UNIVERSITÉ DU QUÉBEC À MONTRÉAL

OPTIMISATION DU PLACEMENT ET DE LA MIGRATION DES RÉPLIQUES DE DONNÉES DANS LES RÉSEAUX DE DIFFUSION DE CONTENU BASÉS SUR LE CLOUD

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUES

PAR

AMINA MSEDDI

AOÛT 2017

REMERCIEMENTS

Mes premiers remerciements s'adressent à Professeur Halima Elbiaze, ma directrice de recherche pour m'avoir encadrée et guidée tout au long de ma maitrise ainsi que pour l'expérience enrichissante et pleine d'intérêt qu'elle m'a offerte. Je la remercie également pour sa bonne humeur et sa convivialité.

Je tiens particulièrement à remercier Professeur Mohamed Faten Zhani mon co-directeur de recherche pour ses remarques enrichissantes, ses précieux conseils et aides apportés lors des différentes étapes de ma maitrise. Je le remercie aussi pour son esprit ouvert, son franc parler et la confiance qu'il a pu m'accorder durant ces mois.

Je remercie aussi Docteur Mohammad Ali Salahuddin qui m'a énormément aidé dans la réalisation de ce mémoire à travers ses conseils et ses remarques.

Je remercie de tout coeur la Fondation de l'UQAM, pour les bourses d'excellence qui m'ont été attribuées ; ces bourses ont grandement facilité mes études.

Bien sûr, je ne voudrais pas oublier dans mes remerciements mes collègues au sein du TRIME et de l'UQAM avec qui j'ai passé de très bons moments et qui m'ont surtout permis de travailler dans une ambiance de travail relaxante et propice à une bonne concentration.

Je remercie ma famille et mon mari pour leur amour, leur soutien indéfectible et la confiance qu'ils ont toujours placée en moi.

TABLE DES MATIÈRES

LISTE DES TABLEAUX vii

LISTE DES FIGURES ix

RÉSUMÉ xi

INTRODUCTION 1

CHAPITRE I
STRATÉGIES DE PLACEMENT DE CONTENU ET DE MIGRATION DANS LES CCDN 7

1.1 Les réseaux de diffusion de contenu basés sur le cloud (CCDN) 7

1.1.1 Définition 8

1.1.2 Architectures existantes 9

1.1.3 Modèle d'affaire 10

1.1.4 Avantages et objectifs des CCDN 10

1.2 Le placement de données dans les CCDN 11

1.2.1 Stratégies de placement des serveurs périphériques 12

1.2.2 Sélection du contenu à répliquer 12

1.2.3 Placement des répliques de données 13

1.3 Migration des répliques de données dans les systèmes de stockage distribués 15

1.3.1 Migration de blocs de données 15

1.3.2 Migration de données dans les systèmes de stockage dans le cloud 16

1.3.3 Migration des répliques de données 17

1.3.4 Limites des solutions actuelles 21

CHAPITRE II
OPTIMISATION DU PLACEMENT DE CONTENU DANS LES CCDN . 27

2.1 Modèle de placement de contenu 27

2.2 Formulation du problème 29
2.2.1 L'énoncé du problème 30
2.2.2 Les contraintes 35
2.2.3 La fonction objectif 43
2.3 Optimisation par essaims particulaires pour le placement des données 43
2.3.1 Introduction à l'optimisation par essaims particulaires (OEP) 44
2.3.2 Le placement des données basé sur OEP 48
2.4 Evaluation des performances du modèle de placement proposé 51
2.4.1 Description des simulations 52
2.4.2 Les résultats des simulations 56
CHAPITRE III
OPTIMISATION DE LA MIGRATION DES DONNÉES DANS LES SYSTÈMES DE STOCKAGE DISTRIBUÉS 59
3.1 Motivations 59
3.2 Formulation du problème 61
3.2.1 Énoncé du problème 62
3.2.2 Les contraintes du problème 64
3.2.3 La fonction objectif 68
3.3 CRANE : Un plan pour la migration des répliques de données dans les systèmes de stockage distribués 68
3.4 Évaluation des performances 71
3.4.1 Évaluation par simulations 73
3.4.2 Évaluation sur une plateforme réelle 76
3.4.3 Résultats expérimentaux 77
CONCLUSION 83
RÉFÉRENCES 87

LISTE DES TABLEAUX

Tableau Page

1.1 Comparaison de stratégies de migration 20

2.1 Les entrées du problèmes de placement dans les CCDN 32
2.2 Les variables du problème de placement dans les CCDN 34
3.1 Les entrées du problème de la migration des répliques 63

3.2 Les variables du problème de la migration des répliques de données 64

3.3 Capacité des liens entre les centres de données de Amazon EC2 . 72

3.4 Scénarios considérés 74

3.5 Scénarios déployés dans les expérimentations 77

LISTE DES FIGURES

Figure		Page
1.1	Configuration de placement initiale et finale	22
1.2	Séquences de migration des répliques	24
2.1	Architecture hierarchisés de serveurs du CCDN	31
2.2	La topologie de serveurs utilisée pour les simulations	52
2.3	Exemple de la définition d'une vidéo dans le fichier JSON	54
2.4	Comparaison des performances analytiques des stratégies de placement	58
3.1	Comparaison des performances analytiques de la séquence de migration optimale, CRANE et Swift	81
3.2	Comparaison expérimentale des performances entre CRANE et Swift	82

RÉSUMÉ

Le cloud, grâce à son modèle d'approvisionnement élastique des ressources, se révèle être une solution rentable pour les fournisseurs de contenu qui peuvent désormais louer des ressources de stockage, de calcul, et de bande passante, menant à l'apparition de réseaux de diffusion basés sur le cloud. Cependant, la performance de ces réseaux est influencée par le placement du contenu dans les différents centres de données géographiquement distribués. À cet effet, nous proposons une stratégie de placement de contenu dans une infrastructure distribuée et une stratégie pour minimiser leur temps de migration entre les différents centres de données. La stratégie de placement de données est basée sur les relations entre le contenu et vise à rapprocher de l'utilisateur le contenu le plus populaire et corrélé afin de réduire la latence et de minimiser le coût de stockage et de bande passante. Nous avons aussi proposé une heuristique baptisée CRANE qui complète n'importe quelle stratégie de placement en gérant efficacement la création de répliques de données et en minimisant le temps nécessaire pour copier les données vers leurs nouveaux emplacements tout en évitant la congestion du réseau et en assurant la disponibilité minimale des données. Nos résultats expérimentaux montrent que le temps de migration de CRANE est environ 20 % proche du temps mis par la séquence de migration optimale et améliore par 40 % le temps de migration par rapport à Openstack Swift.

INTRODUCTION

Le nombre d'utilisateurs accédant à du contenu à distance a augmenté d'une façon significative au cours des dernières décennies. En conséquence, cette forte demande sur les services de contenu a conduit à des problèmes liés à la disponibilité des ressources et à la baisse de la qualité de l'expérience de l'utilisateur. Les réseaux de diffusion de contenu (CDN) viennent surmonter ces limites. Dans ces réseaux, le contenu est dupliqué et stocké dans des serveurs périphériques distribués autour du monde. Les requêtes des utilisateurs sont redirigées aux serveurs qui conviennent le mieux selon des critères tels que la proximité ou l'état du réseau et des serveurs. De cette manière, les CDN permettent aux utilisateurs d'accéder rapidement au contenu, quel que soit leur emplacement géographique tout en réduisant la latence causée par la distance séparant le serveur d'hébergement de l'utilisateur final.

Cependant, les réseaux de distributions de contenus tels que *Akamai* et *Limelight Networks* se révèlent des solutions coûteuses pour les fournisseurs de contenus (Broberg *et al.*, 2009). Par ailleurs, l'informatique en nuage (également appelé *cloud computing*) permet aux fournisseurs de CDN de louer des ressources pour le stockage, le calcul et la bande passante selon leurs besoins et à la demande. En effet, l'informatique en nuage permet de s'adapter aux changements de la charge de travail en approvisionnant et en retirant des ressources de manière dynamique afin de mettre en correspondance les ressources disponibles à la demande actuelle (Herbst *et al.*, 2013). Cet approvisionnement élastique des ressources s'avère une solution rentable pour les fournisseurs de contenu. Cela a mené à la création des CDN basés sur le cloud appelés Cloud-based CDN (CCDN) . Les CCDN

permettent d'augmenter la disponibilité et de réduire le temps d'accès au contenu, tout en minimisant le coût de stockage des données.

Typiquement, les fournisseurs de l'informatique en nuage déploient plusieurs centres de données à grande échelle dans des sites géographiquement distribués. Ils utilisent ensuite la réplication de données comme une technique efficace pour améliorer la tolérance aux pannes, réduire la latence d'accès aux données et minimiser la quantité de données échangées dans le réseau. Par conséquent, la gestion efficace de ces répliques de données est l'un des principaux défis pour les fournisseurs de l'informatique en nuage. La gestion des données et plus particulièrement les répliques de données comprend leur placement, leur synchronisation et leur migration. Le problème de placement consiste à déterminer les serveurs où les répliques de données doivent être stockées de manière à maximiser la disponibilité des données et à minimiser le coût de communication et le temps d'accès. Cependant, les stratégies de placement de répliques de données induisent la création et la migration d'un grand nombre de copies de données au fil du temps entre et au sein des centres de données.

En outre, certaines recherchent démontrent que le contenu généré par les utilisateurs et les vidéos générées à partir de la télévision sur Internet et de la vidéo sur demande constituent le plus grand volume de données qui sont échangés sur Internet dans la prochaine décennie (Cisco, 2012). Ces données sont caractérisées par leurs changement d'accès continue et la variation de leur popularité au cours du temps. Des applications et des réseaux sociaux en ligne sont généralement proposés pour partager ce type de contenu. De plus, ces derniers proposent des options tels que la recommandation de contenu associé et la sélection de contenu à afficher suite à la consultation d'un contenu. Cependant, la suggestion de contenu corrélé à un autre a un impact sur sa popularité et la fréquence de ses accès (Zhou *et al.*, 2015).

Étant donné que le contenu change au fil du temps, la popularité des données hébergées par le CCDN change aussi. Une donnée devenant plus populaire à un moment donné nécessite plus de répliques qu'une autre. De plus, ces répliques devraient être placées dans les centres de données de manière à minimiser le temps nécessaire pour y accéder. Inversement, si un contenu perd de sa popularité, cela devrait engendrer une réduction du nombre de répliques afin de minimiser les coûts de placement. Ainsi, ce changement dynamique en fonction de la popularité et les besoins en performance va engendrer des transferts continus d'une importante quantité de données entre les centres de données. Plus le nombre de répliques qui devraient être créées est élevé, plus le trafic généré entre les différents centres de données est important. Cet échange croissant de quantités énormes de contenu entre les centres de données peut surcharger le réseau. Cela pourrait aussi nuire à la performance globale du réseau en termes de perte de paquets et de latence. Cette congestion ralentit aussi l'arrivée des répliques à leur destination finale. Par ailleurs, un transfert massif de données engendre plusieurs conséquences. D'abord, une consommation élevée des ressources telles que l'unité centrale de traitement, la mémoire, les entrées/sorties des disques au niveau de la machine source et celle de destination peut ralentir les autres tâches en cours d'exécution sur ces machines. De plus, cette migration massive des données peut congestionner le réseau. En effet, le processus de migration de données dans les systèmes de stockage distribué est un processus pair à pair, asynchrone. Il n'y a généralement pas de coordination pour la copie des données. Chaque machine dans le système vérifie pour chaque donnée si cette dernière est présente dans les autres noeuds où elle devrait être stockée. Si celle-ci n'y est pas, la machine copie cette donnée sans vérifier si un autre noeud est entrain d'effectuer cette copie. Ceci peut engendrer la migration redondante et inutile de répliques. Finalement, la disponibilité des répliques peut être affectée. En effet, quand une réplique de donnée est créée/migrée dans un nouvel emplacement, elle ne sera disponible que lorsque tout son contenu est

copié dans le noeud de destination. Si la création de répliques prend un temps élevé, la disponibilité de données peut être affectée si le nombre de répliques de données actives ne suffit pas à satisfaire toutes les demandes des utilisateurs.

L'objectif de ce mémoire est de proposer des solutions qui permettent une meilleure gestion du placement et de la migration des répliques dans les CCDN. Cet objectif peut être atteint en minimisant la quantité de données transférée entre les différents centres de données d'un CCDN. Nous pouvons aussi réduire le nombre de répliques qui doivent être stockées dans de nouveaux centres de données. Pour ce faire, nous considérons les relations entre les différents contenus telles que la popularité et la corrélation pour déterminer le placement dans les serveurs périphériques du CCDN. De plus en ordonnant la migration de ces répliques entre les différents centres de données, nous pourrions respecter l'accord de service (SLA) relatif au délai de réponse pour les requêtes des clients. Par ailleurs, le choix de la machine source et des liens à partir desquels la donnée doit être copiée et l'ordre dans lequel les répliques sont créées minimise la congestion du réseau et accélère le transfert de données.

Dans ce mémoire, nous modélisons les problèmes de placement des répliques de données et celui de leur migration à l'aide de la programmation linéaire en nombres entiers. Ensuite, nous concevons et implémentons deux algorithmes heuristiques pour résoudre ces problèmes. La première solution se base sur l'optimisation par essaims particulaires (OEP) pour le placement des répliques de données dans différents centres de données d'un CCDN. Cette solution vise à trouver le bon placement pour les répliques dans le but de minimiser leur coût de stockage et de bande passante nécessaire pour leurs copies et de rapprocher le contenu le plus populaire et corrélé des utilisateurs pour leur offrir un meilleur délai de réponse.

Nous proposons aussi une seconde solution (Mseddi *et al.*, 2015) (baptisée

CRANE) permettant de planifier la migration des répliques de données dans un système de stockage distribué. CRANE gère la création de répliques dans des infrastructures géodistribuées dans le but de (1) réduire au minimum le temps nécessaire pour la copie des répliques vers leurs nouveaux emplacements, (2) éviter la congestion du réseau, et (3) assurer une disponibilité minimale pour chaque réplique.

Ce document est divisé en quatre chapitres.

Le premier chapitre fournit un aperçu des concepts de base des CDN. Ensuite, nous décrivons des stratégies de placement de contenu. Nous dressons ensuite l'état de l'art de la migration des répliques de données dans les systèmes de stockage distribués.

Dans le chapitre 2, nous formulons le problème du placement des répliques de contenus dans une architecture hiérarchisée de CCDN comme un programme linéaire entier. Nous décrivons ensuite notre solution basée sur l'optimisation par essaims particulaires (OEP). Finalement, nous évaluons les performances de trois stratégies de placement de contenu par rapport aux performances de notre solution et de la solution optimale.

Le chapitre 3 est consacré à l'optimisation de la migration des répliques des données dans les systèmes de stockages distribués. Nous commençons par formuler le problème de migration de répliques comme un programme linéaire en nombre entier (PLNE). Ensuite, nous présentons notre solution heuristique baptisée CRANE ainsi que les résultats expérimentaux. Nous avons implémenté CRANE et nous l'avons intégré dans OpenStack. Nous utilisons cette mise en œuvre pour exécuter des expériences qui montrent la performance de CRANE sur OpenStack Swift pour différents scénarios. Nous comparons également la solution trouvée par CRANE avec la solution optimale trouvée par la résolution du PLNE associé.

Finalement, le dernier chapitre de ce mémoire est dédié à la conclusion.

CHAPITRE I

STRATÉGIES DE PLACEMENT DE CONTENU ET DE MIGRATION DANS LES CCDN

Dans ce chapitre, nous commençons par présenter les concepts des réseaux de diffusion de contenu (CDN) traditionnels et ceux qui sont basés sur le cloud. Ensuite, nous présentons les paramètres influant sur le placement des répliques dans ce type de réseaux et nous établissons l'état de l'art des stratégies proposées dans la littérature. Nous présentons ensuite les différentes stratégies de migration de données dans les systèmes de stockage distribué.

1.1 Les réseaux de diffusion de contenu basés sur le cloud (CCDN)

Le Web, étant devenu le moyen de communication incontournable pour le partage du contenu, a contribué à la croissance rapide de l'internet. Le nombre d'utilisateurs accédant au contenu Web a aussi augmenté d'une manière exponentielle engendrant une forte demande pour les systèmes hébergeant du contenu. En conséquence, cette forte demande sur les services de contenu a engendré des problèmes de disponibilité de ressources et une baisse de performance. Les réseaux de diffusion de contenu ont été proposés pour surmonter ces limitations en offrant des infrastructures et des mécanismes pour livrer du contenu d'une manière évolutive, et améliorant l'expérience des utilisateurs du Web. D'autre part, le cloud, grâce à son modèle d'approvisionnement élastique

des ressources, révèle être une solution rentable pour les fournisseurs de contenu. Ces fournisseurs qui peuvent désormais louer les ressources du cloud (stockage, calcul, et bande passante) pour construire des CDN basés sur le cloud appelés CCDN.

1.1.1 Définition

Un réseau de diffusion de contenu (CDN) est une collection d'éléments de réseau permettant une distribution plus efficace de contenu aux utilisateurs finaux. Un CDN se compose de deux types de serveurs. Des serveurs d'origine qui servent à introduire les contenus dans le réseau, et des serveurs périphériques géographiquement distribués pour répliquer les contenus des serveurs d'origine. Lorsqu'un utilisateur demande une donnée qui fait partie d'un réseau de diffusion de contenu, ce dernier va rediriger la demande à partir du serveur du site d'origine à un serveur dans le réseau qui est le plus proche de l'utilisateur et lui distribue le contenu mis en cache. Le CDN communique également avec le serveur d'origine pour livrer un contenu qui n'a pas été précédemment mis en cache.

La construction de l'infrastructure de CDN est axée de manière à offrir une panoplie de services et de fonctionnalités tels que le stockage et la gestion de contenu, la distribution de contenu entre les serveurs périphériques, la gestion de la cache, la transmission de contenu statique, dynamique et de diffusion en continu (*streaming*), des solutions de sauvegarde et de récupération après les sinistres, la surveillance, la mesure de performance, et les suivis. Pour ce faire, cette infrastructure est composée essentiellement des éléments suivants :

— *Les composantes de diffusion de contenu* comprennent un serveur d'origine et un ensemble de serveurs périphériques qui fournissent des copies de contenu aux utilisateurs finaux ;
— *La composante de redirection* qui est chargé de diriger les requêtes des

clients vers le serveur périphérique le plus approprié (le plus proche ou le moins chargé). Ce composant doit aussi communiquer avec le composant de distribution afin de maintenir une vue à jour du contenu stocké dans les caches CDN ;

— *La composante de distribution* qui copie le contenu à partir du serveur d'origine vers les serveurs périphériques et assure la cohérence du contenu dans les caches ;

— *La composante de facturation* qui enregistre les journaux (*logs*) d'accès des clients et l'utilisation des serveurs CDN. Ces données sont utilisées pour la facturation et la maintenance.

1.1.2 Architectures existantes

Au fil des dernières années, plusieurs travaux de recherche ont été élaborés pour étudier les architectures et les modèles de CCDN. Chen et al. ont étudié le problème de construction des chemins de distribution et le problème du placement des serveurs périphériques conjointement dans les CCDN afin de minimiser les coûts et satisfaire la qualité de service relative aux requêtes des utilisateurs (Chen *et al.*, 2012). D'autre part, Srinivasan et al. ont proposé l'architecture "activeCDN" qui permet à un fournisseur de contenu de mettre à l'échelle dynamiquement les services de diffusion de contenu en utilisant la virtualisation des réseaux et les techniques de cloud computing (Srinivasan *et al.*, 2011). Lin et al. ont proposé une architecture CCDN se basant sur le système Hadoop ayant pour but de minimiser les coûts des CDN et faciliter la gestion de contenu pour les éditeurs de contenu (Lin *et al.*, 2011) . Papagianni et al. se sont intéressé à une infrastructure constituée de plusieurs fournisseurs de cloud (Papagianni *et al.*, 2013). Ils ont proposé et évalué une architecture hiérarchique de CCDN où les ressources en communication inter et intra cloud sont simultanément prises en compte, tout en considérant les ressources traditionnelles du cloud. Finalement,

Wang et al. ont identifié les défis et enjeux et établi un état de l'art des CCDN (Wang *et al.*, 2015).

1.1.3 Modèle d'affaire

Traditionnellement, un fournisseur de contenu (consommateur) peut s'engager avec un fournisseur de services CDN pour livrer son contenu aux utilisateurs finaux. Généralement, les CDN hébergent du contenu d'une tierce partie. Ce contenu peut être statique (pages HTML statiques, images, documents...), des médias pour la diffusion continue (audio, vidéo en temps réel), des vidéos générées par les utilisateurs et des contenus de services (commerce électronique, service de transfert de fichiers). D'autre part, leS fournisseurS de serviceS Cloud CDN peuvent soit posséder tous les services qu'ils utilisent pour exécuter leurs services CDN ou peuvent déléguer ces services à un fournisseur de cloud. Finalement, les utilisateurs finaux interagissent avec le CDN en spécifiant le contenu au travers de leurs téléphones intelligents, ou ordinateurs.

1.1.4 Avantages et objectifs des CCDN

Les CCDN offrent plusieurs avantages (Wang *et al.*, 2015) :

- *Le modèle de paiement des CCDN* : CCDN offre aux utilisateurs un service de diffusion de contenu en utilisant un modèle de paiement au fur et à mesure de l'utilisation (*pay-as-you-go*).
- *Augmentation des points de présence :* Le contenu est placé dans des serveurs plus proches des utilisateurs dans les CCDN comparé aux CDN traditionnels en raison de l'omniprésence des centres de données de cloud. Le CCDN peut réduire la latence de transmission grâce à la possibilité de louer sur demande des ressources pour augmenter la disponbilité et la visibilité du contenu.

— *L'accessibilité du CCDN :* Les CDN basés sur le cloud permettent aux fournisseurs de contenu d'atteindre de nouveaux marchés et de nouvelles régions et de soutenir les utilisateurs nomades. Par exemple, au lieu de mettre en place une infrastructure pour servir un petit groupe de clients en Afrique, les fournisseurs de CDN profitent des fournisseurs de cloud actuels dans la région pour héberger les serveurs périphériques d'une manière dynamique.

— *Plus grande variété d'applications CCDN :* Grâce à son support des changements dynamiques de la charge, le cloud facilite aux CDN de soutenir différents types d'applications dont le trafic augmente d'une manière imprévisible, ou qui requièrent une quantité de ressources variable au cours du temps. Le cloud leur permet aussi d'évoluer et de croître rapidement.

Les principaux objectifs (Wang *et al.*, 2015) du CCDN sont :

— Fournir une grande évolutivité et une infrastructure de cloud d'une haute stabilité.

— Réduire le coût de la communication entre les serveurs périphériques et les services fournis par l'infrastructure de cloud.

— Gérer plus facilement le contenu

— Fournir le contenu populaire au serveur le plus proche pour des utilisateurs finaux.

1.2 Le placement de données dans les CCDN

Dans les CCDN, le problème de placement des répliques se résume à trouver le meilleur ensemble de serveurs pour placer les répliques/copies de contenu. La performance des CDN dépend de plusieurs paramètres tels que le nombre de serveurs périphériques requis, leur emplacement, le modèle de calcul du coût adopté (par exemple les coûts de stockage, de la récupération, et de la mise à jour), et la considération de contraintes liées à la qualité de service.

1.2.1 Stratégies de placement des serveurs périphériques

Le choix du meilleur emplacement pour les serveurs de substitution a un impact direct sur les performances du processus de diffusion de contenu. L'objectif du placement optimal des serveurs de substitution est de réduire la latence perçue par l'utilisateur pour accéder au contenu et de minimiser la consommation globale de la bande passante réseau pour transférer le contenu répliqué des serveurs jusqu'aux clients. Plusieurs heuristiques visant à trouver le meilleur emplacement pour les serveurs de substitution ont été développées. Parmi ces heuristiques, on compte "le placement glouton" (*Greedy replica placement*) (Krishnan *et al.*, 2000) , "le placement avec connaissance de la topologie" (*Topology-informed placement strategy*) (Jamin *et al.*, 2001) et "le placement en arbre" (*Tree-based replica placement*) (Li *et al.*, 1999). Ils prennent en considération des informations existantes du CDN tels que la topologie du réseau et les modèles de variation de la demande. Ces algorithmes offrent des solutions qui visent aussi à minimiser les coûts. Dans ce travail, nous avons choisi le placement en arbre pour le placement des serveurs de substitution. Cette stratégie suppose que la topologie sous-jacente est sous la forme d'un arbre de serveurs.

1.2.2 Sélection du contenu à répliquer

L'efficacité de la distribution du contenu repose en grande partie sur la sélection du contenu à transmettre aux utilisateurs finaux. Une approche appropriée de sélection de contenu peut réduire le temps de téléchargement du contenu par les clients et la charge des serveurs. Il existe deux approches principales pour la sélection de contenu. La première consiste à copier tous les objets du serveur d'origine aux serveurs de substitution. Dans cette approche, c'est le serveur de substitution qui transmet la totalité du site à l'utilisateur final. Dans la seconde approche, quelques objets du site (tels que les vidéos ou les images) sont copiés du

serveur d'origine aux serveurs de substitution. Dans cette approche, la sélection des objets à copier a un impact important sur les performances du CDN. Cette sélection est faite en fonction en 4 critères :

— *Sélection empirique* (Fujita *et al.*, 2004) où l'administrateur choisit empiriquement le contenu à répliquer dans les serveurs périphériques.
— *Sélection basée sur la popularité* (Chen *et al.*, 2003) où le contenu le plus populaire est choisi pour être répliqué.
— *Sélection par objet* (Chen *et al.*, 2003; Wu et Kshemkalyani, 2006) où le choix des objets à répliquer est basé sur une fonction de gain.
— *Sélection par regroupement* (Fink *et al.*, 2002) (Chen *et al.*, 2003) où le contenu à répliquer est regroupé sur la base de la fréquence d'accès ou sur la base de la corrélation. Finalement, ce regroupement de contenu peut aussi être basé sur les sessions des utilisateurs ou sur les URL des contenus.

1.2.3 Placement des répliques de données

Nous considérons le placement de répliques de données dans des serveurs placés suivant une topologie en arbre. Le problème de placement de réplique dans ce type de topologie est NP-complet comme démontré par Benoit et al. (Benoit *et al.*, 2007). Cependant, il existe beaucoup de travaux qui ont proposé des solutions heuristiques pour le placement de contenu dans les systèmes distribués. Étant donné que notre travail concerne le placement statique de contenu en considérant que la distribution géographique des clients et que les tendances (*pattern*) de leurs accès sont connues à l'avance, nous décrivons dans cette section les travaux effectués pour résoudre ce type de problème et considérant d'autres critères tels que la qualité de service, popularité et la corrélation.

Kangasharju et al. ont présenté quatre stratégies de réplication de données dans les CDN permettant de minimiser le nombre de sauts que les données devraient

traverser avant d'atteindre l'utilisateur final (Kangasharju *et al.*, 2002). L'une de ces stratégies proposées considère la popularité des données à répliquer. Ainsi, chaque nœud va stocker les données qui sont les plus populaires chez les clients les plus proches d'eux géographiquement. Dans cette heuristique, les nœuds vont stocker autant de données que leur capacité de stockage le leur permet. Ceci va engendrer naturellement une augmentation du coût de stockage. Dans notre approche, nous tentons aussi de minimiser ce coût. Ainsi, nous plaçons le minimum de copies de donnée tant que celles-ci peuvent répondre aux requêtes des utilisateurs finaux sans enfreindre l'accord de service en termes de temps de réponse. Laoutaris et al. se sont aussi intéressés à minimiser la distance de récupération d'un contenu depuis le serveur où il est stocké jusqu'à l'utilisateur, en tenant compte de la contrainte de la capacité de stockage des serveurs (Laoutaris *et al.*, 2005). Ce travail, comme le nôtre considère une topologie hiérarchique de CDN (en arbre), mais il vise à minimiser le coût de la récupération des données par les utilisateurs finaux, alors que dans notre cas, nous visons à minimiser le coût de placement et de bande passante utilisée lors de la réplication des données. L'étude faite dans (Tewari et Kleinrock, 2006) a complété ces résultats pour montrer que la distribution selon la popularité a des avantages au niveau des performances du réseau. Nous voudrions montrer dans notre travail que la distribution selon la popularité jointe à celle de la corrélation entre les données peut avoir un meilleur impact sur les performances du réseau et sur le temps d'accès des utilisateurs finaux.

Cependant en ce qui concerne la corrélation entre les données, celle-ci a été exploitée dans le but de grouper les données à répliquer ensemble (Fink *et al.*, 2002; Chen *et al.*, 2003). Ce regroupement est effectué de deux façons : selon les groupes d'objets consultés dans les sessions précédentes des utilisateurs ou selon l'adresse URL des objets. Cependant, nous ne visons pas à grouper ces

données corrélées en groupes. Mais nous traitons chaque contenu d'une manière indépendante et nous tentons de minimiser la distance entre ces données corrélées tout en minimisant le coût de stockage et de bande passante. Nous visons à travers ceci à minimiser le nombre de répliques de données qui doivent être relocalisées, étant donnée la ressemblance dans les patterns des données corrélées (Tu et Yen, 2014).

1.3 Migration des répliques de données dans les systèmes de stockage distribués

La migration de données dans les infrastructures de CCDN est un problème connexe au problème de migration de données dans les systèmes de stockage distribués. En effet, un grand nombre de travaux a été consacré à la migration des données dans les systèmes de stockage distribués. Étant donné que notre travail est étroitement lié à la gestion des répliques de données dans les CCDN, nous présentons d'abord les travaux les plus importants traitant la migration des données dans les systèmes de stockage distribués, puis nous décrirons les travaux s'intéressant à la migration des données répliquées.

1.3.1 Migration de blocs de données

Le problème de la migration des données consiste à trouver un plan efficace pour déplacer des données très volumineuses entre les infrastructures. Le problème basique de la migration est évoqué lorsque tous les périphériques de stockage ont un lien direct et une capacité égale, les objets de données sont de même taille et ont une seule copie, et tous les périphériques de stockage ne peuvent migrer qu'un seul objet de données à la fois. Kim et al. ont démontré que ce problème est NP-difficile à partir d'une réduction du problème de la coloration des arêtes (Kim, 2005). Il est ensuite possible de résoudre le problème à l'aide des algorithmes de coloration des arêtes (*edge-coloring*) dans les multigraphes pour

obtenir un plan efficace pour la migration de données (Hall *et al.*, 2001; Anderson *et al.*, 2010; Kari *et al.*, 2011). L'idée est de modéliser les migrations entre différentes infrastructures comme les arêtes d'un graphe dont les nœuds sont la source et la destination de la migration. Les arêtes de même couleur représentent les migrations qui peuvent être programmées simultanément. Le problème de la coloration des arêtes est un problème NP-complet (Holyer, 1981).

Certains travaux ont examiné des cas particuliers du problème basique de la migration des données. Par exemple, Kari et al. ont proposé une heuristique qui vise à minimiser le temps de migration total tout en tenant compte de la capacité de transferts simultanés des nœuds de stockage (Kari *et al.*, 2011). Cependant, leur solution ne considère pas les contraintes liées au réseau telles que les limites de la bande passante et les délais de propagation.

1.3.2 Migration de données dans les systèmes de stockage dans le cloud

Plusieurs travaux ont été proposés pour optimiser la migration des données dans les systèmes géo-distants de stockage dans le cloud. Dans (Wu *et al.*, 2015) les auteurs ont proposé un service qui ordonne dynamiquement les demandes de migration au fur et à mesure de leur apparition dans les centres de données géo-distants. Cet ordonnancement est dicté par les niveaux d'urgence des données nécessitant des délais de transfert de données différents. Ces demandes de migration sont ordonnées de façon optimale et dynamique de manière à exploiter pleinement la bande passante disponible. Par conséquent, dans notre travail, nous traitons le cas où les données possèdent de multiples copies, nous permettant de choisir la source de la migration afin d'exploiter pleinement la bande passante disponible dans le réseau reliant différents centres de données.

Petrol et al. ont proposé un plan d'optimisation de la migration de données visant à accélérer l'extension du stockage dans le cloud (Petrov et Tatarinov, 2009). Pour

cela, ils ont considéré deux types de migrations, celles pour la mise en échelle et les résiduelles. En effet, les migrations de mise en échelle concernent le transfert de données vers des nœuds adjacents et les migrations résiduelles concernent le transfert de données vers des nœuds plus éloignés.

1.3.3 Migration des répliques de données

Le problème de migration des répliques de données se distingue du problème basique de la migration par le fait que pour chaque donnée à créer dans un nouvel emplacement, il existe plusieurs sources de copie potentielles. Le problème réside donc à trouver la meilleure source de copie afin d'atteindre certains objectifs tels que la minimisation du temps de la migration ou le coût de la migration. Ce problème a été démontré comme NP-complet par (Tziritas *et al.*, 2008)

Par ailleurs, S. Khuller et al. ont comparé le problème de la migration des copies de données au problème de "gossiping and broadcasting" et ont proposé plusieurs algorithmes d'approximation visant à minimiser le temps de migration des copies de données (Khuller *et al.*, 2003). D'autre part, N. Tziritas et al. ont employé deux critères de sélection des copies à transférer : le temps de début le plus proche et le temps de fin de migration le plus proche (Tziritas *et al.*, 2008). Ces temps sont calculés sur la base des transferts qui ont été programmés et la bande passante disponible.

T. Loukopoulos et al. visent à minimiser la consommation de la bande passante durant la migration (Loukopoulos *et al.*, 2007; Loukopoulos *et al.*, 2006). Cependant, ils considèrent que les liens reliant les différents nœuds ont la même capacité. Les auteurs de (Tziritas *et al.*, 2009) ont étudié le cas où le même objet doit être migré vers de multiples destinations. Ils utilisent par ailleurs le meilleur arbre de multidiffusion (arbre de Steiner) pour migrer les objets vers leurs destinations finales. Cependant, dans cette solution les nœuds intermédiaires

peuvent commencer à transférer un objet même s'ils ne le stockent pas en entier.

Openstack est le plus important projet libre pour la création d'infrastructures de cloud privés et publics. Il représente un ensemble de logiciels complémentaires permettant le contrôle de grandes ressources de calcul, de stockage et de réseau. Swift, est le projet d'Openstack pour la gestion du stockage de données (foundation, 2015). Dans son processus de gestion, Swift considère des groupes de données appelés partitions. Ces partitions sont obtenues à partir de similitudes dans la clé du nom des données et sont répliquées 3 fois par défaut. L'algorithme de placement de données dans Swift assigne les copies d'une même partition dans une infrastructure distribuée de manière à ce qu'elles soient le plus éloignées les unes des autres (Dickinson, 2013) afin de leur garantir une haute disponibilité. Le placement optimal des données dans Swift est calculé périodiquement (une fois par heure) ou à chaque fois qu'un changement dans l'infrastructure survient. A l'issue de ce calcul, le processus de migration est directement déclenché.

Contrairement aux travaux cités précédemment, CRANE prend en considération la bande passante disponible dans les liens et la disponibilité des données pendant la migration. Notre heuristique exploite l'existence de plusieurs copies de données à travers le réseau ce qui lui permet de choisir la source de la migration et le chemin de transmission afin d'éviter la congestion du réseau et de réduire le temps de migration de données.

Le tableau 1.1 compare les travaux étudiés dans le cadre de la migration des copies de données. Les auteurs de ces travaux ont choisi différentes hypothèses par rapport à la taille des données, la capacité des liens connectant les différents nœuds et l'utilisation de nœuds intermédiaires au cours de la migration. D'autre part deux principaux objectifs divisent les travaux présentés dans le tableau 1.1 : la minimisation du temps de migration et la minimisation du coût de la

bande passante pendant la migration. Cependant CRANE est le seul qui prend en considération la disponibilité des donnée pendant le processus de migration tout en minimisant le temps de migration et évitant la congestion dans le réseau.

Tableau 1.1: Comparaison de stratégies de migration

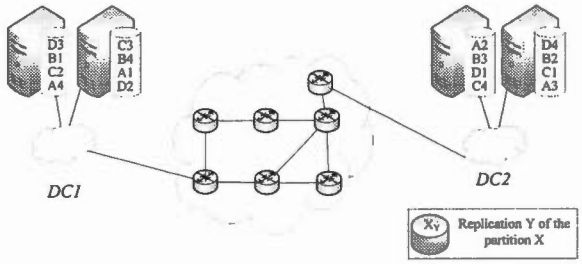
	(Khuller *et al.*, 2003)	(Tziritas *et al.*, 2008)	(Loukopoulos *et al.*, 2007)	(Loukopoulos *et al.*, 2006)	(Tziritas *et al.*, 2009)	Swift (foundation, 2015)	CRANE
Hypothèses							
Taille des objets	Egale	Différent	Egale	Différente	Différente	Différente	Egale
Liens	Capacités égales	Différentes capacités	Capacités égales mais différent coût	Capacités égales mais différent coût	Différentes capacités	Capacités différentes et liens chevauchants	Capacités différentes et liens chevauchants
Utilisation de nœuds intermédiaires	Oui	Non	Non	Non	Non	Non	Non
Objectifs							
Minimiser le temps de migration	Oui	Oui	Non	Non	Oui	Non	Oui
Minimiser le coût de la migration	Non	Non	Oui	Oui	Non	Non	Non
Assurer une disponibilité minimale	Non	Non	Non	Non	Non	Oui	Oui

1.3.4 Limites des solutions actuelles

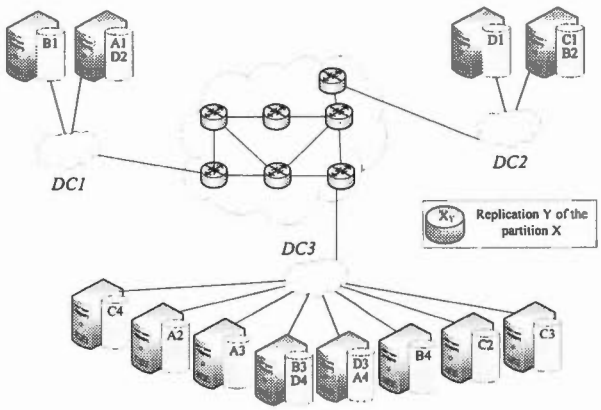
Nous décrivons dans cette section un exemple expliquant les limites des solutions décrites. Nous considérons une infrastructure de cloud composée de deux centres de données géographiquement distants (DC1 et DC2). Différentes capacités de stockages sont disponibles dans chaque centre de données. Ces derniers sont connectés avec des liens de différentes capacités. Ce déploiement de cloud utilise Swift comme solution distribuée pour la gestion du stockage de données. Nous considérons un scénario où 4 partitions A,B,C et D de tailles respectives 300 GB, 100 GB, 500 GB et 200 GB sont répliqués 4 fois et stockés dans les nœuds des deux centres de données. La figure 1.1(a) illustre le placement initial des répliques.

À l'ajout d'un nouveau centre de données DC3, les copies des partitions doivent être replacés suivant l'algorithme "as-unique-as-possible", qui vise à éloigner les répliques les unes des autres afin d'augmenter leur disponibilité (Dickinson, 2013). Ce placement respecte aussi la capacité des disques dans les différents nœuds des centres de données.

Afin d'atteindre la nouvelle configuration du placement de ces répliques, d'énormes volumes de données doivent être déplacés. Dès que le nouveau placement des copies a été calculé, tous les nœuds gérés par Swift vont commencer à suivre ce nouveau placement. Ainsi, les composantes responsables de la redirection des requêtes des clients vont rediriger ces dernières à des copies de données, qui parfois, ne sont pas encore arrivées à leurs destinations optimales. Ceci va aboutir à une indisponibilité des données. En effet, l'algorithme de Swift requiert que la majorité des copies répondent à une requête afin de considérer que celle-ci a abouti. Pour éviter cette indisponibilité, l'algorithme de placement de Swift ne permet le repositionnement que d'une copie d'une même donnée pendant l'intervalle spécifié. Dans notre scénario, nous gardons l'intervalle par défaut de une heure. Ainsi l'algorithme



(a) Configuration de placement initiale des répliques



(b) Configuration de placement finale des répliques

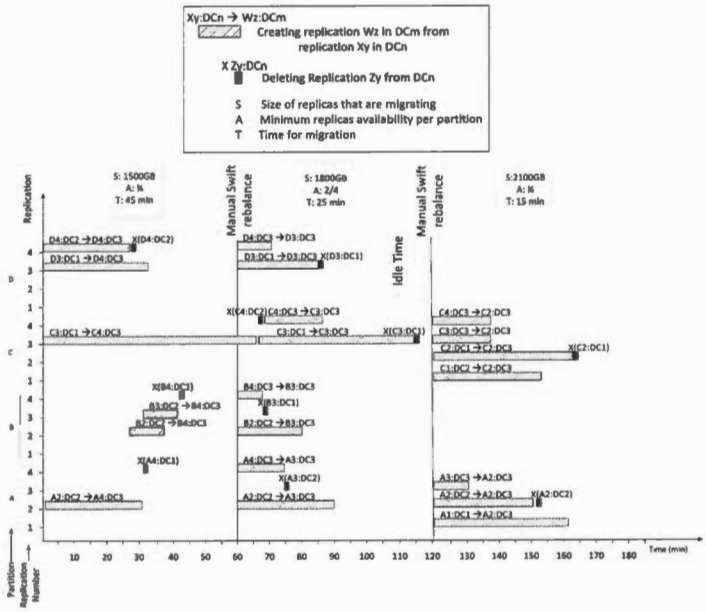
Figure 1.1 Configuration de placement initiale et finale

de placement des copies de données ne replacera pas deux copies durant le même intervalle d'une heure. Il est important de noter que le recalcule de placement des copies de données n'est pas effectué automatiquement après l'écoulement d'une heure. Cependant, un administrateur devra exécuter les commandes pour calculer les nouveaux emplacements des copies de données et c'est là où le processus de migration commencera aussi.

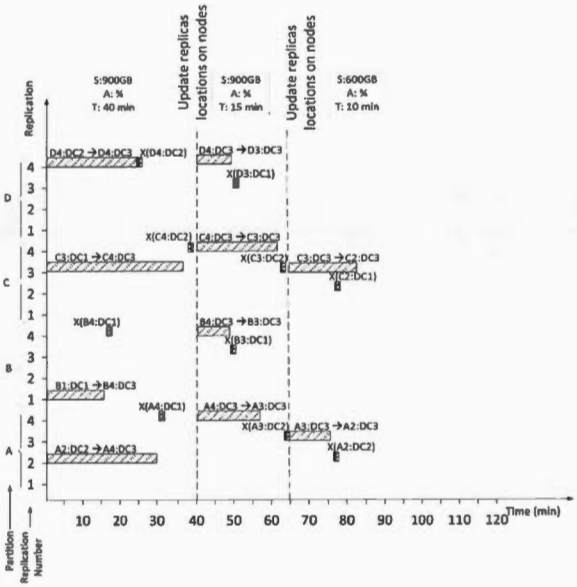
De plus, dans ce scénario, la disponibilité minimale pour une partition est de 3/4. Cette disponibilité est spécifié par Swift pour que, à chaque requête, 3 des 4 copies d'une même partition doivent être disponibles en tout temps. La figure 1.1(b) montre le placement optimal des copies des partitions selon les capacités

des disques et de l'algorithme "*as-unique-as-possible*". Cette configuration finale est atteinte après une suite de calculs du placement des copies des partitions et de migrations.

La migration des copies de partitions dans Swift est un processus pair-à-pair (*peer to peer*) asynchrone. Ainsi, chaque nœud de stockage va séquentiellement parcourir toutes les copies de partitions qu'il stocke et va les comparer à celles des nœuds qui devraient aussi stocker des copies de ces partitions. Le processus de migration utilise un modèle de poussée. En effet, à chaque fois qu'une copie doit être présente dans un nœud et qu'elle ne l'est pas, celle-ci sera copiée à partir du nœud qui a fait la comparaison vers le nœud distant. Et si un nœud possède une copie d'une partition qu'il ne devrait pas stocker, il la supprime. La figure 1.2(a) présente un exemple de la séquence de migration des copies des partitions. Dans cette figure, l'axe horizontal représente le temps en minutes et l'axe vertical représente les identifiants des partitions.



(a) Exemple de migration des répliques dans Swift



(b) Séquence optimisée de migration des répliques

Figure 1.2 Séquences de migration des répliques

Chaque nœud de stockage commence à copier séquentiellement les partitions aux nœuds qui devraient détenir une copie de ces partitions et ne le sont pas. Cependant, les problèmes suivants se posent :

— *Disponibilité* : Le nombre d'heures pour restreindre la migration simultanée des copies d'une même partition est fixé par le fournisseur de stockage. Dans ce scénario, ce paramètre est fixé à une heure. Cependant la migration de partitions peut durer plus qu'une heure. Ceci peut nuire à la disponibilité minimale tolérée par partition. Dans le scénario présenté, la C4 doit être créée dans le centre de données DC3 dans la première heure de migration et C3 doit être créée dans le centre de données DC3 dans la seconde heure. Or, la création de la copie de C4 a duré plus qu'une heure. Et, au début de la seconde heure, C3 et C4 ne sont pas encore totalement stockées dans le centre de données DC3. Ceci nuit à la disponibilité minimale requise par partition, vu que seulement 2 des copies de la partition C ne sont pas disponibles au même moment.

— *Redondance* : La copie D4 de la partition D devrait être créée dans le centre de données DC3. Cependant, D3 et D4 sont copiés simultanément à partir des centres de données DC1 et DC2 respectivement. Cette redondance dans la copie engendre une consommation inutile de la bande passante, ce qui augmente le temps de la migration des données.

— *Temps de migration* : Chaque nœud de stockage sur Swift copie les répliques des partitions qui doivent être migrées sans aucun calcul des délais. Par conséquent, un nœud de stockage pourrait copier une réplique, même si cette copie peut être effectuée d'une manière plus rapide à partir d'un autre serveur. Par exemple, afin de créer D4 dans DC3, la copie D3 de DC1 prend moins de temps que la copie D4 de DC2. Cela pourrait être dû à la différence de la bande passante disponible et aux retards de propagation

entre les centres de données. De plus, la copie d'une réplique entre les nœuds d'un même centre de données aboutira à un meilleur temps de migration.

— *Période d'inactivité* : Nous constatons qu'il y a toujours un temps entre deux séquences de migration où il n'y a pas de nœuds qui sont en train de copier leurs partitions. Cependant, le placement des répliques n'est pas encore optimal, mais il faut attendre l'écoulement de l'intervalle d'une heure avant de commencer la prochaine migration.

Pour éviter les limites mentionnées ci-dessus, nous proposons une solution de migration optimale. Le plan de migration optimale proposé est illustré dans la figure 1.2(b). On remarque que lorsqu'on évite la redondance de la copie des répliques, la bande passante disponible augmente et engendre la diminution du temps de migration. En outre, la sélection intelligente de la source de copie réduit aussi le temps de migration de manière significative. En fait, la copie de répliques entre les nœuds d'un même centre de données ou entre les nœuds de centres de données ayant de meilleurs délais de propagation et une plus haute capacité de liens engendre un temps de migration plus rapide. Finalement, l'automatisation du recalcul du nouveau placement des répliques des partitions après chaque séquence de migration supprime les délais d'inactivité et mène à une convergence plus rapide vers le placement optimal des répliques dans les centres de données.

CHAPITRE II

OPTIMISATION DU PLACEMENT DE CONTENU DANS LES CCDN

Dans ce chapitre, nous présentons la formulation en programme linéaire en nombres entiers du problème de placement de contenu dans un CCDN. Notre modèle vise à placer sur la base de la popularité les copies de données dans les serveurs périphériques tout en minimisant les coûts de stockage et de la bande passante et, la distance entre les contenus hautement corrélés. La limitation de la résolution de ce problème à une faible échelle, nous a motivé à proposer une heuristique. Cette heuristique est basé sur l'optimisation par essaims particulaires (OEP) et nous permet de résoudre le problème à une plus grande échelle. Finalement, nous évaluons les performances de notre solution heuristique par rapport à la solution optimale du problème de placement et à d'autres stratégies de placement.

2.1 Modèle de placement de contenu

Le déploiement et la maintenance des serveurs périphériques du CDN sont coûteux, et engendrent par conséquent un coût élevé pour les fournisseurs de contenus et les utilisateurs (Xu *et al.*, 2006). Cependant, l'avènement de l'approvisionnement élastique des ressources dans le cloud s'avère une solution rentable pour les fournisseurs de CDN, qui peuvent désormais louer des ressources pour le stockage, le calcul, et la bande passante, selon leurs besoins, afin de

construire des CDN basés sur le cloud appelé CCDN. En plaçant le contenu dans le cloud, les fournisseurs de CDN augmentent la disponibilité et réduisent le temps d'accès au contenu, tout en minimisant le coût de stockage des données. Le succès des CCDN est aussi lié au placement stratégique du contenu qui améliore la qualité de service liée au temps de réponse des requêtes des utilisateurs tout en minimisant le coût de placement et de la bande passante utilisée. Les algorithmes les plus utilisés dans les CDN traditionnels sont des algorithmes réactifs appelés "pull-based algorithms". Dans ces algorithmes quand un serveur reçoit une demande pour un contenu qu'il n'héberge pas, celui-ci apporte/récupère le contenu à partir d'un autre serveur périphérique ou à partir du serveur d'origine. Toutefois, étant donné que la taille de ces serveurs est limitée, les fournisseurs de CDN emploient des algorithmes de remplacement de cache qui effacent le contenu le moins utilisé dans une fenêtre temporelle bien déterminée. Ces algorithmes "pull-based" sont largement utilisés en raison de leur faible coût. Cependant, ces algorithmes ne peuvent pas être utilisés dans le cadre des CCDN parce qu'ils sont coûteux en termes de bande passante à cause de la récupération répétitive du contenu.

D'autre part, il existe des algorithmes de placement proactifs qui placent le contenu dans les serveurs périphériques sur la base d'une estimation des demandes des utilisateurs finaux. En outre, Cisco prévoit que d'ici 2019, plus de 70% du trafic Internet généré à partir des services, tels que la vidéo à la demande (VoD) sera livré par CDN (Cisco, 2012). Les services de VoD contemporains ont souvent une composante liée aux réseaux sociaux en ligne (OSN) qui permet aux utilisateurs d'héberger et de partager du contenu vidéo avec d'autres utilisateurs finaux. Ce contenu généré par l'utilisateur (UGC) et par les réseaux sociaux apporte de nouveaux types de relations entre le contenu lié à la corrélation et la popularité du contenu. Par conséquent, il faut adapter les algorithmes de placement de contenu

dans les CCDN pour le contenu vidéo généré par les utilisateurs (UGC).

Ainsi, la différence entre les modèles de coût entre les CDN et les CCDN et la relation complexe qui lie les vidéos générées par les utilisateurs empêchent l'utilisation des algorithmes de placement de contenu des CDN aux CCDN. À cet effet, nous proposons une stratégie de placement de contenu "pull-based" pour le contenu vidéo hébergée dans les CCDN. Dans notre approche, nous considérons des serveurs périphériques hiérarchisés de CCDN et nous proposons un modèle de placement de contenu basé sur la popularité et la corrélation des données et minimisant le coût de placement et de bande passante.

2.2 Formulation du problème

Dans cette section, nous présentons une formulation mathématique de notre modèle. Nous proposons un nouveau modèle basé sur l'organisation hiérarchique des serveurs périphériques de CCDN, ayant des capacités et des coûts différents. Ce modèle est complété par une stratégie de placement basée sur la popularité et la corrélation du contenu minimisant le coût des ressources et le temps de réponse perçue par l'utilisateur. Nous plaçons les données les plus populaires dans les serveurs périphériques d'accès pour qu'elles soient proches des utilisateurs dans le but de réduire le temps de réponse. L'accès au contenu suit la corrélation des données. Nous rapprochons le contenu corrélé dans le but de réduire le temps d'accès.

Dans notre modèle de placement de contenu, les requêtes d'un utilisateur émises à un serveur périphérique d'accès peuvent être servies par de multiples serveurs périphériques et par différents chemins. Cela permet d'avoir un modèle plus réaliste qui veille à ne pas congestionner la bande passante du réseau sous-jacent et permet d'offrir un meilleur service aux utilisateurs finaux. Nous déterminons donc les chemins utilisés pour la transmission de ces requêtes. Chaque serveur qui

transmet un contenu doit héberger ce contenu. Nous assurons aussi que la capacité des différents serveurs ne soit pas dépassée. D'autre part, notre modèle vise aussi à satisfaire l'accord de service relatif au temps de réponse des utilisateurs. Pour ce faire, nous identifions la latence perçue en servant les requêtes des utilisateurs à travers chaque chemin. Nous assurons ensuite que la qualité de service relative au temps de réponse respecte l'accord de service.

2.2.1 L'énoncé du problème

Nous considérons une topologie de cloud constituée de régions géographiquement distribuées. Chaque région contient un ensemble de zones. Ces zones sont isolées les unes des autres et sont généralement utilisées par les fournisseurs de cloud pour améliorer la disponibilité des données et des applications. Cette topologie est représentée à l'aide du graphe $G = (V,\ E)$, où V est l'ensemble des nœuds qui modélisent les zones reliées par les liens directionnels de l'ensemble E. Chaque lien $e_{v,v'} \in E$ a une capacité de bande passante $B_{e_{v,v'}}$. Cette infrastructure de cloud héberge un CDN suivant une topologie hiérarchique comme le montre la Figure 2.1. Dans notre modélisation, nous représentons les zones par des serveurs périphériques . Ces serveurs périphériques sont reliés par des liens de haute bande passante et de faible latence. Finalement, nous considérons que les demandes des utilisateurs sont déjà redirigées vers les serveurs périphériques par le redirecteur sur la base de l'emplacement géographique de l'utilisateur, mais, ignorant le placement de contenu. Donc la requête d'un utilisateur est redirigée vers le serveur périphérique le plus proche de lui, même si le contenu demandé n'y est pas stocké.

Nous assumons qu'un serveur est représenté par une zone de l'ensemble V. Ainsi, nous considérons l'ensemble des serveurs origines $V_o \in V$. Les serveurs d'origine ont une capacité de stockage permettant de stocker toutes les données hebergées dans le CDN. Nous considérons aussi $V - V_o$ comme étant l'ensemble des serveurs

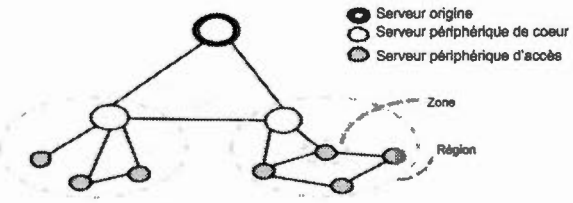


Figure 2.1 Architecture hierarchisés de serveurs du CCDN

périphériques avec une capacité de stockage limitée, où V_c et V_e sont les ensembles de serveurs périphériques de coeur et d'accès respectivement.

Nous considérons aussi l'ensemble C de contenus hébergés dans le CCDN. La taille d'un contenu $c \in C$, est dénotée $|c|$. Notre modèle de placement de contenu est basé sur les relations entre le contenu. En effet, nous visons à travers notre modèle à minimiser la distance entre le contenu hautement corrélé. Pour ce faire, nous considérons un facteur de corrélation $\sigma_{c,c'}$ qui sera égal à 1 s'il existe une relation entre le contenu c et c', $\forall c, c'$, $c \neq c' \in C$. De plus, la relation entre le contenu est aussi mise en valeur à travers la popularité du contenu. Nous considérons la métrique ρ_c, indique la popularité d'un contenu $c \in C$.

Notre modèle de placement de contenu se base sur la connaissance préalable des requêtes émises par les utilisateurs. Ainsi, nous considérons $r_{v,c}$ comme étant le nombre de requêtes pour un contenu $c \in C$ pour un serveur périphérique $v \in V_e$.

Notre modèle de placement de contenu vise à minimiser le coût de stockage et de bande passante. Nous considérons la fonction $\alpha(v)$ qui donne un coût normalisé du stockage d'un contenu dans un serveur $v \in V_c \cup V_e$ et la fonction asymétrique $\beta(l_{e_{v,v'}}, e_{v,v'})$ qui donne un coût normalisé de la bande passante utilisée $l_{e_{v,v'}}$ dans le lien directionnel $e_{v,v'} \in E$. La fonction $\gamma(\pi_{v,v'})$ donne la latence dans le chemin reliant v jusqu'à v', où $v, v' \in V$. Soit Q le paramètre qui indique le seuil de la

QoS relative à la latence et S le paramètre qui indique l'accord de service requis (Service Level Agreement)(SLA) .

Nous voudrions trouver une stratégie de placement de contenu basée sur la popularité, qui minimise conjointement les coûts opérationnels de stockage et bande passante, la distance entre les contenus hautement corrélés et maximise la QoS, tout en respectant la SLA relative à la QoS, toutes les requêtes des utilisateurs finaux et les contraintes liées à la capacité de stockage et à la capacité des liens.

Les Tableaux 2.1 et 2.2 montrent respectivement les entrées et les variables du programme linéaire en nombres entiers.

Tableau 2.1: Les entrées du problèmes de placement dans les CCDN

Entrée	**Définition**
C	L'ensemble de contenu à placer $C = \{c_1, c_2, ..., c_i, ..., c_k, ..., c_{\|C\|}\}$ et $\|c_j\|$ est la taille du contenu c_j
V	L'ensemble des nœuds qui modélisent les serveurs
V_o	L'ensemble des serveurs d'origine, où $V_o = \{v_{o1}, v_{o2}, ..., v_{oi}, ..., v_{ok}, ..., v_{o\|V_o\|}\}$
V_c	L'ensemble des serveurs périphériques de coeur, où $V_c = \{v_{c1}, v_{c2}, ..., v_{ci}, ..., v_{ck}, ..., v_{c\|V_c\|}\}$
V_e	L'ensemble des serveurs périphériques de coeur, où $V_e = \{v_{e1}, v_{e2}, ..., v_{ei}, ..., v_{ek}, ..., v_{e\|V_e\|}\}$
E	L'ensemble des liens connectant les nœuds de l'ensemble V
B_e	Vecteur de taille $\|E\|$ représantant la capacité de la bande passante $\forall e \in E$

Entrée	Définition
$\sigma_{c,c'}$	Matrice de taille $\|C\| \times \|C\|$ représentant le facteur de corrélation où $\sigma_{c,c'} = \begin{cases} 1, \text{ si } c \text{ est correlée à } c' \\ 0, \text{ sinon} \end{cases}$
ρ_c	Vecteur de taille $\|C\|$ représentant la popularité d'un contenu $\forall c \in C$
$r_{v,c}$	Matrice de taille $\|V\| \times \|C\|$ décrivant le nombre de requêtes pour un contenu $c \in C$ pour un serveur périphérique $v \in V_e$
$\alpha(v)$	Une fonction qui donne un coût normalisé de stockage pour un contenu dans un serveur $v \in V_c \cup V_e$
$\beta(l_{e_{v,v'}}, e_{v,v'})$	Une fonction asymétrique donne un coût normalisé de la bande passante utilisée pour une charge $l_{e_{v,v'}}$ dans le lien directionnel $e_{v,v'} \in E$.
$\gamma(\pi_{v,v'})$	Une fonction qui donne la latence en secondes dans un chemin liant v jusqu'à v', où $v, v' \in V$.
G	Matrice de taille $\|V\| \times \|V\| \times \|E\|$ représentant les liens entre les nœuds, où $g_{v,v',e_{v,v'}} = \begin{cases} 1, \text{ si le lien } e_{v,v'} \text{ appartient au plus court chemin reliant } v \text{ à } v' \text{ où } v, v' \in V \\ 0, \text{ sinon} \end{cases}$
$\pi_{v,v'}$	Le plus court chemin entre v et v' où $v, v' \in V$
μ	Granularité des tableaux de recherche (LUT)
$V_{p,e_{v,v'}}$	Tableau de recherche pour la latence en fonction de la charge p sur le lien $e_{v,v'} \in E$
$W_{p,e_{v,v'}}$	Tableau de recherche pour le coût de la bande passante en fonction de la charge p sur le lien $e_{v,v'} \in E$
$\delta_{v,v'}$	Distance normalisée entre v et v' $\forall v, v' \in V$

Entrée	Définition
ω_v	Capacité du serveur périphérique $v \in V_c \cup V_e$
U	Seuil supérieur pour la latence des chemins
T_s	Latence d'accès au serveurs et aux disques
T_{ISP}	Latence de diffusion de l'ISP
Q	Seuil de la QoS
S	SLA slack
A_r	Débit d'accès au contenu
M	Coût maximal de la bande passante
K	Une grande constante

Tableau 2.2: Les variables du problème de placement dans les CCDN

Variable	Définition
X	Matrice de taille $\|V\| \times \|C\|$ représentant le placement de contenu $x_{v,c} = \begin{cases} 1, \text{ si le contenu } c \in C \text{ est placé dans le serveur périphérique } v \in V_c u V_e \\ 0, \text{ sinon} \end{cases}$ On note que, $x_{v,c} = 1$, $v \in Vo, c \in C$, étant donnée que le serveur d'origine stocke toujours une copie du contenu.
Y	Matrice de taille $\|V\| \times \|V\| \times \|C\|$ représentant la proportion du nombre de requêtes pour le contenu $c \in C$ transmis de v à v', où $v, v' \in V$
Z	Matrice de taille $\|V\| \times \|V\|$ représentant la violation par la QoS de la SLA

L	Vecteur de taille $\|E\|$ capturant la charge sur le lien directionnel $e_{v,v'} \in E$ de v à v', où $v, v' \in V$
F	Matrice de taille $\|E\| \times \|P\|$ représntant la valeur binaire pour l'index LUT de la latence pour la charge p dans le lien $e_{v,v'} \in E$ de v à v', où $v, v' \in V$
H	Matrice de taille $\|E\| \times \|P\|$ représantant la valeur binaire pour l'index LUT du coût de la bande passante pour la charge p dans le lien directionnel $e_{v,v'} \in E$ de v à v', où $v, v' \in V$
D	Vecteur de taille $\|E\|$ capturant le délai sur le lien $e_{v,v'} \in E$ de v à v', où $v, v' \in V$
A	Matrice de taille $\|V\| \times \|V\|$ représantant les chemins utilisés lors des transferts des données $a_{v,v'} = \begin{cases} 1, \text{ si le chemin entre v et v' est utilisé, } v, v' \in V \\ 0, \text{ sinon} \end{cases}$
O	Matrice de taille $\|V\| \times \|V\|$ utilisée comme variable auxiliaire utilisée pour l'élimination du produit non-linéaire $a_{v,v'} \times \gamma(\pi_{v,v'})$
ϕ	Matrice de taille $\|V\| \times \|C\| \times \|V\| \times \|C\|$ utilisée comme variable binaire auxiliaire utilisée pour l'élimination du produit binaire non-linéaire $x_{v,c} \times x_{v',c'}$, où $v, v' \in V$ et $c, c' \in C$
τ	Matrice de taille $\|C\| \times \|C\|$ $\tau_{c,c'}$ représantant la distance minimale entre les emplacement des contenus $c, c' \in C$

2.2.2 Les contraintes

Notre modèle de placement de contenu doit veiller à ce que toutes les demandes des utilisateurs finaux soient servies en respectant l'accord de service (SLA) définie pour la QoS relative à la latence. Le modèle doit aussi assurer que la capacité de stockage des serveurs périphériques et de la bande passante ne soient pas dépassée.

Satisfaction des requêtes des clients

Nous allons d'abord nous assurer que les requêtes des clients soient servies et nous déterminerons les chemins par lesquels ces requêtes vont être transmises. Dans notre modèle de placement de contenu, les requêtes d'un utilisateur pour un contenu $c \in C$ émises à un serveur périphérique d'accès $v' \in V_e$ peuvent être servies par de multiples serveurs périphériques et par différents chemins. La contrainte (2.1) assurera que les requêtes $r_{v',c}$ pour un contenu $c \in C$ émises à un serveur périphérique d'accès $v' \in V_e$ soient transmises par l'intermédiaire de plusieurs serveurs $v \in V$ et que la somme des portions transmises par tous ces serveurs soit égale au nombre de requêtes pour ce contenu. Ceci va assurer ainsi que l'on ne transmet que le contenu requis par les requêtes des utilisateurs.

$$\sum_{v \in V} y_{v,v',c} = r_{v',c} \qquad \forall\, v' \in V_e,\ \forall\, c \in C \tag{2.1}$$

Les contraintes (2.2) et (2.3) capturent dans la matrice A les chemins utilisés pour transférer les différentes portions des requêtes.

$$\sum_{c \in C} y_{v,v',c} \leq K \cdot a_{v,v'} \qquad \forall\, v, v' \in V \tag{2.2}$$

$$\sum_{c \in C} y_{v,v',c} \geq a_{v,v'} \qquad \forall\, v, v' \in V \tag{2.3}$$

Afin de satisfaire les requêtes des utilisateurs finaux, le contenu doit être hébergé dans les serveurs périphériques de cœur et d'accès à partir desquels le contenu sera transféré. La contrainte (2.4) assure ceci. La matrice X représente le placement du contenu dans les différents serveurs. Ainsi, si $x_{v,c} = 1$ si le contenu $c \in C$ est stocké dans le serveur $v \in V$.

$$\sum_{v' \in V_e} y_{v,v',c} \leq K \cdot x_{v,c} \qquad \forall\, v \in V_c \cup V_e,\ \ \forall\, c \in C \tag{2.4}$$

Satisfaction de l'accord de service (SLA)

Le contenu doit être livré aux utilisateurs finaux à travers des chemins qui ne violent pas la SLA relative au temps de réponse. Dans les travaux précédents, deux types d'approches ont été adoptées pour limiter la QoS. La première (Salahuddin *et al.*, 2015) consiste à fixer un seuil que la QoS ne doit pas dépasser, et la seconde (Papagianni *et al.*, 2013; Chen *et al.*, 2012) consiste à fixer une distance pour placer le contenu par rapport aux utilisateurs qui permet de ne pas dépasser un seuil pour la QoS.

Dans notre approche, nous garantissons uniquement que les valeurs de la QoS relative au temps d'accès ne dépassent pas le seuil Q pour un certain nombre de chemins. Le nombre de chemins S pouvant dépasser Q est défini dans la SLA, par exemple, 99% des chemins délivreront un service avec un temps de réponse inférieur à Q, ($S = 1$ dans ce cas). Ainsi, dans les contraintes (2.5) nous capturons les chemins dont le transfert de contenu excède le seuil Q dans la matrice Z.

$$a_{v,v'} \cdot (\gamma(\pi_{v,v'}) + T_{ISP} + T_S) - Q \leq K \cdot z_{v,v'} \quad \forall v, v' \in V \tag{2.5}$$

Les contraintes (2.6), (2.7) et (2.8) sont utilisées dans le but de linéariser le produit $a_{v,v'} \cdot \gamma(\pi_{v,v'})$ de la contrainte (2.5). La variable O est une matrice intermédiaire utilisée pour la linéarisation du produit. Nous obtenons ainsi la contrainte linéarisée (2.9) .

$$o_{v,v'} \leq U \cdot a_{v,v'} \quad \forall v, v' \in V \tag{2.6}$$

$$o_{v,v'} \leq \gamma(\pi_{v,v'}) \quad \forall v, v' \in V \tag{2.7}$$

$$o_{v,v'} \geq \gamma(\pi_{v,v'}) - U \cdot (1 - a_{v,v'}) \quad \forall v, v' \in V \tag{2.8}$$

$$o_{v,v'} + (T_{ISP} + T_S) \cdot a_{v,v'} - Q \leq K \cdot z_{v,v'} \quad \forall v, v' \in V \tag{2.9}$$

Puis, dans la contrainte (2.10) nous permettons à S chemins de dépasser le seuil Q.

$$\sum_{v,v' \in V} z_{v,v'} \leq (1 - \frac{S}{100}) \cdot \sum_{v,v' \in V} a_{v,v'} \tag{2.10}$$

Identification de la latence des chemins

Dans notre modèle, nous considérons la qualité de service relative à la latence. La fonction de calcul de la latence est non-linéaire, par conséquent, elle est généralement calculée avec une fonction approximative basée sur la distance, le nombre de sauts ou une table de recherche. Notre modèle utilise une table de recherche pour l'identification de la latence similairement aux travaux (Anttonen et Mämmelä, 2014; Luan *et al.*, 2010).

Construction de la table de recherche

La table de recherche de la latence est une table qui, pour chaque charge de lien, nous donne la latence correspondante. Nous avons construit une table de recherche pour chaque lien $e_{i,j} \in E$. Dans ces tables, la valeur de la latence prend en considération le temps de traitement, le délai d'attente, le temps de transmission et de propagation. Ces tables de recherche (LUT) pour la latence peuvent aussi être basées sur des mesures expérimentales. Afin d'avoir un nombre réaliste d'entrées de la table, nous avons fixé une granularité de recherche (un pas) μ. Le nombre d'entrée de cette table est donc $B_{e_{i,j}}/\mu$. Puis, pour chaque entrée $k \in [0, B_{e_{i,j}}/\mu]$, nous calculons la latence relative à la charge $p = k \cdot \mu$ dans la formule (2.11).

$$V_{p,e_{i,j}} = D_t + D_{tr}(B_{e_{i,j}}) + D_p(e_{i,j}) + D_q(p, B_{e_{i,j}}) \qquad \forall e_{i,j} \in E,\ \forall k \in [0, B_{e_{i,j}}/\mu] \tag{2.11}$$

Dans la formule (2.11), D_t est le délai de traitement en secondes, D_{tr} est une fonction calculant le délai de transmission en secondes en fonction de la bande

passante du lien $B_{e_{i,j}}$, D_p est une fonction retournant le délai de propagation en secondes en fonction de la longueur du lien $e_{i,j}$ et D_q est une fonction retournant le délai de la file d'attente en seconde en fonction de la charge p du lien $e_{i,j}$ et de sa bande passante $B_{e_{i,j}}$.

Charge d'un lien

Dans la contrainte (2.12) , la charge d'un lien $e_{i,j} \in E$ est la somme de toutes les transmissions dans lesquelles le lien participe. Ainsi si ce lien est présent dans un chemin ($g_{v,v',e_{i,j}} = 1$) et que ce chemin transmet un contenu $c \in C$ d'un serveur v vers v' , $v, v' \in V$, on rajoute la portion de contenu qu'il transmet $y_{v,v',c} \cdot A_r$ où A_r est le débit.

$$l_{e_{i,j}} = \sum_{v,v' \in V} \sum_{c \in C} g_{v,v',e_{i,j}} \cdot y_{v,v',c} \cdot A \qquad \forall e_{i,j} \in E \tag{2.12}$$

Identification de l'index de la latence dans la table de recherche

En se basant sur la charge totale d'un lien, on identifie l'index de la latence dans la table de recherche. L'index $f_{e_{i,j},p}$ est identifié de telle sorte que la charge p multiplié par la granularité μ correspond à la charge totale du lien $e_{i,j}$, comme l'indique la contrainte (2.13). La contrainte 2.14 assure qu'uniquement un seul index est actif (mis à 1),

$$l_{e_{i,j}} + l_{e_{j,i}} = \mu \cdot \sum_{p=0}^{\frac{B_{e_{i,j}}}{\mu}} p \cdot f_{e_{i,j},p} \quad \forall e_{i,j} \in E \tag{2.13}$$

$$\sum_{p=0}^{\frac{B_{e_{i,j}}}{\mu}} f_{e_{i,j},p} = 1 \quad \forall\, e_{i,j} \in E \tag{2.14}$$

La latence d'un lien

La latence du lien $e_{i,j} \in E$ est capturée dans la variable $d_{e_{i,j}}$ dans l'équation (2.15). Cette latence est tout simplement la latence correspondante à l'index indiqué dans

les contraintes (2.13) et (2.14) dans la table de recherche V

$$d_{e_{i,j}} = \sum_{p=0}^{\frac{B_{e_{i,j}}}{\mu}} f_{e_{i,j},p} \cdot V_{p,e_{i,j}} \quad \forall\, e_{i,j} \in E \tag{2.15}$$

La latence d'un chemin

Finalement, le calcul de la latence d'un chemin est indiqué dans l'équation (2.16) et correspond à la somme des latences de tous les liens le constituant.

$$\gamma(\pi_{v,v'}) = \sum_{e_{i,j} \in E} g_{v,v',e_{i,j}} \cdot d_{e_{i,j}} \quad \forall\, e_{i,j} \in E,\ \forall\, v, v' \in V \tag{2.16}$$

Capacité des serveurs

Le stockage dans le cloud est considéré comme une solution à faible coût où capacité de stockage presque infinie. Cependant, d'un point de vue réaliste, les serveurs périphériques auront une certaine capacité de stockage limitée. Par conséquent, il est important de veiller à ce que la capacité de ces serveurs ne soit pas dépassée dans la contrainte (2.17)

$$\sum_{c \in C} x_{v,c} \cdot |c| \leq w_v \quad \forall v \in V_c \cup V_e \tag{2.17}$$

Minimisation des coûts

Typiquement, l'un des objectifs du modèle de placement de contenu est de minimiser le coût du stockage et de la bande passante nécessaires pour l'hébergement et la distribution des données. Cet objectif est représenté dans (2.18). Le coût de stockage correspond à la somme des coûts du stockage de tout le contenu $c \in C$ dans les serveurs périphériques $v \in V_e \cup V_c$. La fonction $\alpha(v)$ correspond au coût du stockage d'un contenu dans le serveur v. Le coût de la bande passante d'un lien $e_{v,v'} \in E$ est cependant calculé à travers la fonction

$\beta(l_{e_{v,v'}}, e_{v,v'})$ qui est représentée par une table de recherche qui retourne le coût de l'utilisation du lien $e_{v,v'}$ en fonction de sa charge $l_{e_{v,v'}}$. Le coût de la bande passante totale est le coût total de tous les liens.

$$minimiser(\sum_{v \in V_e \cup V_c} \sum_{c \in C} \alpha(v) \cdot x_{v,c} + \sum_{e_{v,v'} \in E} \beta(l_{e_{v,v'}}, e_{v,v'})) \tag{2.18}$$

Calcul du coût d'un lien

La table de recherche pour le coût de la bande passante est inspirée des modèles de facturation de Google et de Amazon. En effet, le coût par unité d'utilisation décroit plus le nombre d'unités d'utilisation croit. Ainsi, le coût unitaire de l'envoi des quelques gigas de donnée est moins élevé que le coût unitaire de l'envoie d'une centaine gigas de données. Le coût de l'envoi correspond à la quantité de données envoyée multipliée par le coût unitaire. La table de recherche W suit une granularité μ et retourne pour chaque charge p un coût normalisé de l'utilisation de la bande passante dans le lien directionnel $e_{i,j}$

Par conséquent, dans les contraintes (2.19) et (2.20), nous recherchons l'index correspondant à la charge du lien $l_{e_{i,j}}$.

$$l_{e_{i,j}} = \mu \cdot \sum_{p=0}^{B_{e_{i,j}}/\mu} p \cdot h_{e_{i,j},p} \quad \forall e_{i,j} \in E \tag{2.19}$$

$$\sum_{p=0}^{B_{e_{i,j}}/\mu} h_{e_{i,j},p} = 1 \quad \forall e_{i,j} \in E \tag{2.20}$$

Finalement, nous calculons le coût de la bande passante du lien $e_{i,j}$ en nous basant sur l'index retourné par la variable h et de la table de recherche W

$$\beta(l_{e_{v,v'}}, e_{v,v'}) = \sum_{p=0}^{B_{e_{i,j}}/\mu} h_{e_{i,j},p} \cdot W_{p,e_{i,j}} \quad \forall e_{i,j} \in E \tag{2.21}$$

Les relations entre le contenu

Notre modèle de placement, basé sur les relations entre le contenu, vise à rapprocher le contenu le plus populaire et corrélé des utilisateurs finaux. Ainsi, dans (2.22) nous minimisons la distance entre le contenu corrélé et nous pénalisons le placement du contenu le plus populaire dans les serveurs périphériques de coeur. Dans la première partie de (2.22) , nous minimisons la somme des distances $\tau_{c,c'}$ pour les contenus $c, c' \in C$ tout en tenant compte de leurs corrélations normalisées ($\frac{\sigma_{c,c'}}{\sum_{k,k',k\neq k'\in C}\sigma_{k,k'}}$). Dans la seconde partie de (2.22), nous rapprochons le contenu le plus populaire des utilisateurs finaux. Ainsi, le placement d'un contenu dans un serveur périphérique de coeur $x_{v,c}$ est multiplié par la popularité de ce contenu ρ_c. En minimisant la somme de ces popularités, nous incitons le modèle à placer le contenu dans les serveurs périphériques d'accès.

$$minimiser(\sum_{c,c',c\neq c'\in C} \frac{\sigma_{c,c'}}{\sum_{k,k',k\neq k'\in C}\sigma_{k,k'}} \cdot (-\tau_{c,c'}) + \sum_{v\in V_c, c\in C} \rho_c \cdot x_{v,c}) \tag{2.22}$$

Calcul de la distance minimale entre le contenu

La distance minimale entre le contenu c et le contenu c' placés dans les serveurs v et v' respectivement, est calculée dans la contrainte (2.23) où $\delta_{v,v'}$ est la distance entre les serveurs v et v'. La linéarisation de la fonction *argmin* est représentée dans la contrainte (2.24)

$$\tau_{c,c'} = argmin(x_{v,c} \cdot x_{v',c'} \cdot \delta_{v,v'}, \forall v, v' \in V) \quad \forall c, c', c \neq c' \in C \tag{2.23}$$

$$-(\phi_{v,c,v',c'} \cdot \delta_{v,v'}) \leq \tau_{c,c'} \quad \forall v, v' \in V,\ \forall c, c', c \neq c' \in C \tag{2.24}$$

Finalement, nous avons modélisé les contraintes (2.25), (2.26) et (2.27) afin de linéariser le produit $x_{v,c} \cdot x_{v',c'}$ en utilisant la variable intermédiaire $\phi_{v,c,v',c'}$.

$$\phi_{v,c,v',c'} \leq x_{v,c} \quad \forall v, v' \in V, \forall c, c' \in C | c \neq c' \tag{2.25}$$

$$\phi_{v,c,v',c'} \leq x_{v',c'} \quad \forall v, v' \in V, \forall c, c' \in C | c \neq c' \tag{2.26}$$

$$\phi_{v,c,v',c'} \geq x_{v,c} + x_{v',c'} - 1 \quad \forall v, v' \in V, \forall c, c' \in C | c \neq c' \tag{2.27}$$

2.2.3 La fonction objectif

Notre objectif est de trouver la stratégie de placement de contenu basé sur la popularité qui tend à minimiser conjointement les coûts d'exploitation (stockage et bande passante) et la distance entre le contenu hautement corrélé tout en respectant la qualité de service définie par la SLA, les demandes de l'utilisateur final et les capacités de bande passante.

La fonction objective décrite dans (2.28) vise donc à trouver un équilibre entre les coûts relatifs au stockage et à la bande passante, et, la latence relative à l'extraction du contenu populaire et corrélé. En effet, nous pénalisons le modèle quand le contenu le plus populaire n'est pas stocké dans les serveurs périphériques d'accès. Ceci va induire un plus grand coût pour le stockage de ces données, étant donné que le coût des serveurs d'accès est plus élevé, mais va permettre au modèle de bénéficier d'une meilleure QoS.

$$\begin{aligned} minimiser \sum_{v \in V_e \cup V_c} \sum_{c \in C} \alpha(v) \cdot x_{v,c} + \sum_{e_{v,v'} \in E} \beta(l_{e_{v,v'}}, e_{v,v'}) \\ + \sum_{c,c',c \neq c' \in C} \frac{\sigma_{c,c'}}{\sum_{k,k',k \neq k' in C} \sigma_{k,k'}} \cdot (-\tau_{c,c'}) + \sum_{v \in V_c, c \in C} \rho_c \cdot x_{v,c} \end{aligned} \tag{2.28}$$

2.3 Optimisation par essaims particulaires pour le placement des données

Étant donnée la np-complitude du problème de placement de répliques de donnée (Benoit *et al.*, 2007), la résolution du problème que nous avons formulé en nombres entiers est uniquement proposé pour une petite quantité de données. Cependant, le CCDN héberge un nombre élevé de données. C'est pour cette raison que nous proposons dans cette section une heuristique visant à fournir

une solution sous-optimale pour le problème de placement de répliques dans un CCDN. Notre heuristique pour le placement des répliques de données est basée sur la popularité. Elle vise aussi à minimiser les coûts de stockage et de bande passante et à minimiser la distance entre le contenu hautement corrélé. Cette stratégie a pour but en plus de minimiser les coûts, d'améliorer la QoS globale relative aux requêtes des clients et de minimiser les déplacements des répliques entre les zones et les centres de données. Notre heuristique respecte aussi les contraintes liées à la capacité des serveurs et à la capacité des liens et veille à ne pas enfreindre l'accord de service.

L'heuristique que nous proposons est basée sur la métaheuristique d'optimisation par essaims particulaires (OEP). OEP est considéré comme ayant un concept simple, une mise en œuvre facile et une convergence rapide (Yu *et al.*, 2004). C'est pour ces raisons que cet algorithme a gagné beaucoup d'attention et une large application dans différents domaines. OEP a également été reconnu pour être robuste dans la résolution des problèmes se caractérisant par la non-linéarité, la non-différentiabilité et une haute dimensionnalité (Akjiratikarl *et al.*, 2007). Par ailleurs, le nombre considérable de données hebergées par le CCDN, augmente la complexité et la dimentionnalité du problème de leurs placement dans le CCDN.

2.3.1 Introduction à l'optimisation par essaims particulaires (OEP)

L'optimisation par essaim de particules (OEP) est une approche stochastique basée sur la population qui vise à résoudre des problèmes d'optimisation continus et discrets. Elle a été développée par Eberhart et Kennedy en 1995 et est inspirée du comportement social du flocage des oiseaux ou au déplacement des poissons en bancs (Kennedy et Eberhart, 1995). Dans l'optimisation des essaims particulaires, des agents logiciels simples, appelés particules, se déplacent dans l'espace de recherche du problème d'optimisation. La position d'une particule représente une

solution candidate au problème. Chaque particule recherche une meilleure position dans l'espace de recherche en changeant sa vitesse selon les règles initialement inspirées par les comportements de flocage des oiseaux.

L'algorithme d'OEP (Kennedy et Eberhart, 1995) commence par la génération de positions aléatoires des particules dans l'espace de recherche. Les vitesses sont généralement initialisées à l'intérieur de l'espace de recherche, mais peuvent également être initialisées à zéro ou à de petites valeurs aléatoires pour empêcher les particules de quitter l'espace de recherche au cours des premières itérations. À chaque itération, la particule est mise à jour suivant deux "meilleures valeurs" :

- La première est la meilleure solution (indiqué par une fonction de "fitness") que la particule a accomplie jusque là, appelée "pBest"
- La seconde est la meilleure solution obtenue jusque là par toutes les particules présentes dans l'espace appelée "gBest"

Au cours de la boucle principale de l'algorithme, à chaque itération t, la vitesse (V_i^{t+1}) et la position (X_i^{t+1}) de chaque particule i sont mises à jour selon les règles (2.29) et (2.30), respectivement, en supposant que l'intervalle de temps entre t et t+1 vaut 1. Cette boucle prend fin quand un critère d'arrêt est atteint (nombre d'itérations maximal par exemple).

$$V_i^{t+1} = w \cdot V_i^t + \phi_1 \cdot U_1^t \cdot (gBest^t - X_i^t) + \phi_2 \cdot U_2^t \cdot (pBest_i^t - X_i^t) \tag{2.29}$$

$$X_i^{t+1} = X_i^t + V_i^t \tag{2.30}$$

Dans l'équation (2.29), w est un paramètre appelé masse d'inertie, ϕ_1 et ϕ_2 sont deux paramètres appelés coefficients d'accélération. Si les valeurs de w, ϕ_1, ϕ_2 sont bien choisis, il est garanti que les vitesses des particules n'augmenteront pas à l'infini (Clerc et Kennedy, 2002). U_1^t et U_2^t sont deux matrices diagonales $n \times n$, où les valeurs de la diagonale principale sont des nombres aléatoires répartis uniformément dans l'intervalle [0, 1), et n est le nombre de particules. À chaque

itération t, ces matrices sont régénérées.

Pour évaluer la position d'une particule par rapport à la solution recherchée, on utilise une fonction de fitness $f(X_i^t)$. La fonction de fitness est la fonction à optimiser dans le problème. $pBest_i^t$ est la meilleure position trouvée par la particule i jusqu'à l'itération t. Ainsi, $\forall k \in [0,t] \quad f(pBest_i^t)$ est meilleure que $f(X_i^k)$. Et, $gBest^t$ est la meilleure position trouvée jusqu'à l'itération t par toutes les particules. Une fois la condition d'arrêt est atteinte, la solution au problème est représentée par le $gBest^t$.

Le pseudocode de l'algorithme OEP est décrit dans Algorithme 1

Algorithme 1 Pseudocode d'OEP

Entrée : Nombre de particules n

Entrée : Condition d'arrêt

Sortie : Meilleure particule

1: $P_n \leftarrow \{\emptyset\}$
2: $t \leftarrow 0$
3: **for chaque** $i \in [0, n]$ **do**
4: *Initialiser la particule* p_i
5: $P_n \leftarrow P_n \cup p_i$
6: $pbest_i^t \leftarrow X_i^t$
7: **end for**
8: $gbest^t \leftarrow$ *La particle* p_i *avec la meilleure fitness* $f(pbest_i^t)$
9: **while** *Condition d'arret n'est pas atteinte* **do**
10: **for chaque** $p_i \in P_n$ **do**
11: *Calculer la vitesse de la particule* V_i^{t+1} *selon* (2.29)
12: *Calculer la nouvelle position de la particule* X_i^{t+1} *selon* (2.30)
13: **end for**
14: **for chaque** $p_i \in P_n$ **do**
15: *Calculer la fonction de fitness* $f(X_i^{t+1})$
16: **if** $f(X_i^{t+1})$ *est meilleure que* $f(pBest_i^t)$ **then**
17: $pBest_i^{t+1} = X_i^{t+1}$
18: **end if**
19: **end for**
20: $gBest^{t+1} = Choisir\ la\ particule\ avec\ la\ meilleure\ f(pbest_i^{t+1})$
21: $t \leftarrow t + 1$
22: **end while**
23: **retourner** $gbest^{t+1}$

2.3.2 Le placement des données basé sur OEP

Nous avons adapté l'algorithme OEP décrit dans la sous-section 2.3.1 afin de proposer notre solution heuristique pour le placement des répliques.

La particule

Dans OEP, une solution potentielle au problème est représentée par la position d'une particule. Dans notre cas, la particule i est représentée par la matrice X_i qui décrit le placement des différents contenus dans les serveurs. Cette matrice est de dimension $|V| \times |C|$ où $|V|$ est le nombre de serveurs et $|C|$ et le nombre contenu. Nous rappelons que la valeur $x_{i,v,c}$ est égale à 1 si le contenu $c \in C$ est placé dans le serveur $v \in V_c \cup V_e$ et 0 s'il ne l'est pas. De plus, étant donné que tout le contenu est placé au début du problème dans le serveur origine, $x_{i,v',c}$ pour $v' \in V_o$ est toujours égale à 1 pour tout contenu $c \in C$.

Calcul de la position et de la vitesse

Pour le calcul de la position et de la vitesse de chaque particule, nous devons mettre à jour tous les éléments de la matrice X_i. Ainsi, $pBest_i^t$ et $gBest^t$ sont aussi des matrices de dimension $|V| \times |C|$. Par ailleurs, nous calculons la vitesse V_i^{t+1} suivant la formule (2.29) et la position X_i^{t+1} suivant la formule (2.30).

Validation de la particule

Le placement du contenu décrit par la particule X_i peut dans certains cas enfreindre des contraintes imposées par notre modèle. Dans ce cas, nous pouvons soit réparer ou pénaliser la particule. Le processus de pénalisation de la particule consiste à attribuer une très haute valeur au résultat de la fonction de fitness afin que celle-ci ne puisse pas être considérée dans le choix des meilleures valeurs de particules. Par ailleurs, le processus de réparation consiste à changer quelques valeurs de la particule afin d'obtenir une particule valide. Nous avons choisi le processus de réparation dans notre heuristique. Le processus de correction de la

particule consiste à changer des valeurs de matrice X_i. L'algorithme 2 décrit le processus de réparation.

Algorithme 2 Réparation de la particule

Entrées : La particule X_i, Tableau 2.1

Sortie : La particule réparée X_i

1: **for chaque** $r_{v,c} \in R$ **do**
2: $v' \leftarrow$ *Le serveur le plus proche de v stockant c*
3: $V_p \leftarrow prioriser\_serveurs(V)$
4: **while** $(\sum_{c \in C} x^{t+1}_{i,v',c}|c| \geq w_{v'})||(\exists e_{k,j} \in E, g^{t+1}_{i,v,v',e_{k,j}} \cdot l^{t+1}_{i,e_{k,j}} \geq B_{e_{k,j}})||(SLA\ est\ viole)$ **do**
5: **if** $V_p \neq \{\emptyset\}$ **then**
6: **if** $v' \notin V_o$ **then**
7: $x^{t+1}_{i,v',c} \leftarrow 0$
8: **end if**
9: $v' \leftarrow$ *Premier element de* V_p
10: $V_p \leftarrow V_p - \{v'\}$
11: $x^{t+1}_{i,v',c} \leftarrow 1$
12: **else**
13: $X^{t+1}_i \leftarrow +\infty$
14: **end if**
15: **end while**
16: **end for**
17: **retourner** X_i

Afin de réparer une particule, nous parcourons toutes les requêtes $r_{v,c} \in R$, et nous considérons que chacune d'elles doit être servie à partir du serveur v' le plus proche stockant le contenu c demandé (ligne 2) . Nous vérifions que la transmission du contenu c à partir du serveur v' vers le serveur v n'enfreint aucune contrainte

telle que le dépassement de la capacité de stockage d'un serveur ou la capacité de transmission d'un lien, ou alors le dépassement du seuil S de la SLA pour la QoS (ligne 4). Si une contrainte n'est pas enfreinte, nous réparons le placement de c dans le serveur v' en plaçant le contenu dans le prochain serveur de la hiérarchie. Pour ce faire, nous priorisons les serveurs de manière à ce que les serveurs dans la même région que v aient une plus haute priorité que les autres serveurs et que les serveurs périphériques de cœur aient une plus haute priorité que les serveurs d'accès au sein d'une même région (ligne 3). Puis nous plaçons le contenu itérativement dans l'un de ces serveurs jusqu'à ce que le placement n'enfreint aucune contrainte et nous supprimons l'ancien placement (ligne 7). Si aucun serveur dans la liste V_p ne satisfait toutes les contraintes, nous pénalisons la particule en attribuant une valeur infinie à la particule i (ligne 13).

Fonction de fitness

La fonction de fitness décrit les objectifs que nous voulons atteindre à l'issue de notre heuristique. Les objectifs de notre modèle de placement sont la minimisation du coût de stockage et de bande passante, la minimisation de la distance entre les contenus fortement corrélés et la pénalisation du placement du contenu populaire dans les serveurs de cœur. Cette fonction est décrite par l'équation 2.31. La particule $gBest^t$ sera la matrice de placement X_i^t qui aura la valeur minimale de $f(X_i^t)$

$$f(X) = \sum_{v \in V_e \cup V_c} \sum_{c \in C} \alpha(v) \cdot x_{i,v,c} + \sum_{e_{v,v'} \in E} \beta(l_{i,e_{v,v'}}, e_{v,v'}) + \sum_{c,c',c \neq c' \in C} \frac{\sigma_{c,c'}}{\sum_{k,k',k \neq k' \in C} \sigma_{k,k'}} \cdot (-\tau_{i,c,c'}) + \sum_{v \in V_c, c \in C} \rho_c \cdot x_{i,v,c} \quad (2.31)$$

Minimisation du coût de stockage

Le coût de stockage ($\sum_{v \in V_e \cup V_c} \sum_{c \in C} \alpha(v) \cdot x_{i,v,c}$) correspond à la somme des coûts de stockage de tous les contenus $c \in C$ dans les serveurs périphériques $v \in V_e \cup V_c$.

La fonction $\alpha(v)$ correspond au coût du stockage d'un contenu dans le serveur v et $x_{i,v,c} = 1$ si le contenu c est placé dans le serveur v dans la particule i.

Minimisation du coût de la bande passante

Le coût de la bande passante $\sum_{e_{v,v'} \in E} \beta(l_{i,e_{v,v'}}, e_{v,v'})$ correspond à la somme des coûts engendrés par tous les liens $e_{v,v'} \in E$. Le coût de l'utilisation d'un lien est calculé à travers la fonction $\beta(l_{i,e_{v,v'}}, e_{v,v'})$ qui est représentée par une table de recherche qui retourne le coût de l'utilisation du lien $e_{v,v'}$ en fonction de sa charge $l_{i,e_{v,v'}}$ engendré par le placement X_i de la particule i. Nous calculons la quantité de données transmise sur chaque lien se trouvant sur le chemin dans la variable L_i relative à la particule i. Nous considérons dans ce calcul que les requêtes des utilisateurs sont servies à partir du serveur le plus proche stockant le contenu demandé.

Minimisation de la distance entre le contenu corrélé

Dans $(\sum_{c,c',c \neq c' \in C} \frac{\sigma_{c,c'}}{\sum_{k,k',k \neq k' in C} \sigma_{k,k'}} \cdot (-\tau_{i,c,c'}))$, nous minimisons la somme des distances $\tau_{i,c,c'}$ pour les contenus $c, c' \in C$ tout en tenant compte de leurs corrélations normalisées $(\frac{\sigma_{c,c'}}{\sum_{k,k',k \neq k' \in C} \sigma_{k,k'}})$

Rapprochement du contenu populaire des utilisateurs

Dans $(\sum_{v \in V_c, c \in C} \rho_c \cdot x_{i,v,c})$, nous faisons la somme des popularités ρ_c des contenus $c \in C$ qui sont placés dans les serveurs périphériques de cœurs V_c. En minimisant cette somme, nous poussons le placement du contenu le plus populaire dans les serveurs périphériques d'accès.

2.4 Evaluation des performances du modèle de placement proposé

Dans cette section, nous allons décrivons l'environnement de simulation que nous avons considéré pour nos simulations. Nous décrirons ensuite les stratégies de placement que nous implémenterons pour comparer leurs performances avec celles

de notre stratégie. En effet, nous avons modélisé en programmation linéaire en nombre entier les problèmes liés à ces différentes stratégies, pour pouvoir comparer les performances des solutions optimales avec celles de notre solution. Nous démontrerons ainsi la faisabilité et les avantages apportés par notre stratégie pour le placement des répliques dans les CCDN.

2.4.1 Description des simulations

La topologie considérée

La topologie utilisée pour les simulations est inspirée de la topologie de EC2 en Amérique du Nord (AMA, 2016). Cette topologie est constituée de 3 régions, deux d'entres elles contiennent 3 zones et une contenant 5 zones. Dans la Figure 2.4.1, une zone est représentée par le couple routeur/serveur.

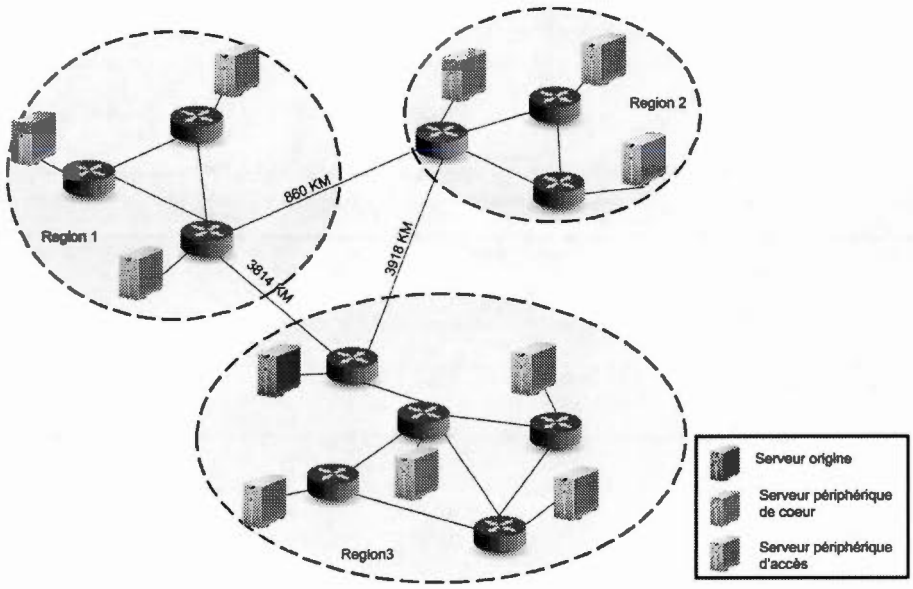


Figure 2.2 La topologie de serveurs utilisée pour les simulations

Dans nos simulations, nous considérons que les serveurs présents dans une même région sont séparés par une distance comprise entre 200 et 400 mètres. Nous

fixons aussi la capacité des liens reliant les différentes régions à 100 Mbps et celle reliant les zones d'une même région est de 1000 Mbps. Pour la taille des serveurs, nous avons calculé la somme de la taille de tout le contenu que nous utilisons dans nos simulations. Nous notons que la capacité du serveur origine est illimité. D'une manière réaliste, nous avons fixé la capacité de ce serveur de manière à ce qu'il puisse héberger tout le contenu. Par ailleurs, la capacité des serveurs périphériques de cœur est égale aux deux tiers de la somme de la taille de tout le contenu. Finalement, nous avons considéré que la capacité des serveurs périphériques d'accès est égale à un tier de la somme de la taille de tout le contenu. Pour ce qui concerne le coût de stockage, nous avons considéré que le coût de placement des données dans le serveur origine est nul. Ce coût est constant et égal pour toutes les stratégies, étant donné que les données sont toutes stockées dans ce serveur au début des simulations. Pour ce qui concerne le coût de stockage relié aux serveurs périphériques d'accès comme étant le double du coût de stockage dans les serveurs périphériques de coeur. Les valeurs des coûts que nous attribuons aux serveurs sont normalisées de manière à ce que la somme de tous les coûts soit égale à 1.

La collecte des données

Les données que nous simulons sont des vidéos extraites de manière aléatoire à partir de YouTube. Ce jeu de données est téléchargeable à partir du site (Zeni, 2016; Zeni *et al.*, 2013) en un fichier contenant la description de 7 millions de vidéos sous format JSON. Nous avons choisi 50 vidéos de manière aléatoire entre le 27 décembre 2013 et le 2 janvier 2014. La figure 2.4.1 présente une partie de la définition d'une vidéo.

Traitement des données

À partir du format de donnée illustré dans la Figure 2.4.1, nous avons extrait et traité pour chaque vidéo les données qui étaient nécessaires pour notre simulation.

```
{ id: u'9eToPjUnwmU',
title: u'Traitor Compilation # 1 (Trouble ...',
...
publishedDate: u'2012-10-09T23:42:12.000Z',
author: u'ServilityGaming',
duration: u'208',
...
},
views: {
...
        daily: {
                data': [15.0, 10.0, 1.0, 0.0, ..]
        }
},
shares: {
        ...
        daily: {
                data: [0.0, 0.0, 0.0, 0.0, ...]
        }
},
Related': { [... ]
},
day': {
data': [1349740800000.0, 1349827200000.0, 1349913600000.0, 1350000000000.0, ...]}
}
```


Figure 2.3 Exemple de la définition d'une vidéo dans le fichier JSON

D'abord, pour le calcul de la taille des vidéos, nous avons considéré la formule 2.32. En effet, à partir du fichier JSON, nous pouvons récupérer la durée de chaque vidéo. Nous avons considéré une taille de l'image 320 × 240 et une profondeur de la couleur égale à 3 bits. Finalement la fréquence des images est fixée à 60 images par seconde.

$$Taille(Mb) = Dure(sec) \times Taille\ du\ cadre\ de\ l'image(Mb) \times Profondeur\ de\ la\ couleur(bit) \times Frequence\ des\ images(MB/sec) \quad (2.32)$$

À partir des champs "related" de chaque vidéo, nous avons établi une matrice de

corrélation pour toutes les vidéos extraites. De plus, la popularité d'une vidéo est calculée à partir du nombre de vues "views" pour le jour considéré comme ratio du nombre total de vue toutes les autres vidéos vues ce jour là comme illustré dans l'équation 2.33

$$Popularite = \frac{Nombre\ de\ vue}{Somme\ des\ nombres\ de\ vue\ de\ toutes\ les\ videos} \tag{2.33}$$

Finalement, le nombre de requêtes considéré pour une vidéo est proportionnel à la popularité de celle ci. Nous fixons le plus haut nombre de requêtes pour une vidéo à 5. Pour ce qui concerne les serveurs périphériques à qui les requêtes ont envoyé, ils seront choisis d'une manière aléatoire de telle sorte que le contenu corrélé ait plus de probabilité d'être sollicité dans le même serveur.

Dans nos simulations, nous avons considéré quatre stratégies pour le placement des copies de données dans les CCDN. La première stratégie est notre stratégie basée sur la popularité qui tend à minimiser les coûts de placement et de bande passante et la distance entre les contenus hautement corrélés. Nous avons aussi considéré la stratégie où seul le coût est pris en compte dans le placement des données. Cette stratégie est restreinte par des contraintes telles que la capacité des serveurs périphériques et a été largement considérée dans la littérature . Nous considérons aussi la stratégie qui tend à minimiser le coût de stockage et de bande passante tout en considérant la popularité des données. Finalement, pour voir le seul impact de la minimisation de la distance entre les données hautement corrélées combinée avec la minimisation du coût de stockage et de bande passante, nous avons considéré la stratégie qui tend à minimiser ces deux variables.

Pour mesurer les performances de ces 4 stratégies, nous avons modélisé les contraintes du problème formulé dans la section 2.2 à l'aide de (AMP, 2016). Puis pour calculer les solutions optimales pour les quatre stratégies citées, nous avons modifié les paramètres de la fonction objective d'une stratégie à une autre.

Finalement, les mesures de performances sont calculées à partir des résultats optimaux obtenus à l'aide du logiciel CPLEXs (Cpl, 2016).

Nous comparons les performances des résultats de la solution optimale relative à notre stratégie de placement avec celles de notre heuristique et des solutions optimales d'autres stratégies.

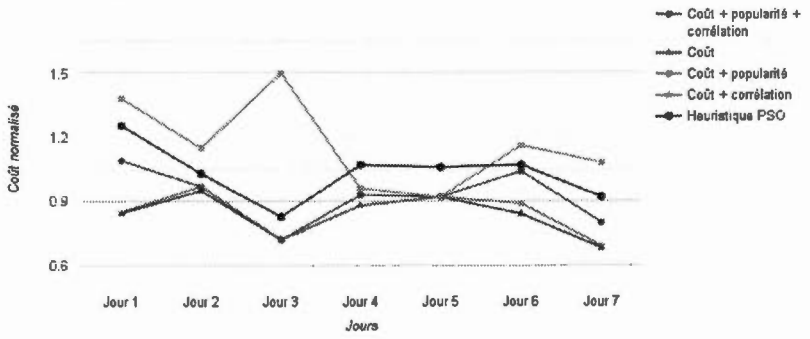
Pour la simulation, nous avons considéré pour la semaine commençant le 24 décembre 2012, 50 vidéos qui ont été visionnées au moins une fois chaque jour.

2.4.2 Les résultats des simulations

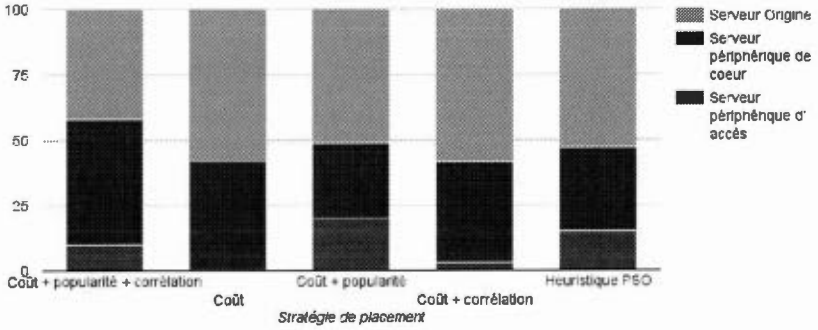
Nous commençons par évaluer le coût de placement et de la bande passante utilisée pour les différentes stratégies. La figure 2.4(a) montre que le la stratégie basée sur la popularité est celle qui présente les plus hauts coûts de stockage et de bande passante. Ceci peut être expliqué par le fait que nous pénalisons le placement de contenu populaire dans les serveurs périphériques de cœur. Il y a donc plus de contenu qui est placé dans les serveurs périphériques d'accès comparé aux autres stratégies, comme illustré dans la figure 2.4(b). Le placement des répliques dans les serveurs les plus proches des utilisateurs a aussi engendré une diminution du temps d'accès comparé aux autres stratégies, comme illustré dans la figure 2.4(c). La figure 2.4(a) montre aussi que la stratégie basée conjointement sur la corrélation et le coût et celle basée uniquement sur le coût présentent des valeurs très proches en termes de coût. Ce coût étant de plus le plus faible. La répartition du contenu dans les serveurs est aussi très proche ce qui explique des coûts presque similaires. En effet, nous remarquons que la majorité du contenu est servie à 60% par le serveur d'origine et le reste est servi par les serveurs périphériques de coeur. Cependant, pour la stratégie se basant conjointement sur le coût et la corrélation, 2% des requêtes sont servies par les serveurs de périphérique d'accès. Par ailleurs, les résultats présentés dans la figure 2.4(c) concernant le délai de réponse montrent

que le délai relatif à la stratégie qui combine le coût et la corrélation du contenu est plus bas. Finalement, les résultats de la solution optimale pour notre stratégie constituent un bon compromis entre les autres stratégies. En effet, d'après la figure 2.4(a), les coûts engendrés par cette stratégie ne sont pas les plus bas, mais s'approchent des valeurs de coûts relatifs à la solution optimale de la stratégie de placement considérant uniquement le coût. De plus, notre stratégie présente des délais de réponse meilleurs que ceux de la stratégie considérant uniquement le coût. En effet, notre stratégie de placement a réalisé des améliorations en termes de délai de réponse aux dépens du coût comparé à la stratégie uniquement basée sur le coût. En revanche, notre stratégie n'a dans aucun des jours présenté des résultats qui étaient les plus bas.

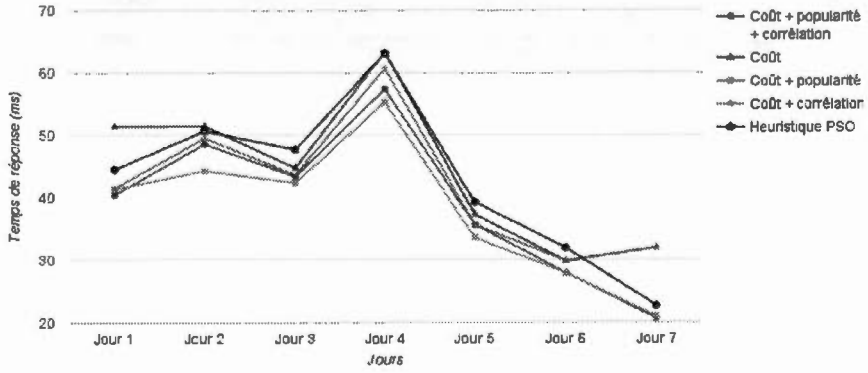
En ce qui concerne notre heuristique, celle-ci présente des résultats sous-optimaux. En effet, le coût de stockage et de bande passante est 20% plus élevé par rapport à celui de la solution optimale et 10% moins élevé que celui de la stratégie considérant le coût et la popularité. De plus, d'après la figure 2.4(b), 15% des requêtes sont servies par les serveurs périphériques d'accès, comparé à 10% pour la solution optimale et 20% pour le placement basé sur le coût et la popularité. Les serveurs périphériques de coeur répondent à 30% des requêtes pour notre heuristique, comparée à 50% et 25% pour la solution optimale et la stratégie considérant le coût et la popularité respectivement. Ceci a influencé les temps de réponse. La figure 2.4(c) montre que le temps de réponse de notre heuristique est le plus haut. Ceci peut être expliqué par le fait que notre heuristique utilise moins de répliques que les autres stratégies.



(a) Coût de stockage et de bande passante normalisé



(b) Répartition du placement des répliques



(c) Délais de réponse des requêtes des utilisateurs

Figure 2.4 Comparaison des performances analytiques des stratégies de placement

CHAPITRE III

OPTIMISATION DE LA MIGRATION DES DONNÉES DANS LES SYSTÈMES DE STOCKAGE DISTRIBUÉS

Dans ce chapitre, nous étudions le problème de la migration des données dans les systèmes de stockage distribués. Nous commençons par décrire la motivation derrière l'étude du problème de la migration des répliques. Ensuite, nous modélisons à l'aide de la programmation linéaire en nombres entiers le problème de la migration dans les systèmes de stockage distribué. Par la suite, nous présentons notre heuristique CRANE . Et, nous finissons par évaluer les performances des résultats de notre heuristique par rapport aux performances de la séquence optimale de migration et par rapport à la séquence générée par l'algorithme d'Openstack Swift.

3.1 Motivations

Avec la large adoption des services Internet et l'analyse des données volumineuses, le cloud est devenu l'environnement idéal pour satisfaire les demandes croissantes de stockage grâce à sa capacité de stockage presque illimitée, la haute disponibilité et le temps d'accès rapide qu'il apporte. Dans ce contexte, la réplication de données a formé une solution ultime pour améliorer la disponibilité des données et de réduire le temps d'accès. Cependant, pour la gestion de ces répliques de données, les systèmes distribués doivent généralement migrer et créer un grand nombre

de répliques de données au fil du temps entre et dans les centres de données. Ceci engendre une surcharge importante en termes de charge du réseau et de disponibilité.

Cependant, les stratégies de placement de répliques peuvent engendrer la création ou la migration d'un grand nombre de répliques dans les centres de données au fil du temps. Ceci impliquera par ailleurs, la génération d'un trafic réseau très important entre les différents centres de données d'un fournisseur de cloud. Plusieurs scénarios peuvent déclencher ce processus de migration de données, par exemple le déploiement d'un nouveau centre de données dans l'infrastructure du fournisseur de cloud, ou la mise à l'échelle d'un centre de données ou lors de la récupération des données après une catastrophe ou tout simplement lorsque les répliques sont déplacées pour atteindre de meilleures performances.

Par ailleurs, un transfert massif de ces données engendre plusieurs conséquences :

— D'abord, étant donné que la copie de données consomme des ressources telles que le CPU , la mémoire, l'écriture et la lecture à partir du disque, au niveau de la machine source et celle de destination, ces noeuds connaîtront plus de partage pour les ressources disponibles, ce qui peut ralentir les autres tâches en cours d'exécution.

— De plus, dans les systèmes de stockage distribué, le processus de migration de répliques est généralement distribués et asynchrone. Par exemple, si on considère le système de gestion de stockage Swift, quand une réplique doit être créée dans une nouvelle machine de destination, chaque machine dans l'infrastructure stockant déjà la même réplique devrait copier les données vers la nouvelle destination. Il n'y a pas de coordination ou de synchronisation entre les noeuds sources. Ainsi, cela va engendrer non seulement une redondance inutile de la copie de données identiques à partir de différentes sources en même temps, mais aussi impliquer la congestion

dans le réseau des centres de données.

— Finalement, les répliques sont généralement placés dans des centres de données géographiquement distants afin d'accroître la disponibilité des données au fil du temps et de réduire la latence perçue par l'utilisateur. De plus, quand une réplique est créée/migrée dans un nouvel emplacement, elle ne sera disponible que lorsque tout son contenu soit copié dans le nœud de destination. Ainsi, si la création de répliques prend un temps important, la disponibilité de données peut être altérée si le nombre de répliques actives ne suffit pas à satisfaire toutes les demandes des utilisateurs. Par exemple, afin d'assurer la disponibilité des données, Swift assure qu'au moins deux répliques des données sont disponibles à tout moment (selon la configuration par défaut (Dickinson, 2013)).

3.2 Formulation du problème

Notre objectif consiste à minimiser d'une part le temps de migration individuel de chaque réplique de partition dans le but de la rendre rapidement accessible dans son emplacement final et optimal. D'autre part, notre objectif vise aussi à minimiser le temps total pris par la migration de toutes les répliques. Nous assurons dans notre modèle que seules les serveurs qui détiennent une réplique entière d'une donnée peuvent initier sa réplication. Les serveurs peuvent cependant héberger une réplique suite à sa migration. Nous garantissons aussi qu'il n'y aura pas de migration inutile de répliques. En effet, nous assurons que seules les répliques qui n'existent pas dans leurs emplacement final sont migrés. D'autre part, nous évitons la redondance de la réplication d'une partition. Notre modèle assure aussi que la capacité des liens et des serveur ne soit pas dépassée.

3.2.1 Énoncé du problème

Nous considérons dans notre problème une topologie de cloud constituée de centre de données géographiquement distribué. Nous représentons le réseau considéré avec le $G = (S, E)$, où S représente l'ensemble de tous les serveurs de tous les centres de données. Nous supposons que ces centres de données sont connectés par un réseau de base appelé aussi "backbone" . Les liens de ce réseau sont représentés par l'ensemble E des cotés du graphe G . Chaque lien $e \in E$ est caractérisé par une bande passante de capacité. B_e. Nous considérons que l'ensemble P représente l'ensemble des partitions dont les répliques sont stockées dans les serveurs de S et $|p_j|$ est la taille d'une réplique de la partition p_j. Nous définissons une configuration comme un placement bien spécifique des répliques dans les serveurs. Étant donnée les configurations initiale et finale désignées respectivement par C^I et C^F, toute différence entre ces configurations nécessite soit la migration ou la suppression d'une réplique de la partition. Nous considérons que la migration ou la suppression d'une réplique est identifiée par les variables $y_{j,k}$ et $d_{k,j}$, respectivement. Notre objectif est de trouver la séquence optimale de la migration des répliques des partitions qui minimise le temps de migration totale de C^I à C^F tout en respectant le seuil de disponibilité minimale d'une partition A et les capacités de bandes passantes des différents liens reliant les centres de données. Nous modélisons ceci comme un problème de programmation linéaire en nombres entiers (PLNE).

Les tableaux 3.1 et 3.2 décrivent respectivement les entrées de la PLNE et ses variables.

Tableau 3.1: Les entrées du problème de la migration des répliques

Entrée	**Définition**
S	Ensemble des serveurs de tous les centres de données
E	Ensemble des liens connectant les serveurs dans S
B_e	Capacité de la bande passante $\forall e \in E$
P	Ensemble des partitions, où $\|p_j\|$ est la taille d'une partition p_j
C^I	Matrice de taille $\|S\| \times \|P\|$ représentant la configuration initiale du placement des répliques, où $c^I_{i,j} = \begin{cases} 1, \text{ si la repléque de } p_j \text{ est stocké dans } s_i \\ 0, \text{ sinon} \end{cases}$
C^F	Matrice de taille $\|S\| \times \|P\|$ représentant la configuration finale du placement des répliques, où $c^F_{i,j} = \begin{cases} 1, \text{ si la repléque de } p_j \text{ est stocké dans } s_i \\ 0, \text{ sinon} \end{cases}$
Y	Matrice de taille $\|P\| \times \|S\|$ représentant le besoin de la migration d'une réplique d'une partition, où $y_{j,k} = \begin{cases} 1, \text{ si la réplique de } p_j \text{ a besoin d'être migrée au serveur } s_k \\ 0, \text{ sinon} \end{cases}$
D	Matrice de taille $\|S\| \times \|P\|$ représentant les répliques qui doivent être supprime au temps T, où $d_{i,j} = \begin{cases} 1, \text{ si la réplique de } p_j \text{ doit être supprimé des} s_i \\ 0, \text{ sinon} \end{cases}$
T	Seuil maximum du temps de migration
A	Seuil de disponibilité minimale pour une partition
G	Matrice de taille $\|S\| \times \|S\| \times \|E\|$ représentant les liens utilisés dans un chemin , où

	$g_{i,k,e} = \begin{cases} 1, \text{ si le lien } e \text{ est utilisé dans le plus court chemin} \\ \quad \text{entre } s_i \text{ et } s_k \\ 0, \text{ sinon} \end{cases}$
α	Un intervalle de temps (1 seconde)
β	Une grande constante

Tableau 3.2: Les variables du problème de la migration des répliques de données

Variable	**Définition**
X	Matrice de taille $\|S\| \times \|P\| \times \|S\| \times T$ représentant la séquence de migration $x_{i,j,k,t} =$ $\begin{cases} 1, \text{ si } s_i \text{ est entrain de migrer la réplique de } p_j \text{ à } s_k \text{ au temps } t \\ 0, \text{ sinon} \end{cases}$
Z	Matrice de taille $\|S\| \times \|P\| \times T$ représantant le placement des répliques $z_{i,j,t} = \begin{cases} 1, \text{ si } s_i \text{ a une réplique de } p_j \text{ au temps } t \\ 0, \text{ sinon} \end{cases}$
R	Matrice de taille $\|S\| \times \|P\| \times \|S\| \times T$ où $r_{i,j,k,t}$ représente la bande passante allouée pour la migration d'une réplique de la partition p_j du serveur s_i au s_k au temps t
W	Un vecteur de taille T, où $w_t = \begin{cases} 1, \text{ si la migration est en cours au temps } t \\ 0, \text{ sinon} \end{cases}$

3.2.2 Les contraintes du problème

Avant de commencer une migration, nous devrions d'abord identifier les serveurs s_i qui maintiennent une réplique de la partition p_j au temps t, dans la variable $z_{i,j,t}$. Au début de la migration uniquement les serveurs s_i contenant une réplique de la partition p_j au début dans la configuration initiale peuvent participer à la

migration comme le décrit la contrainte (3.1). Dans la contrainte (3.2) nous nous assurons qu'un serveur s_i qui ne détient pas une réplique de la partition dans la configuration initiale et finale ne pourra pas participer à la migration.

$$c^I_{i,j} \leq \beta \cdot z_{i,j,t} \quad \forall 1 \leq i \leq |S|, 1 \leq j \leq |P|, 1 \leq t \leq T \tag{3.1}$$

$$z_{i,j,t} \leq c^I_{i,j} + c^F_{i,j} \quad \forall 1 \leq i \leq |S|, 1 \leq j \neq 0 \leq |P|, 1 \leq t \leq T \tag{3.2}$$

La variable $z_{k,j,t}$ qui spécifie les sources potentielles de la migration de répliques est mise à jour à chaque intervalle de temps t. En effet, le serveur s_k peut détenir une réplique d'une partition p_j, après la migration de celle-ci dans un intervalle de temps précédent comme indiquée dans la contrainte (3.3) et (3.4). En outre, un serveur détient une réplique d'une partition p_j quand la somme des bandes passantes allouées à la migration $r_{i,j,k,t'}$ de cette partition du serveur source s_j tau serveur destination s_k, dans les intervalles de temps précédents $\forall t' < t$, est égale à la taille de la partition p_j. Ainsi, le serveur s_k peut potentiellement participer à la migration de la partition p_j ultérieurement.

$$|p_j| - \sum_{i=1, i\neq k}^{|S|} \sum_{t'=1}^{t} r_{i,j,k,t'} \cdot \alpha \leq \beta \cdot (1 - z_{k,j,t+1})$$
$$\forall 1 \leq k \leq |S|, 1 \leq j, c^I_{k,j} = 0 \leq |P|, 1 \leq t \leq T-1 \tag{3.3}$$

$$1 - z_{k,j,t+1} \leq |p_j| - \sum_{i=1, i\neq k}^{|S|} \sum_{t'=1}^{t} r_{i,j,k,t'} \cdot \alpha$$
$$\forall 1 \leq k \leq |S|, 1 \leq j, c^I_{k,j} = 0 \leq |P|, 1 \leq t \leq T-1 \tag{3.4}$$

Une l'initialisation des serveurs pouvant participer à la migration de chaque partition, nous pouvons initier la migration en associant la variable de la migration

$x_{i,j,k,t}$ avec le besoin de migrer une réplique de la partition p_j de s_i à s_k, dans $y_{j,k}$, dans la contrainte (3.5).

$$y_{j,k} \leq \sum_{t=1}^{T} x_{i,j,k,t} \quad \forall 1 \leq i,k,\ i \neq k \leq |S|, 1 \leq j \leq |P| \tag{3.5}$$

En outre, dans la contrainte (3.6), on assure que seuls les serveurs sources s_i, qui détiennent une réplique complète de la partition p_j peuvent initier la migration.

$$\sum_{k=1}^{|S|} x_{i,j,k,t} \leq z_{i,j,t} \quad \forall 1 \leq i \leq |S|, 1 \leq j \leq |P|, 1 \leq t \leq T \tag{3.6}$$

Nous assurons aussi la migration séquentielle des répliques. La contrainte (3.7), assure que chaque serveur ne peut migrer qu'une seule réplique à chaque intervalle t.

$$\sum_{j=1}^{|P|} \sum_{k=1}^{|S|} x_{i,j,k,t} \leq 1 \quad \forall 1 \leq i \leq |S|, 1 \leq t \leq T \tag{3.7}$$

Pour assurer la migration continue et séquentielle de la réplique d'une partition p_j, du même serveur source s_i au même serveur destination s_k pour le prochain intervalle $t+1$, nous fixons l'indicateur de migration $x_{i,j,k,t+1}$, à 1, jusqu'à la migration complète de la réplique. Ceci est pris en compte dans la contrainte (3.8).

$$|p_j| - \sum_{t'=1}^{t} r_{i,j,k,t'} \cdot \alpha \leq \beta \cdot x_{i,j,k,t+1}$$
$$\forall 1 \leq i,k,\ i \neq k \leq |S|, 1 \leq j \leq |P|, 1 \leq t \leq T-1 \tag{3.8}$$

Pendant le processus de migration, les serveurs doivent maintenir le seuil de disponibilité minimale de chaque partition comme illustré dans la contrainte (3.9).

$$\sum_{i=1}^{|S|}\sum_{k=1}^{|S|} x_{i,j,k,t} \geq A \quad \forall 1 \leq j \leq |P|, 1 \leq t \leq T \tag{3.9}$$

Nous assurons aussi que la bande passante totale allouée à la migration d'une réplique de la partition p_j du serveur s_i au serveur s_k ne dépasse pas la taille de la partition p_j, dans la contrainte (3.10))

$$\sum_{t=1}^{T} r_{i,j,k,t} \cdot \alpha \leq y_{j,k} \cdot |p_j| \quad \forall 1 \leq i \leq |S|, 1 \leq t \leq T \tag{3.10}$$

La charge dans un lien ne doit pas dépasser la capacité de ce lien. Ainsi, la somme de toutes les bandes passantes allouées aux migrations passant par le lien e ne doit pas dépasser la capacité de ce lien. La contrainte (3.11) illustre ceci.

$$\sum_{i=1}^{|S|}\sum_{j=1}^{|P|}\sum_{k=1}^{|S|} g_{i,k,e} \cdot r_{i,j,k,t} \leq B_e$$

$$\forall 1 \leq e \leq |E|, 1 \leq t \leq T \tag{3.11}$$

Le temps total de la migration inclut toutes les migrations en cours dans la contrainte (3.12). Et, la contrainte (3.13) arrête le processus de migration.

$$x_{i,j,k,t} \leq w_t$$

$$\forall 1 \leq i,k,\ i \neq k \leq |S|, 1 \leq j \leq |P|, 1 \leq t \leq T \tag{3.12}$$

$$w_{t+1} \leq w_t \quad \forall 1 \leq t \leq T-1 \tag{3.13}$$

3.2.3 La fonction objectif

$$minimize \left(\sum_{t=1}^{T} w_t + \sum_{i=1}^{|S|} \sum_{j=1}^{|P|} \sum_{k=1}^{|S|} \sum_{t=1}^{T} x_{i,j,k,t} \right) \tag{3.14}$$

Comme illustré dans (3.14), notre objectif consiste à minimiser d'une part le temps de migration individuel de chaque réplique de partition. Ceci a pour but de rendre cette partition rapidement accessible dans son emplacement final et optimal. En réduisant au minimum le temps de migration totale, nous répliques seront migrées plus tôt. D'autre part, notre objectif vise aussi à minimiser le temps total pris par la migration de toutes les répliques. Ceci va initier le processus de migration de chaque réplique le plus tôt possible. Étant donnée que notre objectif vise à minimiser les temps de migration, notre modèle sélectionne les serveurs sources pour la migration qui réduisent le temps de migration tout en s'assurant de ne pas congestionner le réseau. Notre modèle garantit aussi une migration séquentielle des répliques respectant le seuil minimal de la disponibilité d'une partition.

3.3 CRANE : Un plan pour la migration des répliques de données dans les systèmes de stockage distribués

Le problème de migration de répliques est un problème NP-complet (Tziritas *et al.*, 2008). Sa résolution est limitée à un petit nombre de données. Cependant, les centres de données hébergent une quantité colossale de données. Ceci nous a poussé à proposer une heuristique permettant la résolution à haute échelle de ce problème. Dans cette section, nous décrivons CRANE, notre heuristique pour le problème de la migration des répliques de données. À partir d'une configuration initiale et finale du placement des répliques dans les centres de données, notre algorithme trouve les meilleures sources pour la copie des répliques de partitions et la meilleure séquence de migration de manière à minimiser le

temps de migration total. Pour répondre à cet objectif, notre algorithme évite les migrations redondantes afin d'avoir plus de bande passante disponible pour la migration d'autres répliques. De plus, dans le but d'accélérer le processus de migration, notre heuristique sélectionne les serveurs et les chemins ayant plus de bande passante disponible. Finalement, CRANE évite le temps d'inactivité et veille à la migration séquentielle des répliques.

Notre solution heuristique est illustrée dans l'algorithme 25. À partir d'une configuration de placement initiale et finale (i.e., C^I et C^F), l'algorithme 25 retourne un ensemble Q de séquences Q_i de migrations. Chaque séquence contient un ensemble ordonné de répliques à copier/migrer de manière à satisfaire la disponibilité minimale par partition. Après chaque séquence de migrations Q_i, les composantes de stockage du cloud seront mises à jour avec le nouvel emplacement des répliques pour que l'utilisateur final soit redirigé au bon emplacement de la partition. La configuration finale est atteinte une fois que toutes les séquences $Q_i, i < n$, sont exécutées.

Initialement, P contient l'ensemble des couples (p, d), où p est une partition qui doit être créée/migrée dans le serveur de destination d. Cet ensemble peut être extrait à partir de la configuration initiale et finales du placement des répliques (à savoir, C^I et C^F). Nous initialisons aussi Q qui est destiné à contenir la séquence de répliques à migrer. Dans la ligne 3 de l'algorithme, nous initialisons la variable i qui représente le nombre de la séquence. Notre algorithme vise ensuite à ajouter itérativement une réplique de partition à la séquence ordonnée Q_i. En effet, nous créons une nouvelle séquence à chaque fois qu'il n'est plus possible d'ajouter aucune migration à la séquence actuelle. Cet ajout peut ne pas être possible pour ne pas enfreindre la disponibilité minimale des partitions. P_i représente l'ensemble des couples (p, d) qui peuvent être créés dans la séquence Q_i sans violer la disponibilité minimale des répliques requise.

Algorithme 3 CRANE

Entrée : Configuration initiale C^I

Entrée : Configuration finale C^F

Sortie : Sequence de migration

1: $P \leftarrow \{(p, d)\}$
2: $Q \leftarrow \{\emptyset\}$
3: $i \leftarrow 0$
4: **while** $P \neq \{\emptyset\}$ **do**
5: $Q_i \leftarrow \emptyset$
6: $P_i \leftarrow \{P\}$
7: **while** $P_i \neq \emptyset$ **do**
8: $T_{Q_i,min} \leftarrow \infty$
9: **for chaque** $(p, d) \in P_i$ **do**
10: **for chaque** $r_{src} \in R_p$ / $\{is\_busy(src, Q_i) = False\}$ **do**
11: **if** $T_{Q_i,r_{src}} < T_{Q_i,min}$ **then**
12: $T_{Q_i,min} = T_{Q_i,r_{src}}$
13: $r_s = r_{src}$
14: $d_s = d$
15: **end if**
16: **end for**
17: **end for**
18: $Q_i = Q_i \cup (r_s, d_s)$
19: $P_i = P_i - \{\forall(p, d)$ / p partition de la replique $r_s\}$
20: $P = P - \{$partition p de la réplique r_s to destination $d\}$
21: **end while**
22: $Q = Q \cup Q_i$
23: $i \leftarrow i + 1$
24: **end while**
25: **retourner** Q

Tant que P_i contient encore des partitions qui puissent être migrées, nous parcourons l'ensemble R_p de toutes les répliques de la partition p de l'ensemble P_i. De plus le serveur détenant la réplique r ne doit pas copier une autre réplique dans le même temps la réplique r sera copié à partir de ce serveur (ligne 10).

Dans notre algorithme, nous choisissons la réplique r_s et le serveur de destination d_s qui minimisent le temps de migration. Pour cela, nous utilisons la variable $T_{Q_i,r_{src}}$ qui indique le temps de migration de la séquence Q_i si la réplique r_{src} est incluse. $T_{Q_i,r_{src}}$ est calculé en considérant l'état des liens lors de la migration de toutes les répliques dans l'ordre de Q_i. Nous comparons cette variable à la variable $T_{Q_i,min}$ qui représente le temps de migration minimum de séquence Q_i après l'ajout d'une réplique (ligne 11). Cela nous permet de sélectionner la meilleure réplique r_s de la partition p de toutes les répliques de toutes les partitions (ligne 13). La réplique et le serveur de destination choisis sont ensuite ajoutés à Q_i (ligne 17). Ensuite, nous retirons le couple (p, d_s) de l'ensemble P, où p est la partition dont réplique r_s a été sélectionné et d_s est le serveur de destination où la partition sera créée. Nous supprimons également tous les couples (p, d) où p est la partition dont la réplique r_s a été sélectionnée de P_i.

Lorsque P_i ne contient plus aucun couple à migrer, nous concluons que nous ne pouvons plus ajouter de répliques à la séquence Q_i. À ce moment-là, nous ajoutons la séquence Q_i à Q (Ligne 22), et nous entamons une nouvelle séquence. Ceci sera répété aussi longtemps que nous avons encore des partitions de P pour migrer.

3.4 Évaluation des performances

Nous avons implémenté CRANE en utilisant le langage Python 2.7 et nous avons évalué ses performances en deux phases. Nous avons comparé les performances de CRANE, Openstack Swift et la séquence de migration optimale. Ensuite, nous avons mené des expériences pour comparer les performances de CRANE avec les

performances d'Openstack Swift.

Pour évaluer les performances de notre heuristique, nous avons utilisé la topologie du réseau reliant les centres de données de Amazon EC2, qui est composée de 7 centres de données qui sont tous reliés les uns aux autres. Dans (Feng *et al.*, 2012) les auteurs ont effectué des expériences réalistes pour mesurer la capacité des liens entre les centres de données de EC2 tableau 3.3 illustre ces capacités.

Tableau 3.3 Capacité des liens entre les centres de données de Amazon EC2

Link capacity (Mbps)	**North California**	**Oregon**	**Virginia**	**Sao Paulo**	**Ireland**	**Singapore**	**Tokyo**
North California	-	520	252	116	98	103	173
Oregon	545	-	215	81	104	81	152
Virginia	240	210	-	139	221	81	110
Sao Paulo	40	60	11	-	22	9	62
Ireland	106	135	215	90	-	77	76
Singapore	125	110	84	57	80	-	242
Tokyo	178	143	99	61	43	116	-

Nous avons considéré plusieurs scénarios, chacun ayant un nombre différent de partitions et différentes moyennes de taille des partitions. Au début de chaque expérience, nous considérons seulement 5 centres de données. Après, deux nouveaux centres de données sont reliés à l'infrastructure, ce qui déclenche

l'algorithme de placement afin de ré-optimiser l'emplacement des répliques. Pour le placement de répliques, nous avons utilisé l'algorithme standard d'Openstack Swift énonçant que pour chaque partition de données, trois répliques doivent être créées et placées selon l'algorithme *as-unique-as-possible*. En outre, Swift considère que si 2 des 3 répliques sont disponibles dans leurs emplacements, les données sont supposées être disponibles (à savoir, toutes les demandes des utilisateurs peuvent être servies). Par conséquent, la disponibilité minimale requise par partition est $\frac{2}{3}$.

Les résultats expérimentaux ont validé, que en optimisant l'ordre de migrer les répliques, CRANE réduit le temps de migration et la quantité de données à migrer.

3.4.1 Évaluation par simulations

Pour obtenir les résultats analytiques, nous avons modélisé le cas particulier du problème de la migration des répliques qui assure la disponibilité minimale d'une partition, comme un problème de programmation linéaire en nombres entiers (PLNE). Pour ce faire, nous avons utilisé le langage AMPL (AMP, 2016). Puis, nous avons utilisé IBM CPLEX Optimizer (Cpl, 2016) pour trouver la séquence de migration optimale qui réduit le temps de migration. Nous avons considéré six scénarios illustrés dans le tableau 3.4. Par ailleurs, dans ces différents scénarios, nous considérons que nous avons 3 répliques pour chaque partition. En outre, dans les quatre premiers scénarios, la taille des partitions varie entre 500Mo et 1Go. Nous décrirons par la suite ces partitions comme petites partitions. Et, pour les quatre derniers scénarios, la taille des partitions varie entre 1 Go et 5 Go. Nous décrirons par la suite ces partitions comme grandes partitions. Nous distribuons ces partitions à travers les 5 centres de données ; ce placement décrira la configuration de placement de la réplique initiale. Ensuite, nous ajoutons deux nouveaux centres de données, et déclenchons l'algorithme de placement pour

avoir le placement final et optimal des répliques. Nous avons ensuite comparé les performances théoriques de CRANE et Swift aux performances de la solution optimale pour chaque scénario, compte tenu des placements initiaux et finaux.

Pour assurer la disponibilité minimale de chaque partition, Openstack Swift déplacer une seule réplique d'une partition à chaque nouveau placement des répliques. S'il y a besoin de déplacer d'autres répliques, il faut exécuter de nouveau l'algorithme de placement des répliques. Ceci peut être exécuté après plus d'une heure. Ayant d'énormes quantités de données à migrer, nous avons évalué les performances des trois plans de migrations en exécutant l'algorithme de placement au bout d'une heure et au bout de deux heures. L'exécution des algorithmes de placement calcule le nouveau placement de répliques et de déclenche ensuite l'algorithme de migration.

Tableau 3.4 Scénarios considérés

Scenario	**Nombre de partitions**	**Nombre de répliques à migrer**	**Taille moyenne des répliques (Mb)**	**Fréquence de calcul du placement**
Sc1	16	16	750	1 heure
Sc2	32	32	750	1 heure
Sc3	64	80	750	1 heure
Sc4	128	160	750	1 heure
Sc5	16	16	3074	2 heures
Sc6	32	32	3074	2 heures
Sc7	64	80	3074	2 heures
Sc8	128	160	3074	2 heures

Les figures 3.1(a) et 3.1(c) représentent le temps de migration et la quantité de

données transférées respectivement pour les quatres premiers scénarios. Dans le scénario Sc1, Swift, CRANE et la séquence de migration optimale, ont pris 8 min pour créer 16 nouvelles répliques. En outre, la quantité de données transférées est la même. Pour Sc2, afin de migrer 32 partitions, la quantité totale de données transférées a été de 25Gb. La séquence de migration optimale a duré 2 minutes moins que CRANE et Swift. Pour Sc3, CRANE et la séquence de migration des répliques optimale, ont transféré la même quantité de données, alors que Swift a transféré 30 % plus de données. Cependant, la séquence de migration des répliques optimale a un temps de migration meilleur que CRANE et Swift de 15 % et 35 % respectivement. Pour Sc4, Swift transfère 40 % plus de données que CRANE et la séquence de migration optimale, en deux fois le temps de migration optimale. Cependant CRANE a transféré un quantité de données optimale et a également amélioré le temps de migration de 30 % par rapport à Swift.

Les figures 3.1(b) et 3.1(d) représentent le temps de migration et la quantité de données transférées, respectivement, pour les quatres derniers scénarios. Dans ces scénarios, la taille des réplique est plus grande variant de 1Go à 5Go. Figure 3.1(d) montre que CRANE transfère une quantité optimale de données, pour les 4 scénarios. Cependant Swift transfère 15 % plus de répliques pour les scénarios Sc 5 et Sc6, et 35 % et 45 % plus de données dans les scénarios Sc7 et Sc8 respectivement. Cela a également induit un plus grand temps de migration, qui s'élève à 35 % de plus que le temps mis par la séquence de migration optimale pour les scénarios Sc5 et Sc6, et 45 % et 60 % plus que le temps mis par la séquence de migration optimale pour les scénarios Sc7 et Sc8 respectivement. En revanche CRANE réalise un temps de migration de répliques 15 % plus que le temps mis par la séquence de migration optimale pour les scénarios Sc5 et Sc6, et environ 20 % pour Sc7 et 30 % Sc8.

À partir de ces différents scénarios, nous pouvons conclure que le temps de

migration de CRANE est d'environ 20 % proche du temps mis par la séquence de migration optimale. Et réalise une amélioration d'environ 40 % par rapport au temps de migration de Swift.

3.4.2 Évaluation sur une plateforme réelle

Mise en place de l'environnement

Le déploiement d'Openstack Swift requiert au moins un *nœud proxy* et plusieurs *nœuds de stockage*. Le *nœud proxy* accepte les demandes de téléchargement des fichiers, de modification des métadonnées, et de création des conteneurs. Les *nœuds de stockage* gèrent, quant à eux, les objets et les conteneurs, d'une manière distribuée. Pour nos expériences, nous étions intéressés par le processus de migration de répliques. Nous avons utilisé la plateforme logicielle open-source pour le cloud computing Openstack (Ope, 2016) pour créer notre infrastructure. Ainsi, nous considérons que chaque *nœud de stockage* est une machine virtuelle hébergée dans un centre de données différent. En effet, nous avons lancé 8 machines virtuelles se basant sur Ubuntu 14.04 ayant chacune 2 unités de calcul et une mémoire de 2 Gb. Nous avons lié un disque de 200Go à chaque instance. Chaque instance est hébergée dans un réseau virtuel différent et nous avons lié ces réseaux avec un routeur virtuel. Ensuite, nous avons fixé les capacités en bande passante des liens entre ces différents réseaux en configurant les interfaces du routeur. Le Tableau 3.3 décrit ces capacités.

Au début de chaque scénario, nous ne considérons que 5 centres de données. Après, deux nouveaux centres de données sont reliés à l'infrastructure, ce qui déclenche l'algorithme de placement de Swift afin de ré-optimiser l'emplacement des répliques. Ensuite, nous utilisons soit l'algorithme de migration de Swift ou CRANE pour migrer les répliques aux emplacements optimaux calculés. Nous avons déployé quatre scénarios. Dans chaque scénario, nous considérons 64

partitions avec 3 répliques chacune comme recommandé par certains fournisseurs OpenStack (Rackspace, 2016). Cependant la taille moyenne des répliques varie d'un scénario à un autre. Le Tableau 3.5 décrit ces scénarios.

Tableau 3.5 Scénarios déployés dans les expérimentations

Scénario	**Nombre de partitions**	**Taille moyenne des répliques (Gb)**
Sc9	64	3
Sc10	64	10
Sc11	64	15

Pour chaque scénario, représenté dans le Tableau 3.5, nous comparons CRANE avec Swift par rapport aux métriques de performance suivantes : (1) le temps de migration totale, (2) la quantité de donnée transférée entre les centres de données, (3) le temps d'inactivité et (4) la disponibilité de répliques par partition. Nous avons considéré deux configurations de Swift, l'une caculant le placement des répliques chaque heure et l'autre toutes les deux heures.

3.4.3 Résultats expérimentaux

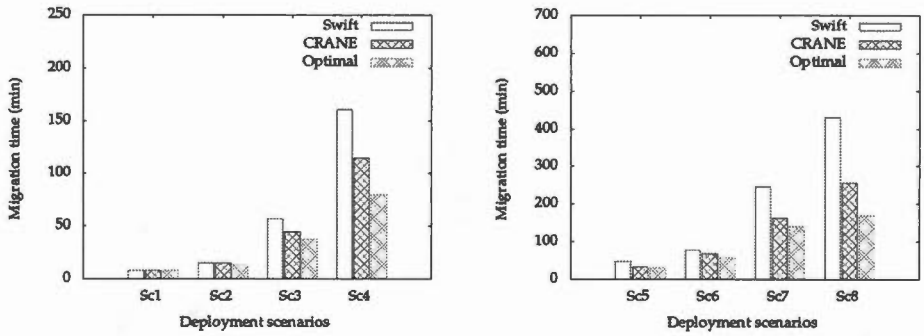
La Figure 3.2(a) indique la durée totale de la migration pour chacun des scénarios considérés. Comme nous pouvons le voir, dans tous les scénarios, CRANE surpasse Swift par une bonne marge pour les deux configurations de Swift. Pour le scénario Sc9, CRANE prend 50 minutes pour la création de toutes les répliques contre 75 minutes pour l'algorithme Swift de 1 heure et 132 minutes pour l'algorithme Swift de 2 heures . Ceci constitue environ 35 % et 65 % d'amélioration comparée à l'algorithme Swift de 1 heure et 2 heures respectivement.

Pour les scénarios Sc10 et Sc11, CRANE réalise également 30 % et 50 % des améliorations par rapport à l'algorithme de Swift configuré à 1 heure et à 2 heures pour le calcul du placement de répliques respectivement. Ces résultats sont prévus, car CRANE choisit toujours de copier la réplique qui minimise le temps de migration total. Comme le temps de migration représente le temps que prennent les serveurs pour transférer toutes les répliques vers leurs nouveaux emplacements, ce temps est également influencé par la durée d'inactivité. Comme le montre la figure 3.2(c), le temps d'inactivité pour CRANE est toujours nul. Cependant, le temps d'inactivité est plus élevé pour les configurations de 1 heure et de 2 heures de Swift. En effet, pour le Sc9, on voit que la configuration de 2 heures de Swift atteint un temps d'inactivité de 85 minutes. Comme la quantité de données à migrer peut être transférée rapidement, le système est resté inactif pendant 85 minutes. Pendant ce temps, les répliques ne sont pas dans leurs emplacements optimaux . Cependant, plus la quantité de données transférée est élevée, et plus le temps d'inactivité est faible.

La quantité de données transférée entre les centres de données est rapportée dans la figure 3.2(b). Pour les 3 différents scénarios , CRANE transfère moins de données que les deux configurations de Swift. L'amélioration est d'environ 25 % pour la configuration de Swift de 1 heure et 45 % pour la configuration de Swift de 2 heures. Cette amélioration est expliquée par le fait que CRANE évite les migrations redondantes des répliques de la même partition. Ceci est également expliqué par le fait que Swift pour la configuration de 1 heure et de 2 heures calcule un nouveau placement des répliques chaque heure et chaque 2 heures respectivement. A chaque nouveau placement des répliques, celles-ci sont placées suivant l'algorithme *as-unique-as-possible* dans les disques. Cela induit à des migrations inutiles qui sont juste déclenchées afin d'équilibrer les répliques sur des disques suivant cet algorithme même si ce placement ne constitue pas

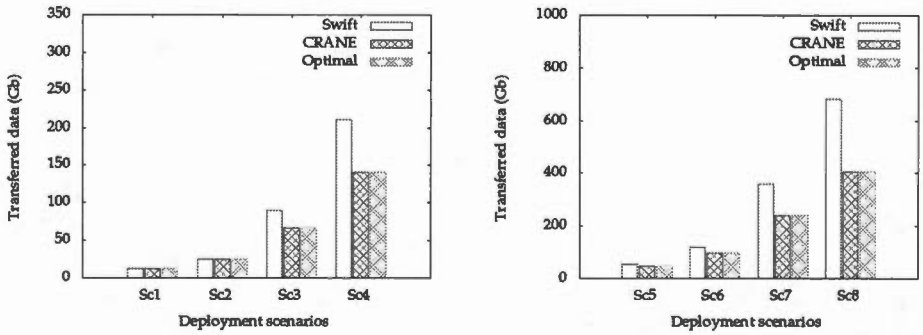
un placement optimal. Cela a également amené à l'amélioration du temps de migration de CRANE comme montré dans la figure 3.2(a).

Finalement, la figure 3.2(d) montre la fonction inverse de la distribution cumulative (ICDF) de la disponibilité. Pour une disponibilité donnée, cette fonction fournit la probabilité d'avoir cette la disponibilité ou plus. La disponibilité minimale requise par partition ($2/3 = 0{,}66$) n'a pas été satisfaite pour la configuration de Swift de 1 heure. La probabilité d'avoir une disponibilité supérieure à 66 % est de 0,98. L'unique heure qui sépare les deux nouveaux placements de répliques n'a pas été suffisante pour assurer la disponibilité. En effet, il existe deux répliques de la même partition qui ne sont pas encore arrivées à leurs destinations finales en même temps. De plus, la probabilité d'avoir une meilleure disponibilité est toujours plus élevée pour CRANE . Par exemple, la probabilité d'avoir une disponibilité supérieure à 80 % est 0,60 pour Swift alors qu'il est d'environ 0,76 CRANE. En comparant les trois courbes, nous pouvons voir qu'en moyenne, CRANE améliore la disponibilité d'une valeur allant jusqu'à 10 %.



(a) Temps de migration pour les petites partitions

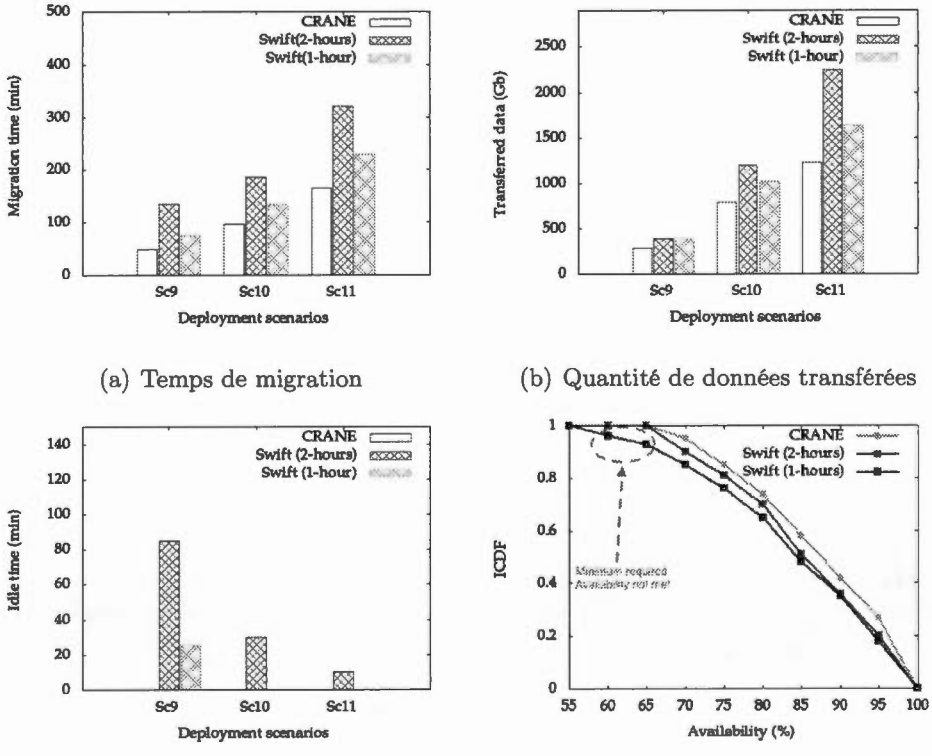
(b) Temps de migration pour les grandes partitions



(c) Quantité de données transférées pour les petites partitions

(d) Quantité de données transférées pour les grandes partitions

Figure 3.1 Comparaison des performances analytiques de la séquence de migration optimale, CRANE et Swift



(a) Temps de migration

(b) Quantité de données transférées

(c) Temps d'inactivité

(d) Disponibilié

Figure 3.2 Comparaison expérimentale des performances entre CRANE et Swift

CONCLUSION

Les réseaux de diffusion de contenu permettent aux utilisateurs d'accéder rapidement au contenu, quel que soit son emplacement et de réduire les temps de latence liée à la distance séparant le serveur d'hébergement et l'utilisateur final. De plus, l'avènement du provisionnement élastique des ressources dans l'informatique en nuage a révélé être une solution rentable pour les fournisseurs de CDN, qui peuvent désormais louer des ressources pour le stockage, le calcul, et la bande passante, menant à la construction des CDN basés sur le cloud appelé CCDN. En effet, en plaçant le contenu dans le cloud, les fournisseurs de CDN augmentent la disponibilité et réduisent le temps d'accès au contenu, tout en minimisant le coût de stockage des données. Cependant les modèles de calcul de coût sont différents pour les CDN et les CCDN.

D'autre part, l'augmentation de l'utilisation des services de vidéos à la demande, et du contenu généré par les utilisateurs a apporté de nouveaux types de relations entre le contenu lié à la corrélation et à la popularité du contenu. Ainsi, la différence entre les modèles de calcul de coût ainsi que la relation complexe qui lie les vidéos générées par les utilisateurs empêchent l'utilisation des algorithmes de placement de contenu des CDN aux CCDN. A cet effet, nous proposons une stratégie de placement du contenu vidéo hébergé dans les CCDN et une stratégie pour minimiser le temps de migration de ce contenu entre les différents centres de données.

Notre approche pour le placement de contenu permet de réduire le coût de l'hébergement de contenu tout en offrant un contenu avec un délai de réponse

borné aux requêtes des utilisateurs. Nous avons modélisé le problème comme un programme linéaire en nombres entiers et nous avons proposé une heuristique basée sur l'optimisation par essaims particulaires pour le résoudre.

L'analyse des performances de notre approche nous a permis de démontrer la faisabilité de notre architecture hiérarchique et ses avantages pour la livraison de contenus dans les CCDN. De plus nous avons comparé les performances de la solution optimale liée à notre stratégie ainsi que celles de l'heuristique proposée avec d'autres stratégies de placement largement utilisés dans les travaux précédents. Ceci nous a permis de conclure que notre stratégie constitue un compromis entre la stratégie considérant uniquement le coût de placement et celle visant uniquement à améliorer la popularité des données.

Dans la seconde partie de ce mémoire, nous nous sommes intéressés au problème de la migration des répliques de données entre des systèmes de stockage distribués. Notre contribution a été dans un premier temps de mettre l'accent sur les limitations des solutions existantes de migration de données. Ensuite, nous avons formulé en programmation linéaire en nombre entier le problème de migration de répliques dans un système de stockage distribué. Nous avons aussi proposé une heuristique baptisée CRANE. En effet, CRANE complète n'importe quelle stratégie de placement en gérant efficacement la création de répliques en minimisant le temps nécessaire pour copier les données vers leur nouvel emplacement tout en évitant la congestion du réseau et en assurant la disponibilité minimale requise pour les données. Nous proposons finalement une évaluation des performances de CRANE. Pour ce faire, nous avons exécuté des expérimentations réelles et des simulations réalistes pour de grandes infrastructures de cloud. Nous avons ainsi implémenté CRANE et nous l'avons intégré dans le système de gestion de données d'Openstack. Nous avons utilisé cette implémentation pour comparer les performances de CRANE avec celles d'Openstack Swift pour

différents scénarios. Ensuite, nous comparons les performances de la solution trouvée avec CRANE avec celles de la solution optimale. Les résultats montrent que CRANE présente des performances sous-optimales en termes de temps de migration et des performances optimales pour la quantité de données transférées.

RÉFÉRENCES

(2016). *AMPL : STREAMLINED MODELING FOR REAL OPTIMIZATION.* AMPL Optimization. Récupéré de `http://ampl.com`

(2016). *IBM CPLEX Optimizer.* IBM. Récupéré de `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

(2016). *Infrastructure mondiale AWS.* Amazon. Récupéré de `https://aws.amazon.com/fr/about-aws/global-infrastructure/`

(2016). *Openstack Cloud Computing Software.* Openstack foundation. Récupéré de `http://www.openstack.org`

Akjiratikarl, C., Yenradee, P. et Drake, P. R. (2007). Pso-based algorithm for home care worker scheduling in the uk. *Computers & Industrial Engineering*, *53*(4), 559–583.

Anderson, E., Hall, J., Hartline, J., Hobbes, M., Karlin, A., Saia, J., Swaminathan, R. et Wilkes, J. (2010). Algorithms for data migration. *Algorithmica*, *57*(2), 349–380.

Anttonen, A. et Mämmelä, A. (2014). Interruption probability of wireless video streaming with limited video lengths. *IEEE Transactions on Multimedia*, *16*(4), 1176–1180.

Benoit, A., Rehn, V. et Robert, Y. (2007). Impact of qos on replica placement in tree networks. Dans *International Conference on Computational Science*, 366–373. Springer.

Broberg, J., Buyya, R. et Tari, Z. (2009). Metacdn : Harnessing 'storage clouds' for high performance content delivery. *Journal of Network and Computer Applications*, *32*(5), 1012–1022.

Chen, F., Guo, K., Lin, J. et La Porta, T. (2012). Intra-cloud lightning : Building cdns in the cloud. Dans *INFOCOM, 2012 Proceedings IEEE*, 433–441. IEEE.

Chen, Y., Qiu, L., Chen, W., Nguyen, L. et Katz, R. H. (2003). Efficient and adaptive web replication using content clustering. *IEEE Journal on Selected Areas in Communications*, *21*(6), 979–994.

Cisco, I. (2012). Cisco visual networking index : Forecast and methodology, 2011–2016. *CISCO White paper*, 2011–2016.

Clerc, M. et Kennedy, J. (2002). The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE transactions on Evolutionary Computation*, *6*(1), 58–73.

Dickinson, J. (2013). *Data Placement in Swift.* Swiftstack. Récupéré de `https://swiftstack.com/blog/2013/02/25/data-placement-in-swift/`

Feng, Y., Li, B. et Li, B. (2012). Jetway : Minimizing costs on inter-datacenter video traffic. Dans *Proceedings of the 20th ACM international conference on Multimedia*, 259–268. ACM.

Fink, J., Koenemann, J., Noller, S. et Schwab, I. (2002). Putting personalization into practice. *Communications of the ACM*, *45*(5), 41–42.

foundation, O. (2015). *Swift documentation.* Openstack foundation. Récupéré de `http://docs.openstack.org/developer/swift/`

Fujita, N., Ishikawa, Y., Iwata, A. et Izmailov, R. (2004). Coarse-grain replica management strategies for dynamic replication of web contents. *Computer Networks*, *45*(1), 19–34.

Hall, J., Hartline, J., Karlin, A. R., Saia, J. et Wilkes, J. (2001). On algorithms for efficient data migration. Dans *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, 620–629. Society for Industrial and Applied Mathematics.

Herbst, N. R., Kounev, S. et Reussner, R. H. (2013). Elasticity in cloud computing : What it is, and what it is not. Dans *ICAC*, 23–27.

Holyer, I. (1981). The np-completeness of edge-coloring. *SIAM Journal on computing*, *10*(4), 718–720.

Jamin, S., Jin, C., Kurc, A. R., Raz, D. et Shavitt, Y. (2001). Constrained mirror placement on the internet. Dans *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, 31–40. IEEE.

Kangasharju, J., Roberts, J. et Ross, K. W. (2002). Object replication strategies in content distribution networks. *Computer Communications*, *25*(4), 376–383.

Kari, C., Kim, Y.-A. et Russell, A. (2011). Data migration in heterogeneous storage systems. Dans *IEEE International Conference on Distributed*

Computing Systems (ICDCS), 143–150.

Kennedy, J. et Eberhart, R. (1995). Particle swarm optimization. Dans *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, 1942–1948. IEEE.

Khuller, S., Kim, Y.-A. et Wan, Y.-C. J. (2003). Algorithms for data migration with cloning. Dans *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS.

Kim, Y.-A. (2005). Data migration to minimize the total completion time. *Journal of Algorithms*, *55*(1), 42–57.

Krishnan, P., Raz, D. et Shavitt, Y. (2000). The cache location problem. *IEEE/ACM Transactions on Networking (TON)*, *8*(5), 568–582.

Laoutaris, N., Zissimopoulos, V. et Stavrakakis, I. (2005). On the optimization of storage capacity allocation for content distribution. *Computer Networks*, *47*(3), 409–428.

Li, B., Golin, M. J., Italiano, G. F., Deng, X. et Sohraby, K. (1999). On the optimal placement of web proxies in the internet. Dans *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, 1282–1290. IEEE.

Lin, C.-F., Leu, M.-C., Chang, C.-W. et Yuan, S.-M. (2011). The study and methods for cloud based cdn. Dans *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on*, 469–475. IEEE.

Loukopoulos, T., Tziritas, N., Lampsas, P. et Lalis, S. (2006). Investigating the replica transfer scheduling problem. Dans *Proc. 18 th Int. Conf. on Parallel and Distributed Computing and Systems (PDCS'06)*. Citeseer.

Loukopoulos, T., Tziritas, N., Lampsas, P. et Lalis, S. (2007). Implementing replica placements : feasibility and cost minimization. Dans *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, 1–10. IEEE.

Luan, T. H., Cai, L. X. et Shen, X. (2010). Impact of network dynamics on user's video quality : analytical framework and qos provision. *IEEE Transactions on Multimedia*, *12*(1), 64–78.

Mseddi, A., Salahuddin, M. A., Zhani, M. F., Elbiaze, H. et Glitho, R. H. (2015). On optimizing replica migration in distributed cloud storage systems. *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, 191–197.

Papagianni, C., Leivadeas, A. et Papavassiliou, S. (2013). A cloud-oriented content delivery network paradigm : Modeling and assessment. *IEEE Transactions on Dependable and Secure Computing*, *10*(5), 287–300.

Petrov, D. L. et Tatarinov, Y. S. (2009). Data migration in the scalable storage cloud. Dans *Ultra Modern Telecommunications & Workshops, 2009. ICUMT'09. International Conference on*, 1–4. IEEE.

Rackspace. (2016). *Swift ring calculator*. Rackspace. Récupéré de `https://rackerlabs.github.io/swift-ppc/`

Salahuddin, M. A., Elbiaze, H., Ajib, W. et Glitho, R. (2015). Social network analysis inspired content placement with qos in cloud based content delivery networks. Dans *2015 IEEE Global Communications Conference (GLOBECOM)*, 1–6. IEEE.

Srinivasan, S. R., Lee, J. W., Batni, D. L. et Schulzrinne, H. G. (2011). Activecdn : Cloud computing meets content delivery networks. *Department of Computer Science, Columbia University.*

Tewari, S. et Kleinrock, L. (2006). Proportional replication in peer-to-peer networks. Dans *INFOCOM.*

Tu, M. et Yen, I.-L. (2014). Distributed replica placement algorithms for correlated data. *The Journal of Supercomputing*, *68*(1), 245–273.

Tziritas, N., Loukopoulos, T., Lampsas, P. et Lalis, S. (2008). Formal model and scheduling heuristics for the replica migration problem. In *Euro-Par 2008–Parallel Processing* 305–314. Springer.

Tziritas, N., Loukopoulos, T., Lampsas, P. et Lalis, S. (2009). Using multicast transfers in the replica migration problem : Formulation and scheduling heuristics. In *Euro-Par 2009 Parallel Processing* 228–240. Springer.

Wang, M., Jayaraman, P. P., Ranjan, R., Mitra, K., Zhang, M., Li, E., Khan, S., Pathan, M. et Georgeakopoulos, D. (2015). An overview of cloud based content delivery networks : research dimensions and state-of-the-art. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XX* 131–158. Springer.

Wu, B. et Kshemkalyani, A. D. (2006). Objective-optimal algorithms for long-term web prefetching. *IEEE Transactions on Computers*, *55*(1), 2–17.

Wu, Y., Zhang, Z., Wu, C., Guo, C., Li, Z. et Lau, F. (2015). Orchestrating bulk data transfers across geo-distributed datacenters. *Cloud Computing, IEEE Transactions on*, *PP*(99), 1–1. `http://dx.doi.org/10.1109/TCC.2015.2389842`

Xu, D., Kulkarni, S. S., Rosenberg, C. et Chai, H.-K. (2006). Analysis of a cdn–p2p hybrid architecture for cost-effective streaming media distribution. *Multimedia Systems*, *11*(4), 383–399.

Yu, X.-m., Xiong, X.-y. et Wu, Y.-w. (2004). A pso-based approach to optimal capacitor placement with harmonic distortion consideration. *Electric Power Systems Research*, *71*(1), 27–33.

Zeni, M. (2016). *THE YOUSTATANALYZER DATABASE.* CONGAS. Récupéré de `http://www.congas-project.eu/sites/www.congas-project.eu/files/Dataset/docs/youstatanalyzer.html`

Zeni, M., Miorandi, D. et De Pellegrini, F. (2013). YOUStatAnalyzer : a tool for analysing the dynamics of YouTube content popularity. Dans *Proc. 7th International Conference on Performance Evaluation Methodologies and Tools (Valuetools, Torino, Italy, December 2013)*, Torino, Italy.

Zhou, Y., Chen, L., Yang, C. et Chiu, D. M. (2015). Video popularity dynamics and its implication for replication. *IEEE transactions on multimedia*, *17*(8), 1273–1285.