolonger excessivement sa durée de migration. Dès lors, en disposant d'une vitesse de transfert 100 fois plus élevée (100 Gbps), il serait théoriquement possible de transmettre les pages mémoire modifiées suffisamment rapidement pour pouvoir augmenter la parallélisation des migrations sur un même chemin réseau sans impacter la durée totale de migration. L'optimisation MaxBandwidth que nous avons proposé pourrait ainsi être ajustée par rapport aux capacités des interfaces et de l'activité mémoire des VMs. Une analyse comparative des quantités d'itérations de pré-copie est nécessaire pour pouvoir statuer sur la viabilité de l'augmentation du parallélisme sur des liens à très haute capacité.

La *conteneurisation*, représentant une forme de virtualisation légère, a vu sa popularité monter en flèche depuis 2013 avec l'apparition du gestionnaire de conteneurs Docker [Mer14]. À la différence de la virtualisation, la conteneurisation se substitue au système d'exploitation virtualisé et aux couches d'hypervision en exécutant directement les applications. Cette technique permet d'exécuter des applications de manière isolée des autres processus tout en partageant les ressources du système d'exploitation de la machine hôte et même parfois les bibliothèques installées. L'objectif de la conteneurisation est de proposer l'environnement le plus léger possible pour l'exécution d'applications afin de minimiser l'utilisation des ressources de la machine hôte. Avec Docker, chaque conteneur est généralement dédié à l'exécution d'un seul processus, par exemple l'exécution d'un serveur Web disposant d'une base de données nécessitera deux conteneurs, un pour le serveur Web et un autre pour la base de données. Initialement basé sur les conteneurs Linux (LXC), la simplicité d'utilisation du gestionnaire de conteneurs Docker a suscité un fort intérêt pour la conteneurisation et c'est rapidement développé pendant ces 3 dernières années. L'utilisation de plus en plus intensive de la conteneurisation a vu apparaître des outils de gestion de grappes de conteneurs tels que *Kubernetes* [Kub] et *Docker Swarm* [Swa] qui permettent de gérer le déploiement, la surveillance et la maintenance de milliers de conteneurs.

Aujourd'hui, de nombreux fournisseurs de services proposent l'hébergement et la gestion de conteneurs Docker tels que Google [Docb], Amazon [Doca] ou encore Microsoft [Docc].

Parmi les nombreux avantages de la conteneurisation, on retrouve la capacité de migration des conteneurs entre différents hôtes. Contrairement à la migration de VMs qui nécessite entre autre le transfert de l'état mémoire du système d'exploitation entre les hôtes, la migration de conteneurs consiste essentiellement à sauvegarder et transférer l'état de l'application vers l'hôte de destination. Un nouveau conteneur est alors forgé sur l'hôte désiré à partir de l'état de l'application et le trafic réseau est enfin redirigé vers le nouveau conteneur. Ainsi, la durée de migration d'un conteneur est plus courte que pour une VM mais la durée d'interruption occasionnée est bien plus longue et correspond à toute la durée de migration. Bien que des solutions de répartition de charge permettent de limiter cette durée d'interruption, la migration de conteneurs disposant de stockage persistant tels que des bases de données par exemple entraîneront toujours une durée d'interruption inévitable bien supérieure à la migration de VMs. Ainsi, pour l'instant, il n'existe pas de modèle de migration à chaud de conteneurs et le modèle de migration actuel est simple et pourrait facilement être intégré à BtrPlace. De manière générale, en consommant beaucoup moins de ressources que les VMs et en profitant de la migration, l'utilisation des conteneurs permet d'augmenter considérablement la consolidation dans les centres de données par rapport à la virtualisation.

La conteneurisation est-elle sur le point de remplacer la virtualisation en entraînant de facto la fin de la migration de VMs ? Malgré l'intérêt croissant lié aux avantages de la conteneurisation, on observe tout de même certaines limitations. L'isolation des processus par exemple est moins robuste qu'avec la virtualisation et peut éventuellement poser des problèmes de sécurité. Le manque de compatibilité entre les versions de Docker empêche la migration d'un conteneur entre les systèmes d'exploitation Linux et Windows. La durée d'interruption élevée de la migration est également un frein à son utilisation sur des serveurs de production par exemple. De plus, il n'est pas possible de tout conteneuriser, comme par exemple les solutions d'*infrastructures de bureaux virtuels* (VDI) qui reposent sur un système d'exploitation complet et nécessitent l'exécution d'un grand nombre de processus. Chaque approche possède ainsi des avantages et des inconvénients et ne peuvent pas se remplacer l'une l'autre. Le choix entre conteneurisation et virtualisation revient aux utilisateurs finaux, certains préféreront déployer une VM pour disposer d'un système d'exploitation complet alors que d'autres préfèreront démarrer rapidement leur application dans un conteneur. De plus, au niveau de la gestion des centres de données des fournisseurs publics de conteneurs (Google, Amazon, etc.), on constate que les conteneurs sont exclusivement exécutés sur des

VMs. En effet, la complémentarité entre VMs et conteneurs semble être profitable pour les fournisseurs de services qui bénéficient alors des avantages de la virtualisation et de la conteneurisation. L'utilisation de VMs comme couche intermédiaire aux conteneurs permet en premier lieu d'utiliser un même système d'exploitation pour héberger l'ensemble des conteneurs et d'éviter tout problème de compatibilité. Ce choix offre également une plus grande flexibilité de consolidation, la quantité de VMs par hôte mais également la quantité de conteneurs par VMs peuvent alors être ajustés séparément. Finalement, pendant que les opérateurs de centres de données continuent de profiter des avantages de la migration à chaud de VMs pour gérer leur infrastructure, les utilisateurs finaux ont également la possibilité de migrer leurs conteneurs entre différentes infrastructures issues de fournisseurs différents par exemple.

Bibliographie

[Car82] Jacques Carlier. "The one-machine sequencing problem". In : *European Journal of Operational Research* 11.1 (1982), p. 42–47 (Cité en page 76).

[BLK83] J. Blazewicz, J.K. Lenstra et A.H.G.Rinnooy Kan. "Scheduling subject to resource constraints : classification and complexity". In : *Discrete Applied Mathematics* 5.1 (1983), p. 11 –24 (Cité en page 59).

[AB93] Abderrahmane Aggoun et Nicolas Beldiceanu. "Extending CHIP in order to solve complex scheduling and placement problems". In : *Mathematical and Computer Modelling* 17.7 (1993), p. 57–73 (Cité en page 67).

[BDP93] Rolfe E Buhrke, Dennis L DeBruler et Vikram Punj. *Arrangement for regulating traffic in a high speed data network*. US Patent 5,231,631. 1993 (Cité en page 35).

[LD93] Scott T Leutenegger et Daniel Dias. *A modeling study of the TPC-C benchmark*. T. 22. 2. ACM, 1993 (Cité en page 23).

[KSD95] Rainer Kolisch, Arno Sprecher et Andreas Drexl. "Characterization and generation of a general class of resource-constrained project scheduling problems". In : *Management science* 41.10 (1995), p. 1693–1703 (Cité en page 60).

[Kri98] Murali R Krishnan. *Adaptive bandwidth throttling for network services*. US Patent 5,799,002. 1998 (Cité en page 35).

[Say02] Roger Anthony Sayle. *Virtual network file server*. US Patent 6,356,863. 2002 (Cité en page 22).

[Wal02] Carl A. Waldspurger. "Memory Resource Management in VMware ESX Server." In : *OSDI*. Sous la dir. de David E. Culler et Peter Druschel. Operating Systems Review 36, Special Issue, Winter 2002. USENIX Association, 2002 (Cité en page 20).

[Bar+03] Paul Barham et al. "Xen and the art of virtualization". In : *ACM SIGOPS Operating Systems Review* 37.5 (2003), p. 164–177 (Cité en pages 2, 18).

[Cla+05] Christopher CLARK et al. “Live migration of virtual machines”. In : *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association. 2005, p. 273–286 (Cité en pages 3, 12, 13, 17).

[NLH+05] Michael NELSON, Beng-Hong LIM, Greg HUTCHINS et al. “Fast Transparent Migration for Virtual Machines.” In : *USENIX Annual technical conference, general track*. 2005, p. 391–394 (Cité en page 17).

[RVBW06] Francesca ROSSI, Peter VAN BEEK et Toby WALSH. *Handbook of constraint programming*. Elsevier, 2006 (Cité en page 64).

[Kiv+07] Avi KIVITY, Yaniv KAMAY, Dor LAOR, Uri LUBLIN et Anthony LIGUORI. “kvm : the Linux virtual machine monitor”. In : *Proceedings of the Linux symposium*. T. 1. 2007, p. 225–230 (Cité en pages 2, 17, 18).

[AFLV08] Mohammad AL-FARES, Alexander LOUKISSAS et Amin VAHDAT. “A scalable, commodity data center network architecture”. In : *ACM SIGCOMM Computer Communication Review*. T. 38. 4. ACM. 2008, p. 63–74 (Cité en page 32).

[JRL08] Narendra JUSSIEN, Guillaume ROCHART et Xavier LORCA. “Choco : an Open Source Java Constraint Programming Library”. In : *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*. Paris, France, 2008, p. 1–10 (Cité en page 64).

[Luo+08] Yingwei LUO et al. “Live and incremental whole-system migration of virtual machines using block-bitmap.” In : *CLUSTER*. IEEE Computer Society, 2008, p. 99–106 (Cité en pages 16, 23).

[VAN08] Akshat VERMA, Puneet AHUJA et Anindya NEOGI. “pMapper : power and migration cost aware application placement in virtualized systems”. In : *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*. Middleware '08. Leuven, Belgium : Springer-Verlag New York, Inc., 2008 (Cité en page 3).

[Her+09] F. HERMENIER, X. LORCA, J.-M. MENAUD, G. MULLER et J. LAWALL. “Entropy : a Consolidation Manager for Clusters”. In : *VEE*. Washington, DC, USA : ACM, 2009 (Cité en pages 3, 64).

[HG09] Michael R. Hines et Kartik Gopalan. "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning." In : *VEE*. Sous la dir. d'Antony L. Hosking, David F. Bacon et Orran Krieger. ACM, 27 mar. 2009, p. 51–60. URL : http://dblp.uni-trier.de/db/conf/vee/vee2009.html#HinesG09 (Cité en page 15).

[Jin+09] Hai Jin, Li Deng, Song Wu, Xuanhua Shi et Xiaodong Pan. "Live virtual machine migration with adaptive, memory compression." In : *CLUSTER*. IEEE Computer Society, 2009, p. 1–10 (Cité en page 20).

[SS09a] Alexander Stage et Thomas Setzer. "Network-aware Migration Control and Scheduling of Differentiated Virtual Machine Workloads". In : *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. CLOUD '09. Washington, DC, USA : IEEE Computer Society, 2009, p. 9–14 (Cité en page 26).

[SS09b] Alexander Stage et Thomas Setzer. "Network-aware migration control and scheduling of differentiated virtual machine workloads". In : *Proceedings of the 2009 ICSE workshop on software engineering challenges of cloud computing*. IEEE Computer Society. 2009, p. 9–14 (Cité en page 37).

[Woo+09] Timothy Wood et al. "Memory buddies : exploiting page sharing for smart colocation in virtualized data centers". In : *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM. 2009, p. 31–40 (Cité en pages 39, 81).

[Ako+10] Sherif Akoush, Ripduman Sohan, Andrew Rice, Andrew W Moore et Andy Hopper. "Predicting the performance of virtual machine migration". In : *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*. IEEE. 2010, p. 37–46 (Cité en pages 24, 52, 88).

[BB10] Anton Beloglazov et Rajkumar Buyya. "Energy Efficient Resource Management in Virtualized Cloud Data Centers". In : *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. CCGRID '10. Washington, DC, USA : IEEE Computer Society, 2010, p. 826–831 (Cité en page 3).

[HDG10] Michael R. Hines, Umesh Deshpande et Kartik Gopalan. "Post-copy live migration of virtual machines." In : *Operating Systems Review* 43.3 (1er fév. 2010), p. 14–26. URL : http://dblp.uni-trier.de/db/journals/sigops/sigops43.html#HinesDG09 (Cité en pages 14, 15).

[SA10] Vijayaraghavan Soundararajan et Jennifer M. Anderson. "The Impact of Management Operations on the Virtualized Datacenter". In : *SIGARCH Comput. Archit. News* 38.3 (juin 2010) (Cité en page 3).

[Swa10] Kent R Swalin. *Evaluating Microsoft Hyper-V Live Migration Performance Using IBM system x3650 M3 and IBM Systems Storage DS3400*. 2010 (Cité en page 17).

[Zha+10] Xiang Zhang, Zhigang Huo, Jie Ma et Dan Meng. "Exploiting Data Deduplication to Accelerate Live Virtual Machine Migration." In : *CLUSTER*. IEEE Computer Society, 2010, p. 88–96 (Cité en page 21).

[BKR11] David Breitgand, Gilad Kutiel et Danny Raz. "Cost-Aware Live Migration of Services in the Cloud." In : *Hot-ICE*. Sous la dir. d'Anees Shaikh et Kobus van der Merwe. USENIX Association, 2011 (Cité en page 26).

[Cal+11] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose et Rajkumar Buyya. "CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms". In : *Software : Practice and Experience* 41.1 (2011), p. 23–50 (Cité en pages 24, 89).

[DWG11] Umesh Deshpande, Xiaoshuang Wang et Kartik Gopalan. "Live gang migration of virtual machines". In : *Proceedings of the 20th international symposium on High performance distributed computing*. ACM. 2011, p. 135–146 (Cité en pages 39, 40).

[Liu+11] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong et Xiaofei Liao. "Performance and energy modeling for live migration of virtual machines." In : *HPDC*. Sous la dir. d'Arthur B. Maccabe et Douglas Thain. ACM, 2011, p. 171–182 (Cité en pages 25, 52, 76, 88, 94).

[STP11] Felix Salfner, Peter Tröger et Andreas Polze. "Downtime Analysis of Virtual Machine Live Migration". In : *The Fourth International Conference on Dependability (DEPEND 2011). IARIA*. 2011, p. 100–105 (Cité en page 24).

[Sv11a] Petter SVÄRD, Benoit HUDZIA, Johan TORDSSON et Erik ELMROTH. "Evaluation of Delta Compression Techniques for Efficient Live Migration of Large Virtual Machines". In : *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. VEE '11. Newport Beach, California, USA : ACM, 2011, p. 111–120 (Cité en page 20).

[Sv11b] Petter SVÄRD, Johan TORDSSON, Benoit HUDZIA et Erik ELMROTH. "High Performance Live Migration through Dynamic Page Transfer Reordering and Compression." In : *CloudCom*. Sous la dir. de Costas LAMBRINOUDAKIS, Panagiotis RIZOMILIOTIS et Tomasz Wiktor WLODARCZYK. IEEE Computer Society, 2011, p. 542–548 (Cité en page 19).

[Sv11c] Petter SVÄRD, Johan TORDSSON, Benoit HUDZIA et Erik ELMROTH. "High Performance Live Migration through Dynamic Page Transfer Reordering and Compression." In : *CloudCom*. Sous la dir. de Costas LAMBRINOUDAKIS, Panagiotis RIZOMILIOTIS et Tomasz Wiktor WLODARCZYK. IEEE Computer Society, 2011, p. 542–548 (Cité en page 20).

[Tro+11] Ulf TROPPENS, Rainer ERKENS, Wolfgang MULLER-FRIEDT, Rainer WOLAFKA et Nils HAUSTEIN. *Storage networks explained : basics and application of fibre channel SAN, NAS, iSCSI, infiniband and FCoE*. John Wiley & Sons, 2011 (Cité en page 22).

[Ver+11] Akshat VERMA, Gautam KUMAR, Ricardo KOLLER et Aritra SEN. "CosMig : Modeling the Impact of Reconfiguration in a Cloud." In : *MASCOTS*. IEEE Computer Society, 2011, p. 3–11 (Cité en page 24).

[WW11] Xiaorui WANG et Yefu WANG. "Coordinating power control and performance management for virtualized server clusters". In : *IEEE Transactions on Parallel and Distributed Systems* 22.2 (2011), p. 245–259 (Cité en page 77).

[Ye+11] Kejiang YE, Xiaohong JIANG, Dawei HUANG, Jianhai CHEN et Bei WANG. "Live migration of multiple virtual machines with resource reservation in cloud computing environments". In : *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE. 2011, p. 267–274 (Cité en page 37).

[DKG12] Umesh DESHPANDE, Unmesh KULKARNI et Kartik GOPALAN. "Inter-rack live migration of multiple virtual machines". In : *Proceedings of the 6th international workshop on Virtualization*

Technologies in Distributed Computing Date. ACM. 2012, p. 19–26 (Cité en page 40).

[LQL12] Chao Li, A. Qouneh et Tao Li. "iSwitch : Coordinating and optimizing renewable energy powered server clusters". In : *39th Annual International Symposium on Computer Architecture (ISCA)*. 2012 (Cité en page 3).

[TSH12] Kazushi Takahashi, Koichi Sasada et Takahiro Hirofuchi. "A fast virtual machine storage migration technique using data deduplication". In : *Proceedings of CLOUD COMPUTING* (2012), p. 57–64 (Cité en pages 21, 23).

[Ye+12] Kejiang Ye, Xiaohong Jiang, Ran Ma et Fengxi Yan. "Vc-migration : Live migration of virtual clusters in the cloud". In : *2012 ACM/IEEE 13th International Conference on Grid Computing*. IEEE. 2012, p. 209–218 (Cité en page 36).

[Bha+13] Arka A Bhattacharya, David Culler, Aman Kansal, Sriram Govindan et Sriram Sankar. "The need for speed and stability in data center power capping". In : *Sustainable Computing : Informatics and Systems* 3.3 (2013), p. 183–193 (Cité en page 96).

[HLM13] F. Hermenier, J. Lawall et G. Muller. "BtrPlace : A Flexible Consolidation Manager for Highly Available Applications". In : *IEEE Trans. on Dependable and Secure Computing* (2013) (Cité en pages 4, 64).

[HLP13] Takahiro Hirofuchi, Adrien Lèbre et Laurent Pouilloux. "Adding a Live Migration Model into SimGrid : One More Step Toward the Simulation of Infrastructure-as-a-Service Concerns". In : *IEEE 5th Intl. Conference on Cloud Computing Technology and Science*. T. 1. 2013, p. 96–103 (Cité en page 89).

[ST13a] Tusher Kumer Sarker et Maolin Tang. "Performance-driven live migration of multiple virtual machines in datacenters." In : *GrC*. Sous la dir. de Shuliang Wang, Xingquan Zhu et Tingting He. IEEE Computer Society, 2013, p. 253–258 (Cité en page 24).

[ST13b] Tusher Kumer Sarker et Maolin Tang. "Performance-driven live migration of multiple virtual machines in datacenters". In : *Granular Computing (GrC), 2013 IEEE International Conference on*. IEEE. 2013, p. 253–258 (Cité en pages 26, 38).

[Zhe+13] Jie Zheng, TS Ng, Kunwadee Sripanidkulchai et Zhaolei Liu. "Pacer : A Progress Management System for Live Virtual Machine Migration in Cloud Computing". In : *IEEE Transactions on Network and Service Management* 10.4 (déc. 2013), p. 369–382 (Cité en pages 25, 41).

[Cas+14] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson et Frédéric Suter. "Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms". In : *Journal of Parallel and Distributed Computing* 74.10 (juin 2014), p. 2899–2917 (Cité en pages 68, 89).

[KMH14] Vincent Kherbache, Eric Madelaine et Fabien Hermenier. "Planning Live-Migrations to Prepare Servers for Maintenance." In : *Euro-Par : Parallel Processing - Workshops*. Sous la dir. de Luís Lopes et al. T. 8806. Lecture Notes in Computer Science. Springer, août 2014, p. 498–507 (Cité en pages 4, 7, 88, 108).

[Mer14] Dirk Merkel. "Docker : lightweight linux containers for consistent development and deployment". In : *Linux Journal* 2014.239 (2014), p. 2 (Cité en page 110).

[VBJ14] A. Verma, J. Bagrodia et V. Jaiswal. "Virtual Machine Consolidation in the Wild". In : *Middleware'14*. Bordeaux, France : ACM, 2014 (Cité en page 3).

[Zhe+14] Jie Zheng, Tze Sing Eugene Ng, Kunwadee Sripanidkulchai et Zhaolei Liu. "COMMA : Coordinating the migration of multi-tier applications". In : *ACM SIGPLAN Notices*. T. 49. 7. ACM. 2014, p. 153–164 (Cité en pages 42, 74).

[DHK15] Sophie Demassey, Fabien Hermenier et Vincent Kherbache. "Optimized Packings with Applications". In : sous la dir. de Giorgio Fasano et D. János Pintér. Cham : Springer International Publishing, 2015. Chap. Dynamic Packing with Side Constraints for Datacenter Resource Management, p. 19–35 (Cité en page 7).

[KMH15a] Vincent Kherbache, Eric Madelaine et Fabien Hermenier. *Memory and Network Aware Scheduling of Virtual Machine Migrations.* EuroSys'15. Poster. Avr. 2015 (Cité en page 7).

[KMH15b] Vincent Kherbache, Eric Madelaine et Fabien Hermenier. "Ordonnancement contrôlé de migrations à chaud". In : *Compas'15*. Lille, France, juil. 2015 (Cité en page 7).

[KMH15c] Vincent KHERBACHE, Eric MADELAINE et Fabien HERMENIER. "Scheduling Live-Migrations for Fast, Adaptable and Energy-Efficient Relocation Operations." In : *UCC'15*. Sous la dir. d'Ioan RAICU, Omer F. RANA et Rajkumar BUYYA. Limassol, Chypres : IEEE, déc. 2015, p. 205–216 (Cité en pages 7, 72, 101).

[Woo+15] Timothy WOOD et al. "CloudNet : Dynamic Pooling of Cloud Resources by Live WAN Migration of Virtual Machines." In : *IEEE/ACM Trans. Netw.* 23.5 (2015), p. 1568–1583 (Cité en page 20).

[Mic+16] Étienne MICHON, Julien GOSSA, Stéphane GENAUD, Léo UNBEKANDT et Vincent KHERBACHE. "Schlouder : A broker for IaaS clouds". In : *Future Generation Computer Systems* (2016) (Cité en page 7).

[Doca] *Amazon EC2 Container Service (ECS)*. URL : `https://aws.amazon.com/ecs` (Cité en page 111).

[Bol+] Raphaël BOLZE, Franck CAPPELLO, Michel CARON Daydé et *al.* "Grid'5000 : A Large Scale And Highly Reconfigurable Experimental Grid Testbed". In : *Int. Journal of High Performance Computing Applications* () (Cité en page 80).

[Swa] *Docker Swarm : Native Docker clustering*. URL : `https://docs.docker.com/swarm/overview` (Cité en page 110).

[Docb] *Google Container Engine*. URL : `https://cloud.google.com/container-engine` (Cité en page 111).

[Hyp] *Hyper-V Network Virtualization Overview*. URL : `https://technet.microsoft.com/en-us/library/jj134230(v=ws.11).aspx` (Cité en page 35).

[Kub] *Kubernetes : Production-Grade Container Orchestration*. URL : `http://kubernetes.io` (Cité en page 110).

[Docc] *Microsoft Azure Container Service*. URL : `https://azure.microsoft.com/en-us/services/container-service` (Cité en page 111).

[Ope] *OpenStack Docs : Network Design*. URL : `http://docs.openstack.org/ops-guide/arch_network_design.html` (Cité en page 35).

[Dc4] *Projet européen DC4Cities (FP7-ICT-2013.6.2)*. URL : `http://www.dc4cities.eu` (Cité en page 96).

[Vmw] *VMware vSphere Documentation Center : vMotion Networking Requirements.* URL : https://pubs.vmware.com/vsphere-51/index.jsp#com.vmware.vsphere.vcenterhost.doc/GUID-3B41119A-1276-404B-8BFB-A32409052449.html (Cité en page 35).

Liste des abréviations

API Application Programming Interface. 64, 73, 74, 106

BMC Baseboard Management Controller. 94

CBPS Content-Based Page Sharing. 39

CDF Cumulative Distribution Function. 85, 88, 90–92

COMMA Coordinating the Migration of Multi-tier Applications. 42, 43, 74, 75

CPU Computing Processing Unit. 3, 12, 21, 37, 38, 41, 48, 49, 80, 89, 90, 94, 98

CSP Constraint Satisfaction Problem. 59, 64, 65

EWMA Exponential Weighted Moving Average. 25

FC Fiber Channel. 22

FICON Fibre Connection. 22

GNU GNU's Not Unix. 8

HPC High Performance Computing. 36

HTTP Hypertext Transfer Protocol. 51–53

iSCSI Internet Small Computer System Interface. 21, 22

ISO International Organization for Standardization. 76

JSON JavaScript Object Notation. 64

KVM Kernel-based Virtual Machine. 17–20, 25, 26, 39–43, 47–50, 52, 53, 55, 69, 80, 84, 105, 110

LACP Link Aggregation Control Protocol. 108

LGPL Lesser General Public License. 8

LXC Linux Container. 110

MB Memory Buddies. 84–89, 93, 103, 104

MRCPSP Multi-mode Resource-Constrained Project Scheduling Problem. 60, 61, 99

NFS Network File System. 21, 22, 80

OSPF Open Shortest Path First. 108

PPC Programmation Par Contraintes (Constraint Programming (CP)). 59, 64, 65, 106

PSE Page Size Extension. 17

QMP Qemu Monitoring Protocol. 50

RCPSP Resource-Constrained Project Scheduling Problem. 59, 60, 71, 98

REST Representational State Transfer. 64

RIP Routing Information Protocol. 108

RP Reconfiguration Problem. 65, 66, 74, 76

SAS Serial Attached SCSI (Small Computer System Interface). 22

SLA Service-Level Agreement. 19, 70

SLI Software-Led Infrastructure. 2, 3

SMB Server Message Block. 22

TCP Transmission Control Protocol. 12

ToR Top of Rack. 32, 33, 58, 80, 98

VC Virtual Cluster. 36, 37, 42

VDI Virtual Desktop Infrastructure. 111

VLAN Virtual Local Area Network. 108

VM Virtual Machine. vii, 2–6, 8, 11–21, 23–27, 29–32, 34–44, 47–58, 60, 63–65, 68–78, 80–84, 86, 88–94, 97–103, 105–112