TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

NIST	National Institute of Standards and Technology
CPU	Central Processing Unit
SLA	Service Level Agreement
PEPS	Premier Entré Premier Servi
WAN	Wide Area Network
TCP	Transmission Control Protocol
API	Application Programming Interface
HTTP	HyperText Transfer Protocol
REST	REpresentational State Transfer
BP	Bande Passante
PLNE	Programme Linéaire en Nombres Entiers
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service

LISTE DES SYMBOLES ET UNITÉS DE MESURE

s	Seconde
GB	Giga Byte

INTRODUCTION

Contexte

De nombreuses applications à grande échelle en sciences, en ingénierie et en domaine d'affaires génèrent des quantités colossales de données, maintenant souvent appelé Big Data. Ces énormes quantités de données sont télé-versées chaque seconde sur de grandes plateformes de stockages. Un exemple de référence peut être le cas de YouTube dont des centaines d'heures de vidéos sont ajoutées aux serveurs chaque seconde. Nous pouvons estimer (en supposant que les vidéos sont en Haute Définition) à des centaines de téra-octets la quantité de données versée par jour [1].

Comme la quantité du contenu multimédia a tendance à augmenter, la demande pour le transfert de données massives est également en croissance continue à diverses fins telles que le stockage de données, le traitement et l'analyse. Ces besoins en récupération de données créent un défi quant à l'acheminement optimal de ce contenu vers l'utilisateur, et ce, avec une qualité de service définie au préalable. En fait, l'infonuage utilise des centres de données pour héberger les applications et les données. Le service de récupération des données est offert aux utilisateurs sur le modèle de paiement à l'utilisation tandis que la qualité de ce service est défini selon les accords de niveau de service également connus sous le nom de Service Level Agreement (SLA). Un des critères les plus importants dans ce contrat de service est la possibilité de récupérer des données dans un laps de temps bien défini.

Garantir la satisfaction des utilisateurs en termes de temps de transfert représente un défi majeur pour la communauté de gestion des données dans les systèmes de stockage de l'infonuage. Afin de satisfaire les accords de niveau de service (SLA), les transferts de données doivent être complétés avant une échéance spécifiée par l'utilisateur. De plus, les ressources WAN,

1. https ://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/

particulièrement les liens inter/intra centres de données, sont assez coûteuses, ce qui implique qu'elles devraient être pleinement utilisées.

Afin d'assurer le transfert de flux dans le réseau de l'infonuage, le contrôle de congestion TCP et l'ordonnancement Premier Entré Premier Sorti (PEPS) des requêtes de transferts de données sont actuellement les plus utilisés pour gérer les transferts de flux (Zhang *et al.* (2017)). Par conséquent, le concept d'ordonnancement des transferts de données et d'allocation efficace des ressources restent des problèmes ouverts. En effet, les ressources sont habituellement limitées. Cela implique, entres autres, un défi quant à la gestion des pics de demandes. Cependant, ils existent plusieurs problèmes ouverts dans ce champs de recherche. Les politiques d'ordonnancement récents sont généralement basées sur la minimisation du temps de transfert des données ou le nombre de transferts satisfaisant leurs échéances. Cependant, l'ordonnancement des requêtes des utilisateurs considérant leurs exigences reste un problème ouvert.

Problématique

Les services de stockage en nuage devraient satisfaire, autant que possible, les exigences de performance spécifiées dans le contrat d'accord de service (SLA), et plus précisément en termes de temps de récupération des données. Le temps de récupération de données est le temps perçu par l'utilisateur à partir du moment où il envoie une requête au système de stockage en nuage jusqu'au moment où il reçoit la donnée demandée. Des études récentes menées par Amazon, Microsoft et Google ont révélé qu'une légère augmentation de ce délai réduirait la satisfaction des utilisateurs et entraînerait donc une perte de revenus (Linden (2006), Schurman & Brutlag (2009)). Par exemple, des expériences menées sur Amazon ont montré qu'un délai de 100 ms pour une demande de récupération de données au cours du processus de présentation Web peut dégrader les ventes de 1% (Linden (2006)).

En même temps, fournir des garanties strictes sur le temps de récupération des données est toujours un défi non résolu en raison des goulots d'étranglement potentiels et imprévisibles dans le réseau, de la préemption possible par d'autres tâches, l'indisponibilité du serveur, les activités de mise à jour et de maintenance ainsi que les pannes imprévisibles (Sanjay *et al.* (2003)).

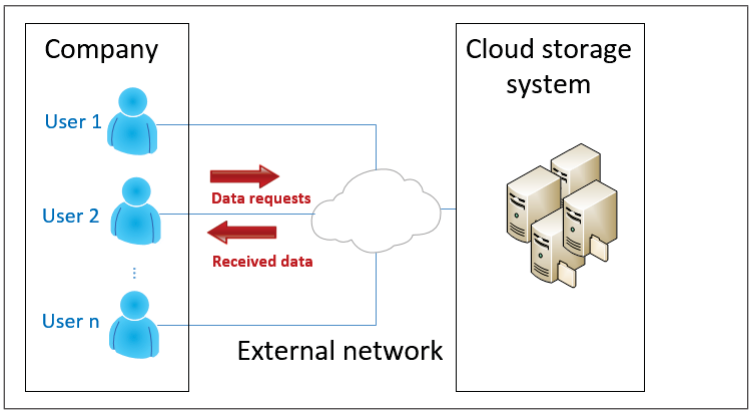

Figure 0.1 Mécanisme de récupération de données dans les systèmes de stockage traditionnels de l'infonuage.

À titre d'exemple, considérons une entreprise qui héberge ses données dans un système de stockage en nuage. La figure 0.1 illustre un aperçu de l'interaction entre les systèmes de stockage en nuage traditionnels et les utilisateurs qui sont les employés de cette entreprise. Les utilisateurs peuvent accéder aux serveurs de données via le réseau. Chaque utilisateur envoie une requête de récupération de données au système de stockage en cloud via des API de type REST. Comme nous considérons le cas d'un système de stockage en nuage traditionnel où les requêtes sont envoyées simultanément, le serveur de stockage les traitera dans un ordre Premier Entré Premier Servi (PEPS), ce qui peut pénaliser les requêtes hautement prioritaires. Le problème devient plus critique lorsque le serveur transfère d'énormes quantités de données sans tenir compte des priorités des requêtes.

Le temps de transfert requis par chaque utilisateur diffère de l'un à l'autre. Par exemple, certains utilisateurs peuvent demander la récupération de vidéos et documents sans aucune

échéance. Ces transferts ne sont pas critiques et peuvent être retardés. D'autres utilisateurs peuvent avoir besoin de récupérer une image de machine virtuelle de sauvegarde à partir du serveur pour une utilisation urgente avec une échéance stricte. Les requêtes de ce type doivent être entièrement satisfaites avant leurs échéances. Comme il est inutile qu'une donnée arrive à l'utilisateur après son échéance, la requête est rejetée car l'échéance ne peut pas être respectée. L'utilisateur est ainsi invité à renvoyer une nouvelle échéance.

Objectif et contributions

Pour résoudre ce problème, nous proposons d'agir du côté de l'entreprise en déployant un module de gestion de requêtes qui ordonnance intelligemment les requêtes des utilisateurs.

- Nous avons tout d'abord élaboré un module de surveillance de bande passante mesurée en temps réel entre le système de stockage et les utilisateurs afin d'estimer le temps de transmission de chaque fichier de données demandée.
- Nous avons, par la suite, développé une formulation mathématique du problème d'ordonnancement de requêtes à l'aide d'un programme linéaire à nombres entiers (PLNE) qui permet de trouver les décisions optimales d'ordonnancement de requêtes en déterminant quelle requête doit être traitée et à quel instant, tout en maximisant la consommation de la bande passante disponible et le taux de requêtes respectant leurs échéances.
- Finalement, nous avons implémenté un module d'ordonnancement de requêtes déployé du côté des utilisateurs pour planifier les requêtes des fichiers de données de manière dynamique dans le but de minimiser les délais de transfert des données à partir de Swift (le système de stockage de OpenStack) et de respecter les échéances spécifiées par les utilisateurs. Ces fonctionnalités sont mises en oeuvre à travers deux algorithmes baptisés DA et DA-Resch que nous proposons dans le présent travail.

Méthodologie du travail

Afin de répondre aux questions de la problématique et atteindre notre objectif, nous proposons une méthodologie subdivisée en quatre parties. D'abord, nous commençons par une exploration du contexte de notre recherche en mettant l'accent sur les concepts relatifs à l'infonuage et en particulier le stockage dans l'infonuage. Ensuite, nous élaborons une revue de littérature en présentant les solutions antérieures qui abordent des problématiques similaires à la nôtre. L'étape suivante consiste à proposer une solution pour l'ordonnancement des requêtes de transferts de données en tenant compte des échéances des requêtes et maximisant la consommation de la bande passante disponible avant leur implémentation et intégration dans Swift. Finalement, nous procédons à la phase de test et de validation du système proposé dans laquelle nous étudions l'impact de notre solution sur la performance de Swift notamment en termes de temps de récupération de données, taux de requêtes satisfaisant leurs échéances ainsi que celle du réseau en évaluant la consommation de bande passante durant les transferts.

Publications

Nous avons publié les résultats obtenus dans ce travail dans un article de conférence nommé «On Providing Deadline-Aware Cloud Storage Services» (Tlili, Ghada, Zhani, Mohamed Faten et Elbiaze, Halima, 2018) qui sera présenté à la conférence Innovation in Clouds, Internet and Networks (ICIN 2018) à Paris (France) en Février 2018.

Organisation du rapport

Le diagramme présenté dans la figure 0.2 illustre l'organisation de ce mémoire composé de quatre chapitres. Le premier chapitre présente une introduction générale sur l'infonuage détaillant les différentes notions relatives au stockage de données. Ce chapitre commence par une description générale des types de l'infonuage, suivi d'un aperçu sur le stockage dans l'in-

fonuage. Il se concentre ensuite sur Swift de OpenStack comme un exemple de service de stockage dans l'infonuage.

Dans le deuxième chapitre, nous présentons une revue de littérature qui se focalise sur les travaux existants traitant la gestion de transfert de données dans l'infonuage notamment la notion d'ordonnancement des requêtes de transferts de données. La dernière section de ce chapitre présente une étude comparative entre notre solution et les travaux existants.

Le troisième chapitre présente les détails de notre solution proposée. Nous commençons par une modélisation mathématique du problème. Nous présentons, par la suite, l'architecture de notre solution tout en discutant ses composantes et les principaux choix de conception qui ont été considérées afin d'atteindre les objectifs visés tels que la réduction du temps de transfert des données et la maximisation de la consommation de la bande passante disponible afin d'atteindre la qualité de service souhaitée des utilisateurs de l'infonuage.

Les résultats expérimentaux sont décrits dans le quatrième chapitre. Le chapitre commence par présenter notre environnement expérimental et les scénarios considérés. Ensuite, il se concentre sur l'étude de l'efficacité de note solution en évaluant les résultats obtenus en termes de taux de requêtes respectant leurs échéances, consommation de bande passante disponible ainsi que les retards enregistrés pour les différentes requêtes.

Ce rapport se termine par une conclusion générale sur notre projet de recherche et quelques perspectives pour les futurs travaux.

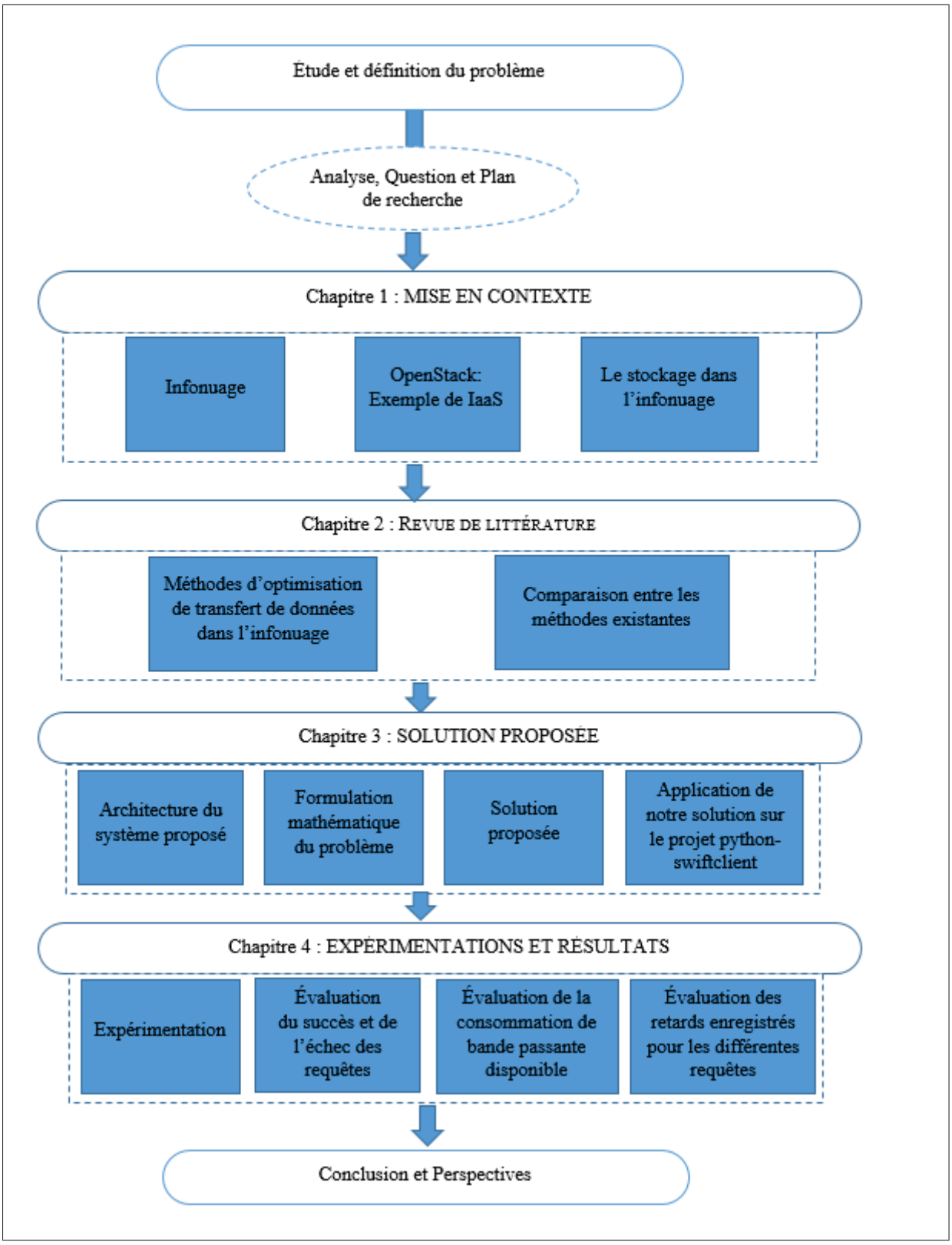

Figure 0.2 Organisation du rapport.

CHAPITRE 1

MISE EN CONTEXTE

1.1 Introduction

Ce chapitre présente les concepts de base relatifs à l'infonuage avant de mettre l'accent sur la composante stockage à laquelle nous nous intéressons dans notre mémoire. Il commence par une description générale des types de l'infonuage, suivi d'un aperçu sur le stockage dans l'infonuage. Il se concentre ensuite sur Swift de OpenStack comme un exemple de service de stockage dans l'infonuage.

1.2 L'infonuage

Dans cette section, nous allons présenter le concept de l'infonuage, ses différents caractéristiques et types.

1.2.1 Définition

L'infonuage, selon l'institut national des normes et de la technologie (NIST) (Fang Liu (2016)), est une technologie qui permet de servir un ensemble de ressources de calcul ou de stockage de serveurs informatiques distants auxquels les usagers se connectent via un réseau, généralement Internet. Ces serveurs sont loués à la demande selon l'utilisation.

Les utilisateurs de l'infonuage payent pour le service selon leurs consommations suivant le modèle 'pay as you go' d'une façon mensuelle ou annuelle. Amazon est la première entreprise qui a commencé à utiliser cette technologie dans ses centres de données en 2000 (Amazon_S3 (2013)).

Nous notons qu'un centre de données est un site physique qui regroupe les équipements constituants le système d'informations de l'entreprise à savoir les équipements réseaux, les ordina-

teurs, les serveurs et les baies de stockage. Il est interne ou externe à l'entreprise, exploité ou non avec le soutien de prestataires.

1.2.2 Caractéristiques

Les principales caractéristiques d'un infonuage sont l'accès aux services à la demande, le paiement à l'usage, l'ouverture, la mutualisation et l'élasticité (Fang Liu (2016)) :

- Accès aux services à la demande : La mise en œuvre des systèmes est totalement automatisée. En fait, l'utilisateur met en place et gère la configuration à distance au moyen d'une console de commande. Dans l'infonuage, la demande est automatique et la réponse est immédiate.
- Paiement à l'usage : Pour des raisons d'adaptation des moyens techniques, de facturation et de contrôle, la quantité de service consommée dans l'infonuage est mesurée.
- Ouverture : Les services de l'infonuage sont mis à disposition sur Internet et utilisent des techniques standardisées permettant de servir les consommateurs qui se disposent d'un ordinateur, un téléphone ou une tablette.
- Mutualisation : Les ressources hétérogènes (matériel, logiciel, trafic réseau) sont combinées afin de servir plusieurs utilisateurs auxquels les ressources sont automatiquement attribuées.
- Élasticité : Elle permet d'aligner les ressources aux variations de la demande d'une manière automatique.

1.2.3 Types de services infonuagiques

Il existent principalement trois types de services offerts pour les utilisateurs de l'infonuage, comme le montre la figure 1.1. Nous les présentons brièvement dans ce qui suit.

- Infrastructure as a Service (IaaS) : Ce modèle livre les ressources physiques ou les machines virtuelles en termes de CPU, stockage, mémoire ou système d'exploitation.

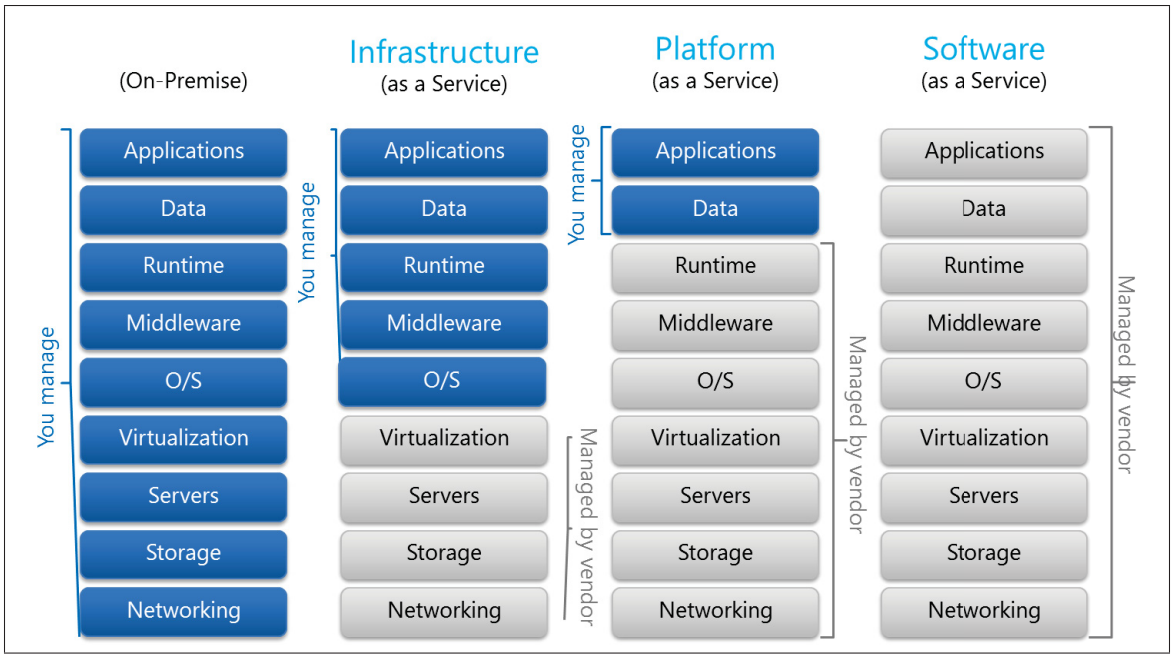

Figure 1.1 Modèle de l'infonuage. Tirée de Saurabh (2016)

- Platform as a Service (PaaS) : Dans ce type de service, les utilisateurs gèrent les services et les applications qu'ils développent. Ils ont le contrôle complet sur le déploiement de logiciels et leurs configurations. Tandis que le fournisseur maintient la plate-forme d'exécution de ces applications à savoir le matériel des serveurs, les logiciels de base et l'infrastructure qui regroupe principalement la connexion au réseau, le stockage et la sauvegarde.
- Software as a Service (SaaS) : Pour plusieurs années, les utilisateurs ont été obligés d'installer leurs propres plateformes. Cependant, aujourd'hui, grâce à l'infonuage, les logiciels d'application, à savoir, la messagerie électronique ou les jeux en ligne s'exécutant sur les infrastructures de l'infonuage, sont fournis aux utilisateurs. Ainsi, les ressources sont utilisées d'une manière efficace indépendamment des contraintes d'implantation TI. De plus, cela permet de minimiser les coûts de maintenance logicielle.

1.3 OpenStack : Exemple de IaaS

Dans cette section, nous allons présenter la plateforme OpenStack comme un exemple de IaaS en détaillant ses principaux composants.

1.3.1 Définition

OpenStack est une plateforme infonuagique qui contrôle un ensemble de ressources de calcul, de stockage et de réseau dans un centre de données à travers plusieurs services connectés (OpenStack). Toutes ces ressources sont gérées via un tableau de bord (dashboard) qui permet aux administrateurs de contrôler les ressources de l'infrastructure en permettant à leurs utilisateurs d'approvisionner des ressources à travers une interface web, une ligne de commande ou des APIs (Application Programming Interface) comme montre la figure 1.2.

Les services offerts par OpenStack (détaillés au niveau de la section suivante) sont efficacement surveillés par la composante *Monitoring* qui permet de collecter, normaliser et transformer les données produites par ces services. La solution OpenStack est un IaaS (Infrastructure as a Service) libre et ouvert (open-source). La première version livrée par la communauté fut disponible dès octobre 2010. Le projet est en développement continu depuis 2010. Chaque six mois, une nouvelle version voit le jour.

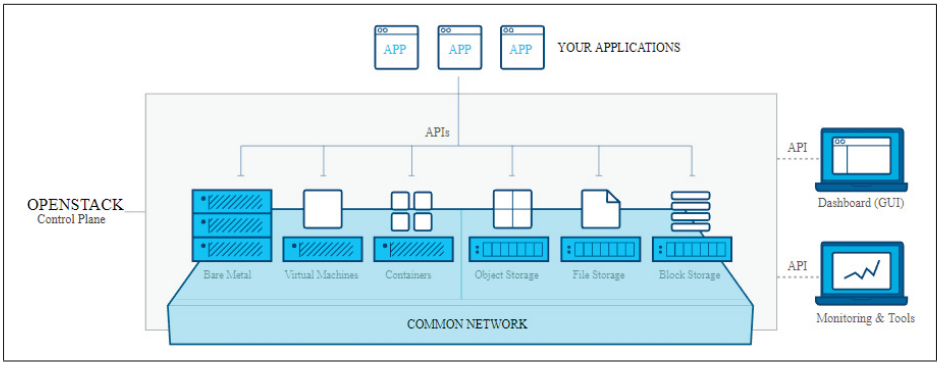

Figure 1.2 Vue globale de OpenStack. Tirée de OpenStack (2017)

1.3.2 Composants

La solution OpenStack a une architecture modulaire qui renferme plusieurs services. Dans ce qui suit, nous présentons brièvement les principaux services.

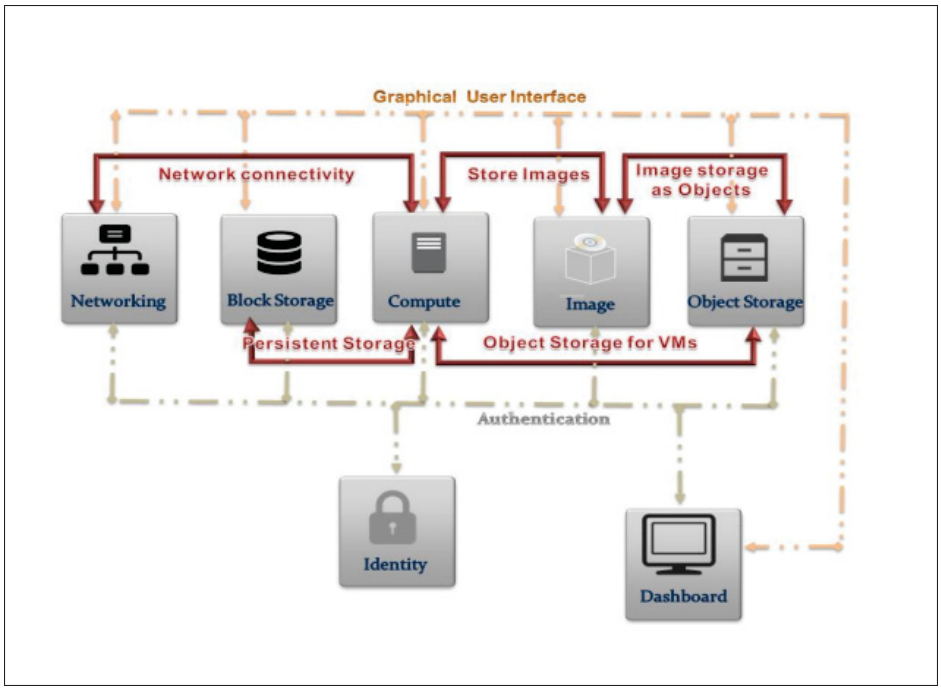

Figure 1.3 Architecture conceptuelle des services OpenStack.
Source : Architecture_OpenStack

- Calcul (Nova) : Nova est considéré comme une des briques principales d'OpenStack. Son objectif est de gérer les ressources de calcul des infrastructures. Pour ce faire, Nova fournit une API pour la manipulation des (machines virtuelles par les utilisateurs à la demande. De plus, Nova interagit fortement avec d'autres services OpenStack tels que Keystone pour l'authentification, Horizon pour son interface Web et Glance pour la fourniture des images.
- Image (Glance) : Ce composant permet de stocker les images disque et leurs méta-données.
- Identité (Keystone) : Ce composant fournit le service d'authentification et d'autorisation pour les autres services de OpenStack.
- Réseau (Neutron) : Ce composant permet de créer les réseaux, les sous réseaux, les routeurs et les ports lorsque les utilisateurs créent les machines virtuelles. Autrement, il fournit le réseau comme un service entre les équipements gérés par le service de calcul.
- Stockage : Le stockage est considéré comme l'une des fonctionnalités principales d'OpenStack. Il s'agit de traiter les demandes de gestion des espaces de stockage. OpenStack offre deux types de stockage.

- Stockage bloc (Cinder) : Il s'agit d'une exigence fondamentale pour les infrastructures virtuelles. Il est dédié pour le stockage des machines virtuelles ainsi que les données qu'elles utilisent. En fait, Cinder permet de virtualiser la gestion des périphériques de stockage et fournit aux utilisateurs finaux une API pour demander et consommer ces ressources sans avoir besoin de savoir où leur stockage est réellement déployé.
- Stockage objet (Swift) : Ce composant permet la gestion du stockage objet. Il est généralement utilisé pour le stockage des données des utilisateurs. Nous détaillerons ce composant dans la section suivante.

1.4 Le stockage dans l'infonuage

Le stockage dans l'infonuage est considéré comme l'un des services de l'infonuage les plus pertinents. Dans cette section, nous présentons plus de détails sur le stockage dans l'infonuage, ses types et quelques cas d'utilisation.

1.4.1 Définition

Le stockage dans l'infonuage est un modèle de stockage de données dans lequel les données sont stockées dans des serveurs, détenues et gérées par les propriétaires de ces derniers, appelés fournisseurs de services de stockage. Ces fournisseurs sont responsables d'assurer la disponibilité et l'accessibilité des données d'une part et le bon fonctionnement de l'environnement physique d'autre part [1].

Les organisations et les individus approvisionnent la capacité de stockage auprès des fournisseurs pour stocker leurs données.

1.4.2 Types de stockage dans l'infonuage

Nous distinguons trois types de stockage dans l'infonuage.

1. https ://aws.amazon.com/fr/what-is-cloud-storage/

- Stockage infonuagique public : L'entreprise externalise ses moyens de stockage en dédiant la gestion du stockage à une autre entreprise. Un des avantages de ce type de stockage est que l'entreprise n'a pas besoin de recruter des spécialistes dans ce domaine. Cependant, elle perd le contrôle de l'infrastructure et des données. Amazon S3, Dropbox et Google Drive sont des exemples de stockage infonuagique public.
- Stockage infonuagique privé : L'entreprise exploite l'infrastructure de stockage et ses données. Tout est déployé généralement en interne. Ainsi, l'infonuage ne sera accessible aux utilisateurs qu'à travers des réseaux sécurisés. Le stockage privé est surtout adapté aux grandes entreprises ou à celles exigeant la sécurité de leurs données. OpenStack, OpenNebula sont des exemples de stockage infonuagique privé.
- Stockage infonuagique hybride : Il regroupe les avantages du stockage public et privé. Il s'agit de mettre les données critiques dans le stockage privé et le reste dans le stockage public. Au vu de minimiser les coûts d'exploitation, les entreprises ont tendance à utiliser le stockage privé pour leurs charges usuelles et dédier le stockage public pour les surcharges ponctuelles.

1.4.3 Utilisation du stockage dans l'infonuage de l'entreprise

Au cours des dernières années, les services de stockage en nuage comme Amazon S3, Dropbox, et Google Drive, ont gagné une grande popularité. De nombreuses entreprises et de simples utilisateurs comptent sur de tels services pour héberger leurs données dans l'infonuage puisque les capacités des périphériques de stockage locaux sont encore limitées et ne sont pas suffisamment extensibles pour stocker les quantités croissantes de données. Cette croissance est dûe à la prolifération des téléphones intelligents équipés d'appareils photo haute-résolution combinée à l'amélioration des capacité des réseaux mobiles, dont les besoins en stockage et transfert sont en constante augmentation. Nous pouvons aussi citer à titre d'exemple le contenu vidéo à la demande (e.g., Netflix) dont le nombre d'abonnés a dépassé celui de la télévision câblée aux états-unis d'Amérique [2].

2. http ://money.cnn.com/2017/04/17/technology/netflix-subscribers/index.html

Le système de stockage en ligne Amazon S3 est rapidement passé de 14 milliards d'objets en janvier 2008 jusqu'à 905 milliards d'objets en mars 2012, comme le montre la figure 1.4 [3]. Cela a entraîné des prix de plus en plus bas : de l'ordre de 0.1 euro par giga-octets et par mois. Des études récentes ont estimé un taux de croissance annuel de 25,8% sur le marché mondial du stockage en nuage, qui passera de 23,76 milliards de dollars en 2016 à 74,94 milliards de dollars en 2021 (Barr (2013)).

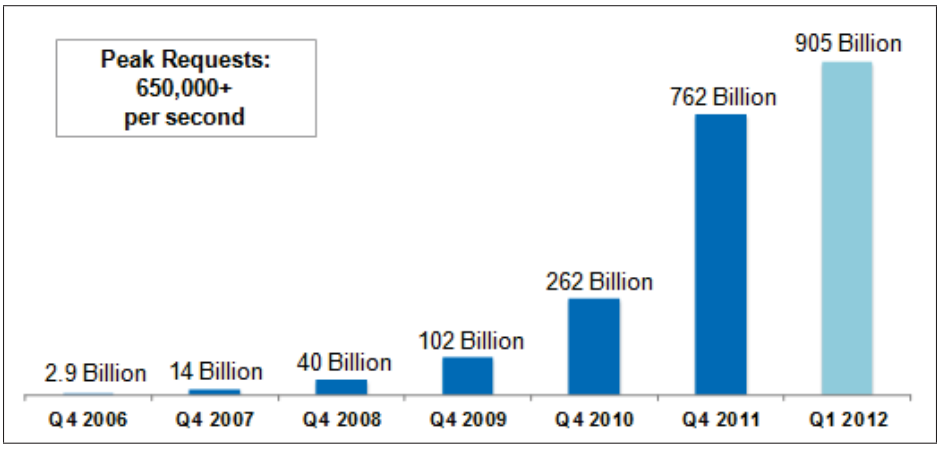

Figure 1.4 Nombre total des objets stockés dans Amazon S3.
Tirée de Barr (2013) (2016)

De plus en plus d'entreprises ont tendance à héberger leurs données dans l'infonuage notamment pour les raisons suivantes :

- Stockage à très grande échelle : Les entreprises produisent d'énormes quantités de données à savoir des e-mails, des documents, des fichiers audio et vidéo et des fichiers journaux. L'infrastructure de l'infonuage est considéré comme un moyen économique permettant de stocker ces gros volumes de données.
- Sauvegarde à distance : Au cas de la non disponibilité d'un site, il faut prévoir la distribution des données sur plusieurs sites. Pour ces fins, les services de stockage dans l'infonuage constituent une méthode simple pour sauvegarder les données hors site.

3. https ://aws.amazon.com/fr/blogs/aws/amazon-s3-more-than-449-billion-objects/

- Archivage : Les entreprises ont besoin d'archiver leurs données pour répondre aux exigences réglementaires et pour alimenter les applications d'analyse métier. Cela permet aux entreprises de conserver davantage de données de manière économique pendant plus longtemps.

1.4.4 Méthode d'accès au système de stockage

L'API de stockage est la méthode la plus utilisée pour l'accès et l'utilisation du système de stockage. La forme la plus commune des API est le service web REST (REpresentational State Transfer) [4]. Une des plus importantes propriétés de REST est son architecture sans état. Cela signifie que la requête contient toutes les informations nécessaires pour que le système de stockage renvoie la réponse adéquate. Ainsi, l'établissement d'une session entre l'utilisateur et le système de stockage n'est pas requis.

1.4.5 OpenStack Swift : Exemple de système de stockage dans l'infonuage

OpenStack Swift est parmi les systèmes open-source de stockage de l'infonuage les plus populaires (OpenStack_Swift_Community). Plusieurs entreprises comme Rackspace (Rackspace), eBay (ebay) et Instagram (Instagram) sont en train d'utiliser Swift pour le stockage, la gestion et la récupération de quantités de données massives. OpenStack a implémenté deux projets pour la gestion du stockage objet à savoir Swift du côté serveur et python-swiftclient du côté client. La figure 1.5 présente l'interaction entre ces deux projets.

- Le serveur Swift : Il est codé en Python et il est typiquement utilisé pour stocker des données non structurées (Base de données dans la figure 1.5) à savoir les machines virtuelles, les images, les fichiers audio/vidéo ainsi que les sauvegardes. Swift est capable de gérer des quantités massives de données tout en garantissant leur disponibilité et durabilité [5].

4. http ://www.restapitutorial.com/lessons/whatisrest.html
5. https ://wiki.openstack.org/

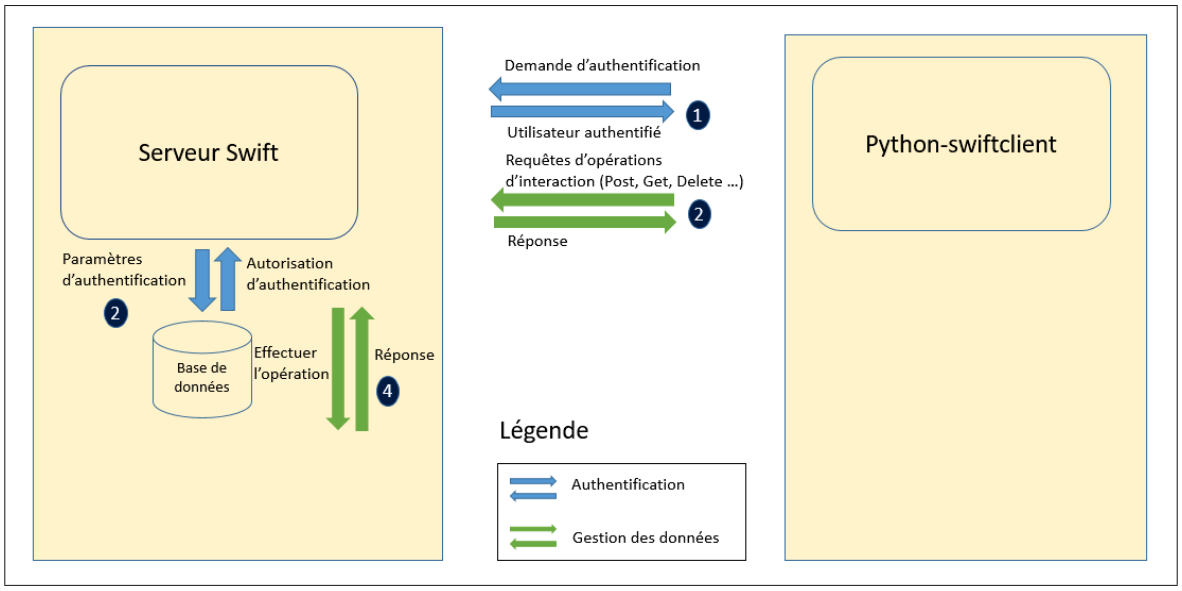

Figure 1.5 Interaction entre le serveur et le client de Swift.

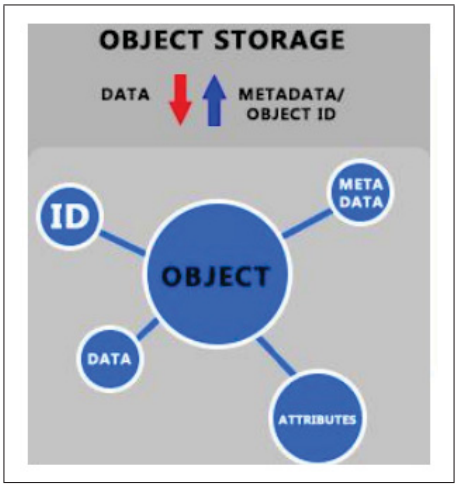

Figure 1.6 Structure de l'objet. Tirée de Sivaram (2016)

Swift gère les données en tant qu'objets contrairement aux systèmes de stockage traditionnels comme les systèmes de fichiers qui gèrent les données en utilisant une hiérarchie de fichiers. Ainsi, la donnée et ses attributs sont stockés dans le système qui permet par la suite de générer un identifiant et un ensemble de méta-données caractérisant la donnée. La figure 1.6 montre la structure d'un objet.

- Le client Swift : Les utilisateurs de Swift peuvent récupérer les données à travers une API RESTful. Une authentification de l'utilisateur est nécessaire avant toute opération d'interaction avec le serveur comme le montre la figure 1.5. Il existe une panoplie de projets non officiaux utilisés pour interagir avec le serveur Swift comme RSwift (RSwift), SwiftBox (SwiftBox) et Java OpenStack Storage (JOS). Dans ce mémoire, nous comparons nos solutions au projet python-swiftclient étant donné qu'il s'agit du projet officiel de OpenStack pour les utilisateurs de Swift [6]. Il inclut un client python pour l'API de Swift et une interface de ligne de commande qui fournissent des méthodes pour effectuer des opérations communes (par exemple téléverser, télécharger et supprimer des objets) afin d'interagir avec les serveurs de Swift en parallèle moyennant un ensemble de threads.

1.4.5.1 Les APIs RESTful dans Swift

Dans les architectures RESTful, l'entité basique de données est un objet qui peut être assimilé à un fichier dans un système de fichiers. Le protocole HTTP limite la taille de l'objet à 5GB par défaut [7]. Comme illustré dans la figure 1.7, les utilisateurs accèdent aux objets du système de stockage via une interface REpresentational State Transfer (REST) basée sur HTTP et offerte par le fournisseur de service dans le but de faciliter les interactions. Les utilisateurs de Swift peuvent gérer les données à l'aide des méthodes HTTP qui sont implémentées en tant que des web services REST. Dans ce qui suit, nous présentons les méthodes HTTP les plus utilisées :

- PUT : pour créer des objets et des conteneurs principalement.
- GET : pour récupérer des objets, lister le contenu des conteneurs et des comptes.
- DELETE : pour supprimer des objets et des conteneurs vides.
- HEAD : pour récupérer des informations sur le compte, le conteneur ou l'objet.

6. https ://docs.openstack.org/developer/python- swiftclient/introduction.html/
7. https ://aws.amazon.com/blogs/aws/amazon-s3-object- size-limit/

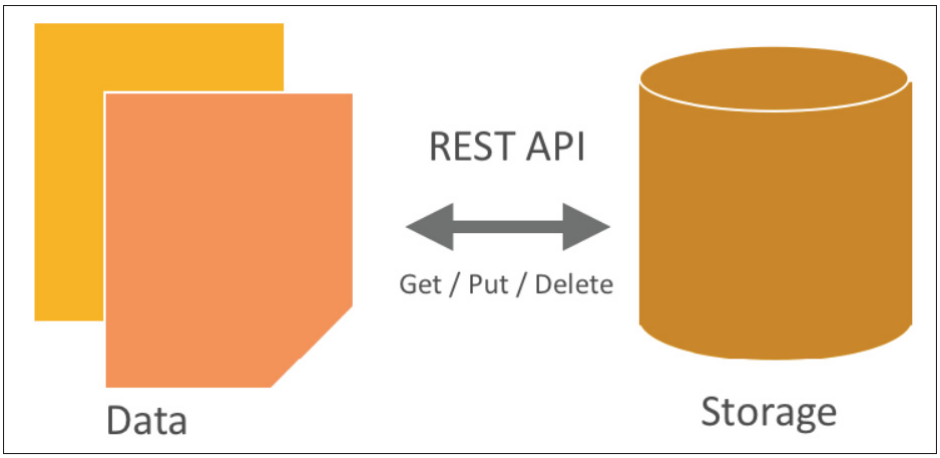

Figure 1.7 Le stockage objet dans Swift. Tirée de Architecture_OpenStack (2017)

1.4.5.2 Hiérarchie d'organisation de données dans Swift

Le système de stockage objet organise les données selon la hiérarchie suivante, comme le montre la figure 1.8 :

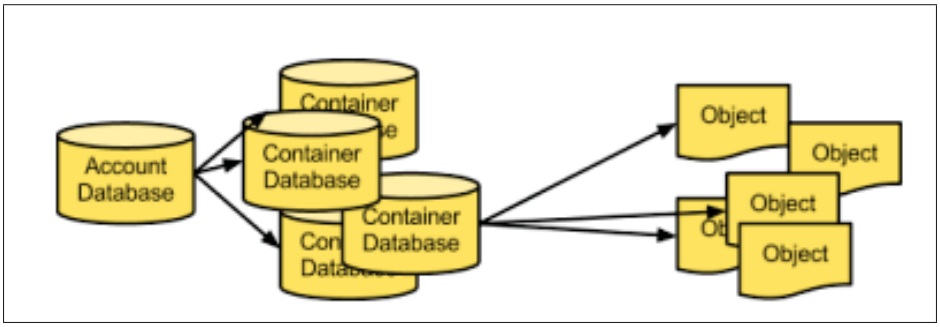

Figure 1.8 Hiérarchie des entités dans Swift. Tirée de Documentation_Swift (2017)

- Compte : Il représente le niveau le plus haut de la hiérarchie. Le fournisseur de service crée le compte et l'utilisateur est propriétaire de toutes les ressources dans ce compte. Un compte définit un espace de noms pour les conteneurs.
- Conteneur : Il définit un espace de noms pour les objets. L'administrateur du système de stockage peut créer un nombre quelconque de conteneurs dans un compte.

- Objet : Il stocke le contenu des données à savoir les documents, les images, les vidéos, etc. Chaque objet ne constitue pas seulement la donnée stockée mais aussi les méta-données et un identifiant unique. Un objet est caractérisé par un URL (Unique Ressource Locator) composé d'une adresse de service, un conteneur (groupe logique d'objets) et un nom d'objet. Spécifiquement, le chemin d'accès à l'objet reflète cette structure et possède le format suivant : /v1/Compte/Conteneur/Objet, comme le montre la figure 1.8. Nous notons que v1 se réfère à la version de Swift. Par exemple, l'objet fleurs/rose.jpg situé dans le conteneur images qui existe dans le compte 'compte1' possède le chemin suivant : /v1/compte1/images/fleurs/rose.jpg

1.4.5.3 Architecture

Swift possède une architecture client-serveur comme illustré dans la figure 1.9. Il fournit une API REST qui facilite aux utilisateurs l'interaction avec le serveur. L'architecture est composée des éléments suivants :

- Serveur proxy : Le proxy de Swift est considéré comme la porte d'entrée au système de stockage. Il reçoit et gère les requêtes et communique avec les noeuds de stockage afin d'effectuer l'action au niveau du compte, du conteneur ou de l'objet. Une fois qu'une requête arrive au serveur proxy, elle est gardée dans une file d'attente jusqu'à ce que le serveur soit disponible pour la traiter. Cette communication se fait à travers le ring que nous présentons ci-dessous. Nous notons que chaque donnée stockée possède par défaut 3 réplicas. Le noeud qui répond le premier est le responsable d'envoyer la donnée à l'utilisateur.

  Au cas où le serveur proxy reçoit simultanément un grand nombre de requêtes, certaines requêtes risquent de ne pas être traitées lorsqu'aucun serveur n'est disponible pour répondre à la requête. Par la suite, elles échouent, et par suite, l'utilisateur ne reçoit pas le fichier demandé. Pour pallier à ce problème, le serveur proxy va vérifier la possibilité d'avoir un serveur qui prend la relève en appelant le ring. Ce dernier route la requête d'une manière transparente.

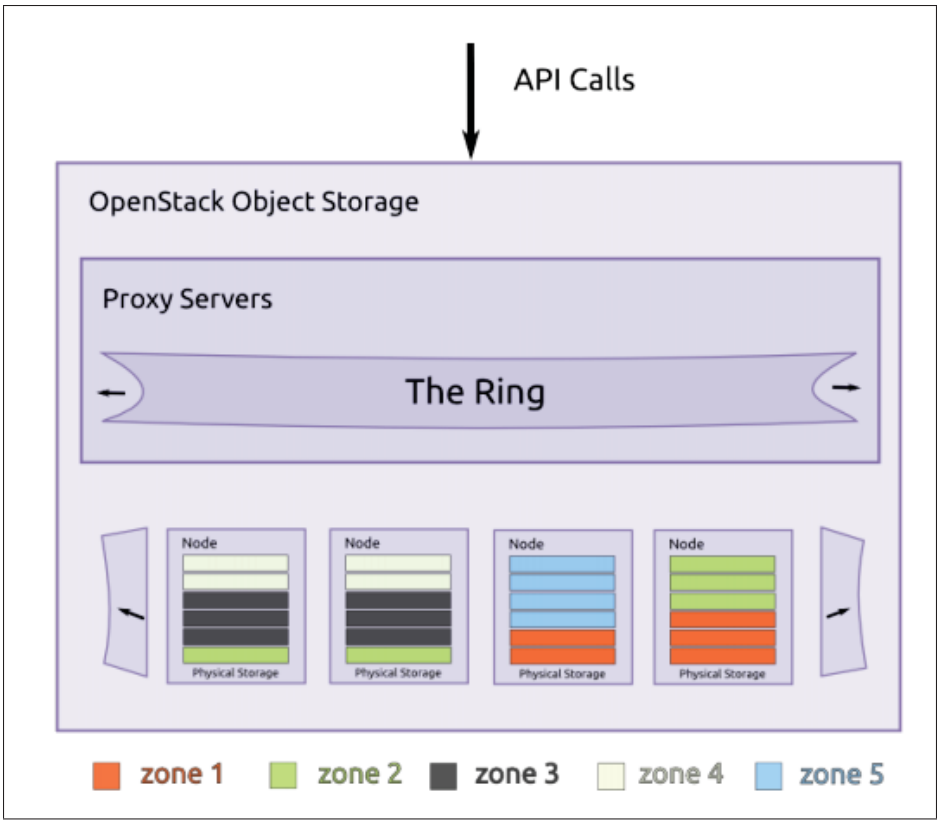

Figure 1.9 L'architecture de Swift. Tirée de Documentation_Swift (2017)

- Ring : Il s'agit d'une structure de données utilisé pour déterminer l'emplacement de toutes les entités incluant les comptes, les conteneurs et les objets, étant donné que les données sont distribuées à travers une grappe de serveurs. Les ring fonctionnent de la manière suivante : chaque ring possède une liste des disques se trouvant au niveau des noeuds de la grappe. Les partitions sont créées en utilisant une méthode de hachage et liées aux disques. Une fois une requête de stockage ou d'accès à un objet est arrivée, la valeur de hachage du chemin relatif à l'objet est calculée et utilisée comme un index pour indiquer la partition correspondante.

  Grâce au recours à la structure ring, le système de stockage peut avoir une meilleure évolutivité en gérant la capacité de stockage. Il est possible d'ajouter ou supprimer un noeud du ring sans interrompre tout le service. De plus, étant donné que la structure est totalement distribuée, cela permet d'éviter le point unique de défaillance des contrôleurs centralisés dans plusieurs systèmes de fichiers distribués.

- Noeud de stockage : Les données incluant les comptes, les conteneurs et les objets sont stockées dans les différents noeuds de stockage.
- Région/Zone : Swift permet d'assurer la disponibilité de données en isolant les partitions suivant les concepts de régions et zones. Ainsi, les données sont placées sur différents niveaux de domaines d'échec. Premièrement, les données sont réparties entre les régions, puis les zones, puis les serveurs, et enfin entre les disques. Si plusieurs régions sont déployées, le service de stockage d'objets place les données dans les régions. Dans une région, chaque réplica de données doit être stockée dans des zones uniques, si possible. S'il n'y a qu'une seule zone, les données doivent être placées sur des serveurs différents. S'il n'y a qu'un seul serveur, les données doivent être placées sur des disques différents.

1.5 Conclusion

Ce chapitre présente un aperçu de l'infonuage en général et l'outil OpenStack en particulier tout en se concentrant sur sa composante de stockage Swift. En premier lieu, il décrit les principaux types et caractéristiques de l'infonuage. Par la suite, il présente la solution IaaS OpenStack pour la gestion de ressources physiques dans l'infonuage. Ce chapitre met l'accent sur le stockage dans l'infonuage, ses types, ses cas d'utilisation et la méthode d'accès en deuxième lieu. Pour finir, ce chapitre présente Swift comme un exemple de système de stockage dans l'infonuage tout en décrivant son architecture et l'organisation des données qui y sont stockées. Le chapitre suivant présente une revue de la littérature des approches utilisées pour la gestion du transfert des données dans l'infonuage.

CHAPITRE 2

REVUE DE LITTÉRATURE SUR LA GESTION DE TRANSFERT DES DONNÉES DANS L'INFONUAGE

2.1 Introduction

À travers ce chapitre, nous présentons une étude de littérature qui se focalise sur les travaux existants traitant la gestion de transfert de données dans l'infonuage notamment la notion d'ordonnancement des requêtes de transferts de données. Par la suite, nous discutons leurs limites.

Ce chapitre commence par une brève introduction sur les méthodes d'ordonnancement. Plusieurs approches existent, nous commençons par détailler les approches visant à garantir les accords de niveau de services (SLA) ensuite celles visant une utilisation maximale de la bande passante. Par la suite, nous détaillons les travaux dont l'objectif de la maximisation à la fois du taux de requêtes satisfaites, et de l'utilisation de la bande passante. Ensuite, d'autres méthodes sont examinées. Enfin, nous comparons les différentes méthodes ainsi que leurs caractéristiques avant de conclure.

Notre solution vise à assurer le transfert de données entre les utilisateurs et les serveurs de stockage dans l'infonuage en tant que service où les utilisateurs peuvent planifier leurs requêtes à l'avance. Grâce à notre solution, les transferts de données sont désormais gérés par un composant intégré du côté client dans lequel les requêtes de transfert sont ordonnancées pour une meilleure performance en termes de consommation maximale de bande passante, de temps de transfert de données minimal et de nombre maximal de requêtes satisfaisant leurs échéances.

2.2 Méthodes d'optimisation de transfert de données dans l'infonuage

Dans cette section, nous présentons les différentes techniques et méthodes d'optimisation du temps de transfert de données dans l'infonuage. Le temps de transfert est défini comme étant

le temps entre la requête sur une donnée et le temps de la récupération de l'utilisateur de cette donnée incluant le temps d'attente et le temps de traitement.

L'idée de réduire le délai de transfert de données récupérées à partir de l'infonuage a été exploré dans des contextes variés constituant un défi pour les fournisseurs de l'infonuage. La figure 2.1 présente notre classification des travaux existants.

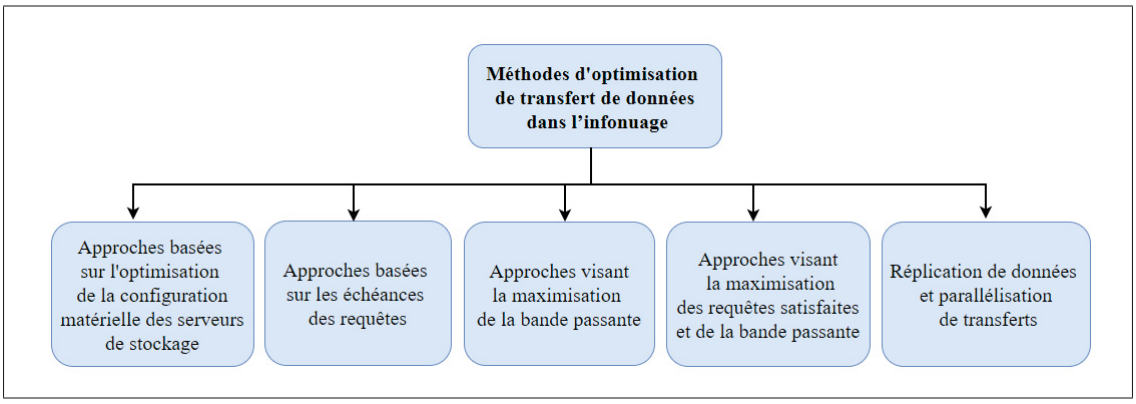

Figure 2.1 Classification des travaux existants

2.2.1 Approches basées sur la maximisation des requêtes satisfaites

Le contenu multimédia est massivement généré d'une variété d'applications et traité dans les centres de données de l'infonuage. Un principal objectif des fournisseurs de services multimédia est que leurs données puissent être récupérées par les utilisateurs le plus rapidement possible de façon à satisfaire leurs exigences spécifiées dans le contrat de services (SLA). Cela nécessite le transfert d'énormes quantités de données (flux vidéo, images, images de machines virtuelles, etc ...) à travers le réseau étendu de l'infonuage.

Afin de définir les ententes entre les fournisseurs de service de stockage infonuagique et les utilisateurs, il y a globalement recours à ce qu'on nomme des accords de niveau de service. Ces accords sont souvent définis à l'aide de plusieurs documents représentant les détails et les attentes concernant le niveau de service attendu par les tiers. Les services dits 'à-la-demande' nécessitent ces accords plus que les autres.

Avec ces ententes, les utilisateurs de l'infrastructure infonuagique peuvent composer et personnaliser leurs services et ne payer que pour ce qu'ils ont utilisé. Les accords entrent en fonction dès que le paiement est réalisé. Les fournisseurs de services sont tenus d'assurer ces ententes et ce, pour toute la durée de l'utilisation. Certains travaux sur les accords de niveau de services sont publiés et listent les métriques considérés comme les plus adéquates pour exprimer les contraintes. Dans notre contexte, nous nous intéressons aux travaux qui déterminent des SLA afin de maximiser le nombre des requêtes satisfaites à savoir ceux de Liu *et al.* (2016) et Heidari & Kanso (2016).

Dans cette partie, nous citons des exemples de travaux qui ont visé la maximisation des requêtes satisfaites soit en considérant leurs échéances ou en optimisant la configuration matérielle des serveurs de stockage. Cette optimisation est basée sur le calcul des ressources matérielles requises pour satisfaire les SLA. Le besoin d'affecter une échéance à une donnée dépend essentiellement de sa nature d'après Hong *et al.* (2013) et Chen *et al.* (2011). Nous commençons par définir les différents types de trafic inter-centres de données avant de présenter les travaux. Dans ce mémoire, nous considérons les données nécessitant la spécification d'une échéance.

Le trafic inter-centres de données peut être classifié en trois catégories en se basant sur la sensibilité au temps (Hong *et al.* (2013), Chen *et al.* (2011)) :

- Trafic interactif : C'est la catégorie la plus sensible au retardement qui requiert une réponse le plus rapidement possible. Nous citons à titre d'exemple le trafic déclenché par les services dédiés à l'utilisateur final à savoir : la recherche web, la messagerie électronique, la discussion instantanée, les jeux en ligne, etc. Une légère augmentation du temps de réponse dégrade l'expérience de l'utilisateur.
- Trafic élastique : Les transferts nécessitant un long temps d'exécution doivent souvent s'achever dans un temps spécifique Kandula *et al.* (2014). Ce genre de trafic nécessite l'achèvement de transferts de données à un temps bien spécifique qui peut être considéré comme une sorte d'échéance (quelques secondes ou minutes). Le fait de les retarder peut

directement avoir un impact sur la qualité de service, notamment réduire le temps de réponse.

Les transferts sont inutiles si les données arrivent à leurs destinations en retard. Les conséquences du retard varient selon le service. Nous citons comme exemple la réplication de la mise à jour d'une donnée dans un autre centre de données. Dans le cas d'un échec ou un retard de transfert de la mise à jour, nous risquons la perte ou l'incohérence de la donnée.

Nous citons à titre d'exemple le cas de déplacement d'un index de recherche généré dans un centre de données vers les autres centres de données où d'un jeu de données collecté dans un centre de données pour une analyse ultérieure dans un autre centre de données. En outre, retarder les transferts vers les serveurs de stockage tels que Azure ou AWS peut réduire le revenu des fournisseurs de ces services ou accroître les coûts en gaspillant d'autres ressources. Par exemple, les serveurs et les développeurs peuvent rester inactifs en attendant l'arrivée des données afin d'être en mesure de les utiliser.

- Trafic en arrière plan : Ce trafic n'est pas strict en terme de temps limite. Il est dû à des tâches internes comme l'indexation de recherche, la sauvegarde périodique de données, etc. Ce genre de trafic est gourmand en terme de bande passante. Bien que l'échéance n'est pas spécifiée, il reste désirable d'achever les transferts dès que possible.

Bien que le trafic interactif occupe la plus petite partie du volume de données échangé entre les centres de données (de 5% à 15%), de plus en plus de travaux dans la littérature visent garantir le transfert de données avant leurs échéances. Une échéance peut être suffisamment loin de sorte à permettre une flexibilité dans le processus de transfert. Ainsi, il n'est pas nécessaire que tous les transferts finissent le plus tôt possible.

Liu *et al.* (2016) se sont concentrés sur garantir la satisfaction des échéances des différents transferts de données dans la mesure où chaque utilisateur spécifie le pourcentage de requêtes qui doivent être satisfaites. Cependant, leur solution, nommée DGCLoud, s'est limitée à garantir le temps de transfert de chaque requête et n'a pas pris en compte la maximisation de l'utilisation de bande passante.

Le travail de Heidari & Kanso (2016) est une approche qui vise à calculer les ressources matérielles (Processeurs, Mémoire, etc.) des serveurs requises pour satisfaire les accords de niveau de service (SLA). Il s'agit notamment du temps de réponse et du nombre maximal de requêtes que le système peut traiter dans un laps de temps déterminé. Un accord de niveau de service peut aussi concerner les performances applicatives ainsi que les exigences de disponibilité.

Cette approche permet de mieux dimensionner l'infrastructure des systèmes d'information tout en répondant à ces accords de niveau de service. En outre, cela permet d'économiser du point de vue budget. Dans le cas IaaS, le but est de faire fonctionner toutes les machines virtuelles en respectant les exigences de SLA. Ce travail agit du côté serveur en essayant de configurer les ressources de manière appropriée. L'inconvénient de cette méthode est que la tâche de configuration doit se reproduire pour tout nouveau serveur ajouté à l'architecture, ce qui peut prendre du temps. Notre approche est différente dans la mesure où nous optimisons du côté utilisateur.

2.2.2 Approches visant la maximisation de la bande passante

Le besoin de transférer des données de tailles toujours croissantes à travers le réseau ne cesse d'augmenter. Un composant majeur nécessaire pour établir au mieux le transfert de données massives est l'infrastructure de communication qui permet de les acheminer vers leurs destinations avec succès. En particulier, une bonne gestion de bande passante disponible peut agir efficacement afin d'optimiser le temps de transfert de données ainsi que le nombre de données transférées avec succès. La bande passante peut être définie comme étant la vitesse de transfert moyenne entre les différents utilisateurs et le système de stockage infonuagique.

L'efficacité de l'ordonnancement de la bande passante dans les réseaux à hautes performances est critique pour optimiser l'utilisation des ressources du réseau et la satisfaction des requêtes des utilisateurs. Il existe deux types d'ordonnancement de bande passante, i.e., ordonnancement instantané et ordonnancement périodique.

- Ordonnancement instantané : L'ordonnancement instantané est exécuté immédiatement pour chaque requête utilisateur entrante.
- Ordonnancement périodique : L'ordonnancement périodique n'est invoqué que périodiquement pour ordonnancer les requêtes accumulées pendant une certaine période appelée période d'ordonnancement.

Récemment, l'ordonnancement périodique attire de plus en plus l'attention les chercheurs au vu de ses avantages pour l'amélioration de l'utilisation du réseau et la satisfaction les requêtes des utilisateurs.

Kosar *et al.* (2013) a élaboré une conception et un prototype d'implémentation d'un système d'ordonnancement de transfert de données réduisant le goulot d'étranglement de transfert de données. Ce système est basé sur la prédiction de la meilleur combinaison de paramètres dans le but de maximiser l'utilisation de bande passante. Ces paramètres incluent la concurrence, le parallélisme et le séquençage des requêtes. La décision de la combinaison optimale des paramètres est basée sur la taille de la donnée et la bande passante. Cependant, la garantie de la satisfaction de échéances des différentes requêtes n'a pas été prise en considération. En fait, leur algorithme commence par trier les données par ordre croissant selon leurs tailles avant de les ordonnancer. Ainsi, la taille de la donnée est considérée comme critère de priorisation. Nous considérons que cette approche n'est pas applicable pour un jeu de données où les échéances sont définies par les accords de qualité de services auxquels les utilisateurs et le fournisseur de services se sont entendus au préalable.

Laoutaris *et al.* (2011) ont élaboré la solution Netstitcher qui a pour but l'exploitation de la bande passante non utilisée sur plusieurs datacenters et réseaux et l'utiliser pour des applications non temps réel, telles que les sauvegardes, la propagation de mises à jour volumineuses et la migration de données. Cette solution vise seulement à minimiser le temps de transfert indépendamment des échéances. Netstitcher répond aux demandes suivant le mode premier arrivé premier servi, ce qui implique que les premières demandes peuvent toujours respecter leurs délais. Elle se concentre sur l'approche «stocke-et-transfère» pour ordonnancer les transferts

de données à grande échelle entre les centres de données sans tenir compte des échéances des requêtes. L'approche "stocke-et-transfère" consiste à transmettre une donnée d'une source à un noeud intermédiaire avant de la transférer au noeud de destination au cas où il n'existe pas suffisamment de bande passante.

Guo *et al.* (2016) présentent une stratégie de placement de données nommée SLA-DO dans le but d'améliorer la qualité de service et l'utilisation des ressources simultanément en garantissant la protection des données privées des utilisateurs. Afin d'évaluer les accords de niveau de service (SLA) d'un service de stockage dans l'infonuage, Guo *et al.* (2016) ont eu recours à un modèle à quatre métriques : protection des données, disponibilité, utilisation de bande passante, temps de transfert. Cependant, ces métriques sont estimées au préalable en se basant sur des formules mathématiques. Or, ces estimations peuvent s'avérer imprécises en les comparant avec les valeurs obtenues suite à la récupération des données. Dans notre travail, nous essayons d'optimiser la récupération des données de la part des utilisateurs considérant que les données sont déjà placées dans les serveurs. Ainsi, l'utilisation de bande passante et le temps de transfert sont calculées en temps réel dans la période de la récupération des données. Ceci permet à notre approche d'être plus proche des besoins réels des utilisateurs.

2.2.3 Approches visant la maximisation des requêtes satisfaites et de la bande passante

Les solutions existantes permettant de garantir une utilisation efficace de bande passante dans un centre de données (Guo *et al.* (2010), Xie *et al.* (2012), Ballani *et al.* (2011)) ne peuvent être adoptées pour résoudre notre problème. En fait, les modèles de réservation de bande passante prédéterminée peuvent garantir la satisfaction d'un nombre limité de requêtes en fournissant des garanties minimales de bande passante. Par contre, ces modèles sont moins flexibles et réactifs que ceux basés sur la réservation basée sur les échéances.

Ils fournissent des garanties sur la bande passante tandis que notre approche se concentre sur garantir le volume total de transfert avant des échéances données. Plus précisément, notre ap-

proche est plus flexible, car l'allocation de la bande passante peut changer au fil du temps tant que la requête est satisfaite avant son échéance.

Wang *et al.* (2016) ont adressé le problème d'ordonnancement périodique de bande passante appelé ordonnancement de bande passante à contrainte d'échéances multiples (i.e., M-DCBS). Leur solution, nommée MUNRRA permet de maximiser le nombre de requêtes respectant la contrainte d'échéance en considérant un chemin de réseau fixe (fixed network path). Nous notons que la priorité d'ordonnancement est accordée aux requêtes relatives aux plus petites tailles de données à transférer en triant les requêtes selon les données à transférer par ordre croissant. Cela aura pour impact de désavantager fortement les requêtes ayant pour objet des fichiers de grandes tailles, qui ne sont pas nécessairement moins urgentes que les autres. Dans notre approche, nous considérons que l'échéance est le paramètre le plus approprié pour le tri des requêtes. Ainsi, la requête la plus urgente sera en mesure d'être traitée en premier lieu.

Wang *et al.* (2016) ont considéré comme paramètres d'évaluation le temps d'exécution et le ratio de succès d'ordonnancement défini comme le nombre de requêtes satisfaites divisé par le nombre total de requêtes. Les résultats obtenus montrent un ratio de 25%, 45% et 55% pour des bandes passantes maximales de 20 Gbps, 60 Gbps et 100 Gbps respectivement. La solution que nous proposons a pour but d'augmenter davantage ce ratio.

Noormohammadpour *et al.* (2016) ont proposé RCD qui est une technique d'ordonnancement de transferts de données ayant des échéances spécifiques. Le but est de maximiser le nombre de requêtes satisfaisant leurs échéances tout en maximisant l'utilisation de bande passante.

Cette approche a principalement trois inconvénients. Premièrement, la requête est rejetée au cas où il n'existe pas de slot de temps disponible. Le principe de ré-ordonnancement que nous avons adopté dans notre solution permet d'accepter toute requête dont l'échéance n'est pas encore dépassée. Deuxièmement, RCD utilise le principe d'allouer des slots de temps le plus tard possible afin de minimiser les opérations de ré-ordonnancement des requêtes déjà traitées. Cependant, cela augmente inutilement le temps de transfert. Un autre inconvénient se résume dans la réservation de bande passante pour un trafic qui est sensé arriver en une date précise,

alors que son traitement à cette date n'est pas garanti. Par conséquent, la réservation risque d'être inutile. Ces trois limitations font que l'expérience utilisateur peut être négativement impactée sans pour autant que les accords de qualité de service ne soient totalement assurés.

Dans une étude récente, Zhang *et al.* (2017) ont proposé Amoeba, un système qui permet aux utilisateurs de l'infonuage de définir les échéances et garantir que ces derniers sont satisfaits tout en maximisant l'utilisation de bande passante. Par contre, ce système doit modifier l'ordonnancement de transferts déjà établi au cas où un transfert ne peut pas être achevé avant son échéance. Dans notre travail, nous essayons de minimiser le délai entre la décision d'ordonnancement et le transfert des données aux utilisateurs. Plus précisément, nous allouons des slots de temps à chaque transfert de telle façon à ce que si la date de transfert est arrivée, le transfert doit commencer. Par conséquent, aucun transfert ne peut être modifié que s'il est planifié prochainement.

De plus, Amoeba traite instantanément les requêtes entrantes par ordre de leur arrivée selon la stratégie Premier Entré Premier Sorti (PEPS). Ainsi, plusieurs opérations de ré-ordonnancement peuvent se produire au cas où les requêtes servies en premier ne possèdent pas des échéances proches. Dans notre approche, le traitement de requêtes se produit d'une façon périodique. Au bout de chaque période, les requêtes en attente sont rassemblées et triées par ordre d'échéance croissante. De ce fait, nous réduisons les opérations de ré-ordonnancement qui peuvent être gourmandes en terme de temps particulièrement si la quantité de requêtes est énorme. De plus, dans des cas où les ré-ordonnancements sont fréquents, les performances s'effondreraient pour les requêtes à priorité relativement basse.

Étant donné que la décision de l'ordonnancement des requêtes se fait chaque slot de temps, le choix de la durée du slot de temps est un paramètre important qui peut avoir un impact sur la performance du système d'ordonnancement. Nous fournissons plus de détails dans le chapitre suivant. Un autre inconvénient d'Amoeba se manifeste dans le choix fixe de la durée du slot de temps. Nous proposons de varier cette valeur en fonction se son impact sur le taux de requêtes satisfaites.

Yassine *et al.* (2016) ont proposé un système de bande passante à la demande nommé BoD utilisant un algorithme d'ordonnancement qui considère une variété d'échéances relatives aux requêtes de transferts de données massives. Ce système permet d'allouer la bande passante en se basant sur la priorité des transferts dans le but de maximiser le nombre de requêtes transférées avant leurs échéances.

L'inconvénient de ce système est la réservation de bande passante pour un trafic non prévu. Cette stratégie peut retarder certaines requêtes qui auraient besoin de cette quantité de bande passante réservée.

2.2.4 Réplication de données et parallélisation des transferts

Le recours à l'ordonnancement des requêtes dans l'infonuage dans le but de s'aligner avec les accords de niveau de services établis avec les utilisateurs est une approche récente qui est de plus en plus utilisée. Cependant, une partie des travaux dans la littérature a choisi d'autres approches pour ce faire, à savoir la réplication et la parallélisation des données. Nous en citons des exemples dans ce qui suit.

Joshi *et al.* (2015a); Wang *et al.* (2015) et Joshi *et al.* (2015b) ont eu recours à la même approche pour optimiser la récupération des répliquas de données. Leurs solutions consistent à l'acheminement des requêtes aux différentes sources hébergeant les répliquas de données. Une fois la donnée complètement récupérée à partir de l'une de ces sources, l'opération de transfert à partir des autres est annulée. L'inconvénient de cette approche est que la réplication des données augmente d'une manière abondante la consommation de bande passante. En outre, cela peut provoquer la congestion du réseau et par la suite accroître les temps de transfert. Toute approche peut s'avérer intéressante dans le cas où le nombre de clients est connu à l'avance.

Une autre piste à suivre pour l'optimisation est le traitement parallélisé (i.e., Multi-Threading) qui permet d'améliorer les performances lors de transferts de données dans l'infonuage. Cette optimisation est d'autant plus efficace lorsqu'il faut répondre à plusieurs requêtes à un temps donné.

Liu *et al.* (2010) utilisent une méthode qui consiste à augmenter le nombre de threads de transfert jusqu'à ce que les performances chutent. Ce travail certes, efficace à une certaine mesure, ne prend pas en compte d'autres paramètres comme la bande passante disponible. De plus, dans un cas où les fichiers sont de petites tailles, multiplier le nombre de threads peut mener à une instabilité du système.

Hou *et al.* (2017) démontrent qu'en utilisant une combinaison optimale de traitement parallèle par rapport à la taille du fichier, il est possible d'avoir de bons résultats. Par exemple, il est plus efficace (d'un point de vue utilisation de la bande passante) de transférer 16 fichiers de 256 Ko qu'un seul de 4 Mo. Une autre possibilité serait de télécharger 16 parties d'un fichier de 4Mo en parallèle. Par contre, la charge sur le serveur et le coût supplémentaire de multiplication des connections TCP peut s'avérer problématique. En effet, cela reviendrait à multiplier par un facteur donné le nombre de connexions total du serveur, sans pour autant augmenter l'utilisation de la bande passante. Dans notre cas d'usage, nous considérons que chaque fichier possède une échéance maximale avant laquelle il devrait être complètement téléchargé. Cette échéance peut être calculée où définie par l'utilisateur. L'approche visant à paralléliser tout les transferts en maximisant l'utilisation de la bande passante ne peut s'avérer efficace si nous voulons avoir un ordre d'arrivée des fichiers défini au préalable.

2.3 Comparaison entre les méthodes existantes

Cette section présente une étude comparative entre notre solution et les différents travaux déjà couverts dans la revue de la littérature. Le tableau 2.1 compare notre solution (i.e., DA et DA-Resch) à celles existantes par rapport à leurs objectifs de performances ciblés lors de la récupération des données, à savoir, une gestion de transfert de données qui prend en considération la maximisation des requêtes satisfaites en les priorisant selon leurs échéances ainsi qu'une bonne gestion de la bande passante disponible.

Aucune des solutions existantes ne peut à la fois assurer une bonne gestion de la récupération des données tout en utilisant la bande passante disponible efficacement. Dans notre travail,

nous visons à atteindre simultanément ces objectifs afin d'optimiser l'utilisation du réseau et respecter la qualité de service exigée par le client concernant la minimisation du temps de réponse et la maximisation des requêtes satisfaisant leurs échéances.

Tableau 2.1 Notre approche versus les approches existantes

Approche	**Ordonnancement des requêtes : Slot de temps dynamique**	**Gestion de transfert de données**			**Gestion de la BP disponible**		
		Minimisation du temps de transfert	Maximisation des requêtes satisfaites	Priorisation des requêtes selon leurs échéances	Monitoring de la BP en temps réel	Maximisation de l'utilisation de la BP disponible	Ordonnancement périodique de BP
Heidari & Kanso (2016)	✗	✓	✓	✗	✗	✗	✗
DGCLoud	✗	✓	✓	✓	✗	✗	✗
StorkCloud	✗	✓	✓	✗	✗	✗	✗
Netstitcher	✗	✓	✗	✗	-	✓	✓
SLA-DO	✗	✓	✗	✗	✗	✓	✓
Amoeba	✗	✓	✓	✓	-	✓	✗
RCD	✗	✓	✓	✓	-	✓	✓
MUNRRA	✗	✓	✓	✗	-	✓	✓
BoD	✗	✓	✓	✓	-	✓	✓
Joshi *et al.* (2015a) Wang *et al.* (2015) Joshi *et al.* (2015b)	✗	✓	✗	✗	✗	✗	✗
Liu *et al.* (2010) Hou *et al.* (2017)	✗	✓	✗	✗	✗	✓	✗
DA et DA-Resch	✓	✓	✓	✓	✓	✓	✓

2.4 Conclusion

Le problème d'ordonnancement des transferts de données a été introduit dans la littérature. Cependant, il reste encore un champ émergent et non parfaitement résolu. Les politiques d'ordonnancement sont généralement basées sur la minimisation du temps de transfert des données ou le nombre de transferts satisfaisant leurs échéances. Certains travaux visent également la maximisation de l'utilisation de la bande passante. Cependant, au meilleur de nos connaissances, aucun travail existant ne couvre tous ces aspects. Après avoir présenté une revue de littérature, le chapitre suivant détaille notre solution proposée qui visera à améliorer davantage la performance de transfert de données et traiter simultanément ces objectifs.

CHAPITRE 3

SOLUTION PROPOSÉE

3.1 Introduction

Dans ce chapitre, nous présentons l'architecture de la solution que nous proposons. Ainsi, nous discutons dans ce chapitre ses composantes et ses principales raisons de conception qui ont été considérées afin d'atteindre les objectifs visés comme réduire le temps de transfert des données et maximiser la consommation de la bande passante disponible afin de respecter le contrat de service avec les utilisateurs de l'infonuage.

Pour ce faire, nous avons recours d'abord à une modélisation mathématique de ce problème à l'aide d'un Programme Linéaire en Nombres Entiers (PLNE) qui permet de maximiser le nombre de requêtes satisfaites tout en assurant une consommation de bande passante optimale qui ne dépasse pas le seuil pour ne pas surcharger le réseau. Ensuite, nous avons proposé DA et DA-Resch, deux stratégies d'ordonnancement des requêtes qui prennent en considération les objectifs cités.

3.2 Architecture du système proposé

Afin de satisfaire les exigences en termes de temps de transfert de données pour un ensemble d'utilisateurs au sein de la même entreprise, nous proposons d'ajouter une couche intermédiaire pour la gestion des requêtes du côté de l'entreprise (côté client). Comme le montre la figure 3.1, le système proposé est composé des modules suivants :

3.2.1 Module de surveillance de la bande passante

Ce module est responsable d'estimer la bande passante disponible entre l'entreprise et le système de stockage infonuagique en temps réel.

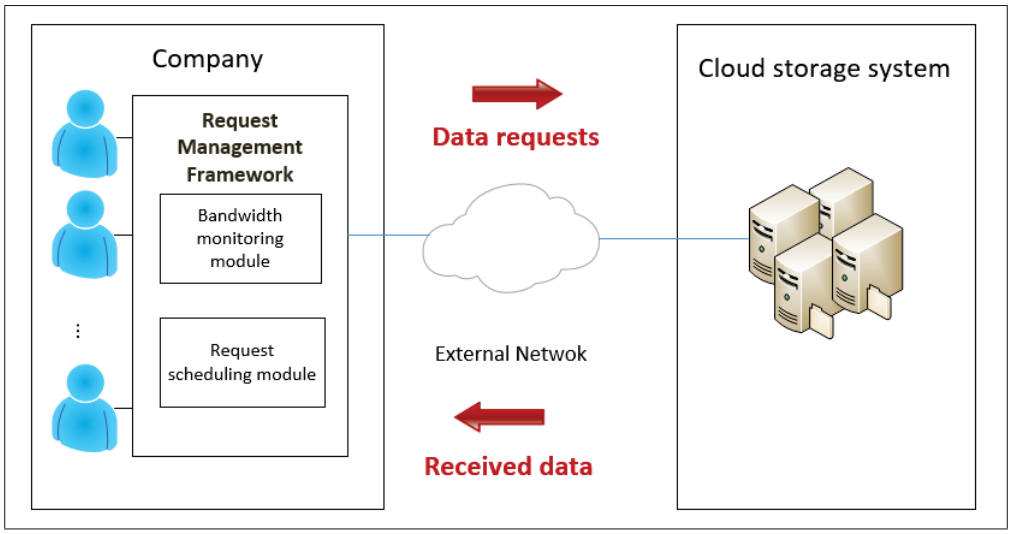

Figure 3.1 Mécanisme proposé de récupération des données.

Afin d'ordonnancer les requêtes arrivées, nous avons élaboré un module qui fournit des mesures de la bande passante disponible en temps réel sur les données actuelles récupérées par l'entreprise.

L'objectif de ce module est de prédire la bande passante disponible durant les transferts notée B_t entre le système de stockage en nuage et les utilisateurs à chaque créneau temporel t. La bande passante est utilisée pour estimer le temps de transfert de chaque fichier demandé pour le prochain créneau temporel dans le mécanisme d'ordonnancement. En effet, il est nécessaire d'avoir une estimation actualisée de la bande passante utilisée en raison de la variation du trafic dans le temps. Ainsi, nous utilisons un modèle régressif pour mesurer la bande passante. À $t = 1$, nous assignons une valeur par défaut équivalente à la capacité du lien à la bande passante. Pour $t > 1$, nous considérons que la bande passante qui va être consommée à chaque créneau temporel est égal à la bande passante mesurée pendant le créneau précédent, comme présenté par la formule (3.1).

$$B_t = B_{t-1} \quad \forall 2 \leq t \leq |T| \tag{3.1}$$

3.2.2 Module d'ordonnancement de requêtes

Dans les systèmes actuels, une requête de transfert de données est gérée par l'ordonnanceur sans considérer les contraintes de l'utilisateur. La requête est placée dans une file d'attente jusqu'à l'achèvement des opérations en cours. Ainsi, une requête peut être retardée en raison de transferts antérieurs de longue durée, ou elle peut être reportée pour opérer d'autres petits transferts selon la politique de l'ordonnanceur. Dans tels cas, le nombre de transferts achevés augmentera, mais le temps de transfert de chaque donnée va également augmenter. Retarder l'opération de transfert de données pour l'achever après le temps de fin attendu peut créer plusieurs problèmes.

Pour pallier à ce problème, nous avons implémenté un module d'ordonnancement de requêtes. Ce dernier rassemble les requêtes de transfert de données des utilisateurs de l'entreprise et les planifie en tenant compte des échéances requises par les utilisateurs et en maximisant l'utilisation de la bande passante disponible. Pour ce faire, le module prend en compte la bande passante estimée fournie par *le module de surveillance de la bande passante*. Nous détaillerons les fonctionnalités de ce module dans la section 3.4.

3.3 Formulation mathématique du problème

Dans cette section, nous proposons une modélisation mathématique du problème d'ordonnancement de requêtes en tant qu'un programme linéaire en nombres entiers (Integer Linear Program – ILP). Nous commençons par définir les différentes variables du problème avant de décrire ses contraintes et sa fonction objectif.

3.3.1 Variables

Étant donné un ensemble d'utilisateurs envoyant N requêtes simultanément afin de récupérer des données à partir d'un système de stockage infonuagique, nous définissons S comme un ensemble contenant les tailles des données à récupérer, où $S = (s_i)_{i\in[1,N]}$. Soit R désignant un ensemble de temps de transfert estimé de la donnée à récupérer, où $R = (r_i)_{i\in[1,N]}$. Chaque don-

née doit être transférée avant une échéance spécifique. Soit D désignant l'ensemble d'échéances correspondant aux requêtes, où $D = (d_i)_{i\in[1,N]}$. En fait, les utilisateurs peuvent ajouter une contrainte de temps spécifiant la date limite souhaitée pour terminer l'opération de transfert.

Considérons un timeline T divisé en slots de temps. La décision d'ordonnancement des requêtes en attente est prise à chaque créneau temporel $t \in [t_0, t_0 + |T|]$. Soit Δt la durée du créneau temporel. Nous admettons que les serveurs et les utilisateurs du système de stockage infonuagique sont connectés à travers un réseau backbone.

La bande passante mesurée entre le système de stockage infonuagique et les utilisateurs pendant le créneau temporel t est dénotée par B_t. Étant donné que les requêtes de transfert de données peuvent être traitées simultanément, nous définissons π_t^{max} comme le nombre maximal de requêtes simultanées qui sont servies pendant un créneau temporel t. Cela correspond au nombre maximal de transferts simultanés.

Nous définissons x_{it} comme la variable de décision représentant la séquence d'ordonnancement dans le temps. En d'autres termes, nous avons :

$$x_{it} = \begin{cases} 1, & \text{si la requête } i \text{ est en cours de traitement au créneau temporel } t. \\ 0, & \text{sinon.} \end{cases}$$

De plus, nous définissons z_i afin d'identifier les requêtes qui respectent leurs échéances, tel que :

$$z_i = \begin{cases} 1, & \text{si la requête } i \text{ respecte son échéance.} \\ 0, & \text{sinon.} \end{cases}$$

Notre objectif est de trouver le schéma d'ordonnancement de transferts optimal en décidant de quelle donnée doit être transférée à l'utilisateur au créneau temporel t tout en respectant les échéances spécifiées et maximisant l'utilisation de la bande passante disponible.

3.3.2 Fonction objectif

Le problème d'optimisation proposé se concentre sur les deux objectifs suivants. Premièrement, il vise à maximiser le nombre de transferts de données qui respectent leurs échéances (c'est-à-dire, le premier terme de l'équation Eq.(3.2)). Deuxièmement, il vise à maximiser le nombre de requêtes planifiées dans chaque créneau temporel afin de maximiser l'utilisation de la bande passante disponible (c'est-à-dire, le second terme dans Eq.(3.2)). La fonction objectif peut être écrite comme suit :

$$\text{maximize} \sum_{i=1}^{N} z_i + \sum_{t=1}^{|T|} \sum_{i=1}^{N} x_{it} \tag{3.2}$$

3.3.3 Contraintes

Nous présentons dans ce qui suit un ensemble de contraintes qui devraient être satisfaites tout en réalisant la fonction objectif. Dans l'équation Eq. (3.3), nous supposons que le nombre de transferts planifiés à chaque créneau temporel t ne doit pas dépasser le nombre maximum de transferts concurrents.

$$\sum_{i=1}^{N} x_{it} \leq \pi_t^{max} \quad \forall 1 \leq t \leq |T| \tag{3.3}$$

Nous calculons le temps de transmission de la requête i comme suit :

$$r_i = \sum_{t=1}^{|T|} x_{it} \frac{s_i}{B_t . \pi_t^{max} . \Delta t} \quad \forall 1 \leq i \leq N \tag{3.4}$$

Étant donné que le fichier de données relatif à une requête i sera transféré pendant un ou plusieurs créneaux temporels, le temps de transmission r_i est exprimé comme une somme des temps de transmission estimés pour chaque créneau temporel. En utilisant Eq. (3.5) et Eq. (3.6), nous nous assurons que la valeur de x_{it} est maintenue à 1 pendant la période de transfert de

données estimée r_i. En d'autres termes, si x_{it} est égal à 1, alors $x_{i(t+j)}$ doit être égal à 1, $\forall j \in [1, r_i]$. Si x_{it}=1, alors $x_{i(t+j)}$ est égal à 1. Si x_{it} est égal à 0, cela signifie que la requête i n'est pas transmise et donc $x_{i(t+j)}$ peut être égal à 0 ou 1. Pour capturer cette contrainte, nous utilisons les équations suivantes :

$$M.x_{i(t+j)} \geq x_{it} \quad \forall 1 \leq t \leq |T|, 1 \leq i \leq N \tag{3.5}$$

$$x_{i(t+j)} \geq x_{it} \quad \forall 1 \leq t \leq |T|, 1 \leq i \leq N, 1 \leq j \leq r_i \tag{3.6}$$

Eq. (3.7) assure que le nombre de slots de temps assignés à une requête est égal au temps de transmission estimé de la requête i.

$$\sum_{t=1}^{|T|} x_{it} = r_i \quad \forall 1 \leq i \leq N \tag{3.7}$$

Dans Eq. (3.8) et Eq. (3.10), nous désignons par y_i la différence entre l'échéance d_i et le temps de transmission $(t + r_i)$, qui est le temps estimé de la fin de transmission. Notre objectif est d'identifier les requêtes respectant leurs échéances grâce à la variable de décision z_i. Par conséquent, lorsque la fin de la transmission de la donnée relative à la requête i est prévue après son échéance (c'est-à-dire que y_i est strictement négatif), nous affectons la valeur 0 à z_i. Sinon, si les données demandées sont transférées avant leurs échéances (c'est-à-dire y_i est positif), z_i est égal à 1.

$$y_i > -M.(1 - z_i) \quad \forall 1 \leq i \leq N \tag{3.8}$$

$$y_i = d_i - (t + r_i) \quad \forall 1 \leq t \leq |T| \tag{3.9}$$

$$y_i < M.z_i \quad \forall 1 \leq i \leq N \tag{3.10}$$

où M est une constante ayant une grande valeur.

Ce problème est NP-difficile étant donné qu'il se rapporte au problème générique d'ordonnancement de ressources (Garey & Johnson (2002)) qui est NP-difficile. Par conséquent, il ne peut pas être résolu dans un temps polynomial pour un grand nombre de requêtes. Nous proposons donc dans la section suivante des algorithmes gloutons pour traiter le problème à une large échelle.

3.4 Solution proposée

Nous proposons un schéma d'ordonnancement de requêtes qui permet à l'utilisateur de choisir entre deux algorithmes : Deadline-Aware (DA) et Deadline-Aware avec Rescheduling (DA-Resch) en fonction de ses préférences. Dans ce qui suit, nous détaillons ces algorithmes et présentons un exemple illustrant notre mécanisme d'ordonnancement de requêtes (Fig. 3.2).

3.4.1 Ordonnancement de requêtes

Dans notre solution, les demandes sont stockées dans une file d'attente dès leur arrivée. L'algorithme d'ordonnancement est exécuté avant le début de chaque créneau temporel. Les requêtes sont triées en fonction de leurs échéances. Dans le cas où deux requêtes ont la même échéance, nous conservons l'ordre de leur arrivée.

A chaque créneau temporel, les requêtes sont affectées aux threads disponibles un par un. La requête est rejetée au cas où son échéance a été atteinte. L'utilisateur est, ainsi, demandé à définir une autre échéance. Nous notons que la bande passante disponible à chaque créneau temporel est répartie équitablement entre tous les threads. L'ensemble des requêtes placées en file d'attente présentées dans la figure 3.2 est déjà triée par échéance.

A chaque fois que nous avons une nouvelle requête, nous calculons le temps de transmission estimé r_i en termes de créneaux temporels en fonction de la taille de la donnée et de la

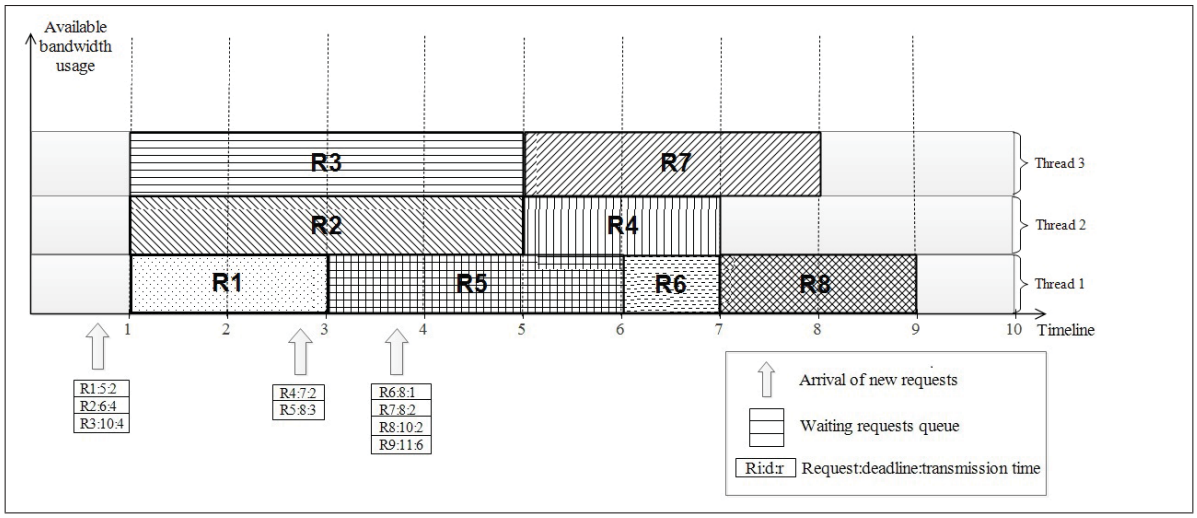

Figure 3.2 Exemple d'ordonnancement de requêtes.

bande passante disponible fournie par le module de surveillance. Nous assignons ensuite les demandes aux threads disponibles tout au long du scénario. Par exemple, dans la figure 3.2, le créneau temporel t_1 est assigné aux requêtes $R1$, $R2$ et $R3$, chacune traitée par un thread. Les requêtes sont ensuite envoyées au serveur qui commencera à transférer les données relatives à ces requêtes.

Nous remarquons que la requête $R5$ est planifiée avant $R4$ même s'il n'est pas en haut de la file d'attente afin de maximiser l'utilisation des créneaux temporels ce qui conduit à maximiser la bande passante disponible. La requête $R6$ est retardée afin de programmer la requête $R7$ avant son échéance. Enfin, la requête $R9$ est rejetée car son échéance n'a pas pu être respectée.

L'algorithme Schedule requests (Figure 3.3) permet de choisir entre deux options. La première option (*DA-Resch* = False) implique que seulement l'algorithme Deadline-Aware (DA) est exécuté. Sinon (*DA-Resch* = True), l'algorithme Deadline-Aware avec rescheduling (DA-Resch) est exécuté après l'algorithme DA. Le recours à DA-Resch est avantageux au cas où le nombre de requêtes non planifiées est important. Nous notons que les deux algorithmes présentés sont exécutés au début du créneau temporel courant c_t.

Algorithm 1: Schedule requests

Input : $DA\text{-}Resch$ (= False by default), current time slot c_t, monitored bandwidth B_t, list of waiting requests I_t, timeline T, time slot duration Δt, maximum number of threads at time t π_t^{max}

Output: Vector H = $(H_t)_{t \in T}$ where H_t is the set of requests to be sent at time slot t

$I_t \leftarrow$ List of waiting requests sorted by closest deadline.
foreach i *in* I_t **do**
    $r_i = \frac{s_i}{B_t . \pi_t^{max} . \Delta t}$
    $H \leftarrow$ DA(c_t, i, r_i, π_t^{max}, T)
end
if $DA\text{-}Resch$ = *True* **then**
    $H \leftarrow$ DA-Resch(c_t, I_t, π_t^{max}, T, H)
end

Figure 3.3 Algorithme d'ordonnancement de requêtes.

3.4.2 Algorithme Deadline-Aware (DA)

Cet algorithme consiste à trier les requêtes placées en file d'attente selon l'échéance la plus proche. Pour chaque requête, il calcule le temps de transfert estimé r_i en fonction de la bande passante surveillée B_t à l'instant t. La fonction DA (Figure 3.4) est appelée afin d'assigner le nombre requis de créneaux temporels pour chaque requête i.

L'idée est d'assigner les premiers créneaux temporels disponibles nécessaires au transfert des données demandées en tenant compte de l'échéance et du nombre maximal de threads. Comme nous considérons que les transferts de données sont non préemptifs, les créneaux temporels disponibles devraient être consécutifs. La fonction renvoie une liste de requêtes planifiées pour chaque créneau temporel. Seules les requêtes planifiées pour le créneau temporel courant sont retirées de la liste des requêtes en attente I_t et envoyées au serveur pour être traitées selon

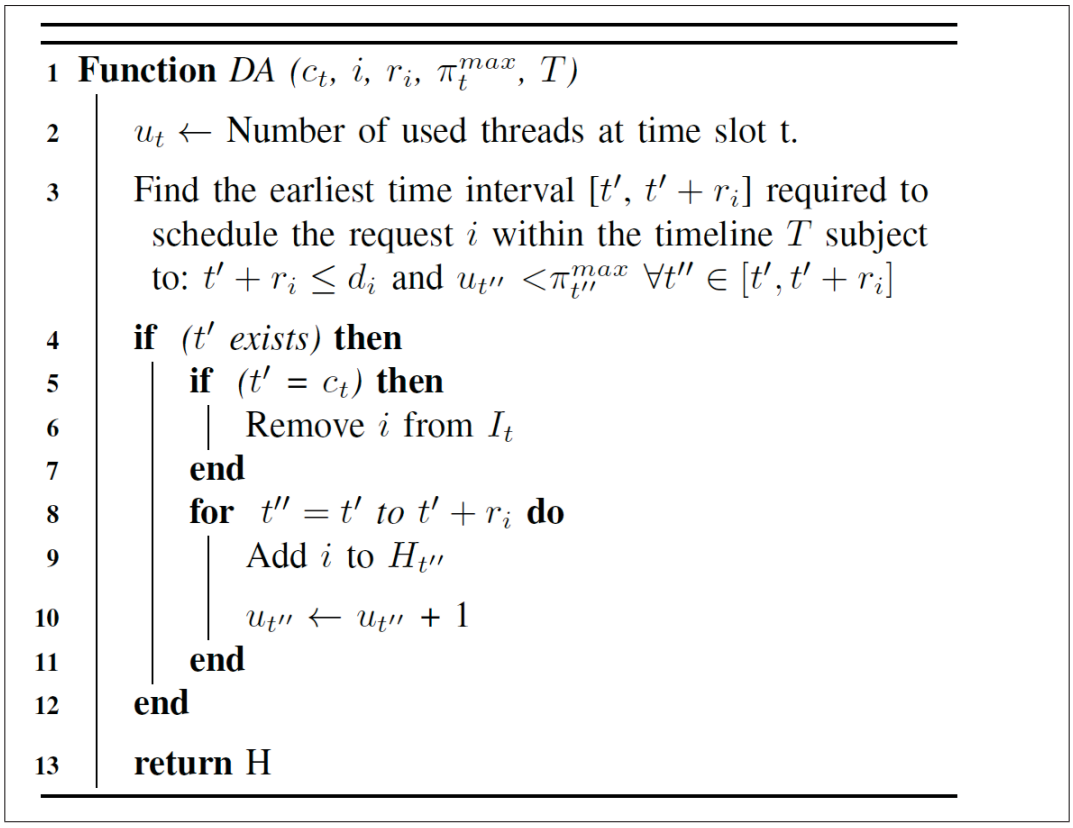

Figure 3.4 Fonction d'allocation de créneaux temporels aux requêtes.

le même ordre. Les autres requêtes seront rajoutées à la liste des requêtes en attente pour le prochain créneau temporel.

3.4.3 Algorithme Deadline-Aware avec Rescheduling (DA-Resch)

L'algorithme DA fournit un ordonnancement provisoire pour l'ensemble des requêtes I_t triées selon l'échéance la plus proche. Cependant, certaines requêtes de I_t ne peuvent pas être planifiées en raison du manque de threads et de créneaux temporels libres. À travers des expériences, nous avons remarqué que le nombre de requêtes non planifiées peut être important dans certains cas. Afin de surmonter cette limitation, nous proposons d'optimiser davantage l'ordonnancement obtenu avec l'algorithme DA en utilisant l'algorithme DA-Resch (Figure 3.5).

```
1  Function DA-Resch (c_t, I_t, π_t^max, T, H)
2      foreach i in I_t do
3          if (i not scheduled) then
4              l_i ← d_i - r_i ; t ← c_t ; t' ← c_t ;
                FreeSlots ← False ;
5              while (t <|T| and FreeSlots = False) do
6                  j ← 0
7                  while (j <length(H_t) and FreeSlots =
                    False) do
8                      resch ← H_t(j)
9                      Find the earliest time interval
                        [t', t' + r_resch] required to
                        reschedule the request resch within
                        the timeline T in a way to minimize
                        (r_resch − r_i) subject to: t' ≤ l_i,
                        r_resch ≥ r_i, d_resch ≥ t' + r_resch and
                        u_t'' <π_t''^max ∀t'' ∈ [t', t' + r_i]
10                     if (t' exists) then
11                         Remove resch from H and put i
12                         for t'' = t' to t' + r_resch do
13                             Add resch to H_t''
14                             u_t'' ← u_t''+ 1
15                         end
16                         FreeSlots ←True
17                     else
18                         j ← j + 1
19                     end
20                     if (j = length(H_t)) then
21                         t ← c_t + 1
22                     end
23                 end
24             end
25         end
26     end
27     return H
```

Figure 3.5 Fonction de ré-ordonnancement de requêtes.

L'idée est d'essayer d'ordonnancer les requêtes non planifiées en retardant les requêtes planifiées par l'algorithme DA, si possible. Par exemple, si une requête i n'est pas planifiée par DA, la fonction DA-Resch tente de retarder l'une des requêtes planifiées (notée *resch* dans la fonction Rescheduling) afin de faire de la place à la requête i. En d'autres termes, si la requête *resch* peut être replanifiée en respectant son échéance (i.e., $d_{resch} \geq t' + r_{resch}$), elle sera retirée de H_t qui est l'ensemble de requêtes à planifier à l'instant t et déplacée à $H_{t'}$ (à planifier à l'intervalle $[t', t' + r_{resch}]$). La requête i est alors planifiée à sa place et ajoutée à H_t. Ainsi, la réservation de ressources peut être annulée dans le but d'affecter les ressources à une requête plus prioritaire au cas où nous pouvons retarder la requête préalablement planifiée. Par conséquent, DA-Resch permet de maximiser le nombre de requêtes planifiées au créneau temporel t en optimisant l'ordonnancement établi par l'algorithme DA.

3.5 Application de notre solution sur le projet Python-swiftclient

L'ordonnancement d'objets suivant la méthode Premier Entré Premier Servi (PEPS), sans prendre en compte les échéances des requêtes, est actuellement utilisé pour envoyer les données aux utilisateurs dans le projet Python-swiftclient que nous avons présenté dans le chapitre 1. Comme le serveur Swift envoie les données aux utilisateurs dès que les requêtes sont traitées, certaines requêtes peuvent être pénalisées. Dans ce cas, les besoins des utilisateurs en termes de temps de transfert ne peuvent pas être satisfaits.

Pour résoudre ce problème, nous avons implémenté notre solution, incluant le module de surveillance en temps réel de bande passante et celui d'ordonnancement de requêtes, en langage Python et intégré dans la solution Python-swiftclient de OpenStack. Notre solution peut également être implémentée dans d'autres systèmes de stockage infonuagique étant indépendante de la plateforme de stockage.

3.6 Conclusion

Dans ce chapitre, nous avons présenté la solution proposée pour optimiser la gestion des transferts de données entre le système de stockage de l'infonuage et ses utilisateurs. Nous avons considéré comme paramètres d'optimisation la satisfaction des utilisateurs dans la mesure de maximiser le nombre de requêtes de transferts de données respectant leurs échéances tout en assurant une consommation maximale de la bande passante disponible. Pour ce faire, nous avons eu recours à la programmation linéaire pour modéliser notre problème d'ordonnancement de requêtes. Ce problème, étant NP-difficile, ne peut pas être résolu dans un temps polynomial pour un grand nombre de requêtes. C'est la raison pour laquelle nous avons proposé des algorithmes gloutons, baptisés DA et DA-Resch pour traiter le problème à une large échelle. Par la suite, nous avons implémenté ces algorithmes en Python et intégré un module au projet Python-swiftclient de OpenStack. Dans le chapitre suivant, nous allons décrire notre environnement d'expérimentation, discuter du méchanisme de fonctionnement de Swift en détail avant de présenter l'ensemble des résultats obtenus.

CHAPITRE 4

EXPÉRIMENTATIONS ET RÉSULTATS

4.1 Introduction

Dans ce chapitre, nous commençons par décrire l'environnement expérimental utilisé pour mettre en place le transfert de données entre le serveur Swift et les utilisateurs ainsi que le scénario expérimental utilisé pour comparer les algorithmes DA et DA-Resch avec la solution python-swiftclient de OpenStack. Enfin, nous comparons les performances de Swift en utilisant nos algorithmes proposés DA et DA-Resch pour l'ordonnancement des requêtes avec le Swift traditionnel, en termes de la proportion de requêtes respectant leurs échéances, la consommation de bande passante et les retards enregistrés.

4.2 Expérimentation

Dans cette section, nous présentons l'environnement expérimental qui nous a permis de tester les performances des différentes solutions étudiées. Nous décrivons, par la suite, la démarche suivie pour mener à bout nos tests incluant la méthode de génération des requêtes, les opérations principales d'interaction avec Swift utilisées dans l'expérimentation ainsi que la méthode de spécification des échéances des requêtes.

4.2.1 Environnement expérimental

Les expériences ont été conduites sur un cluster Linux à deux nœuds, comme illustré dans la figure 4.1. Chaque nœud est équipé de deux processeurs Intel Xeon 10 cœurs (E5-2650), à une fréquence de 2,3 GHz (40 threads) et 128 Go de mémoire. Le premier nœud est utilisé en tant que client pour émuler les requêtes simultanées des utilisateurs. Le deuxième nœud est utilisé comme le nœud de stockage en nuage exécutant le serveur de stockage et le serveur proxy qui accepte les requêtes entrantes provenant du nœud client. Chaque nœud est équipé de deux disques durs Seagate Constellation ST3000NM0033-9ZM178 (SN04) de 3 To, dont l'un

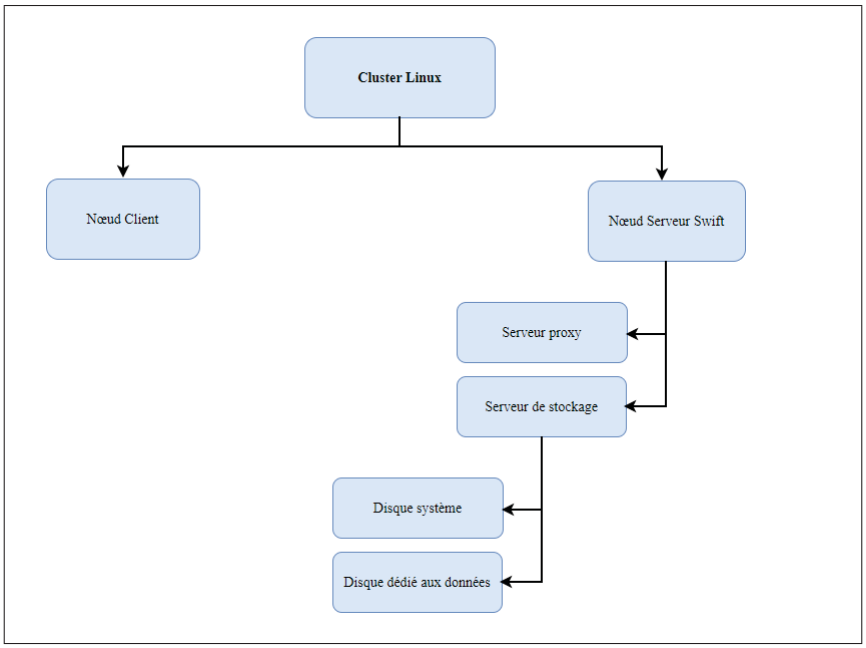

Figure 4.1 Environnement expérimental.

est utilisé pour les expériences et l'autre est utilisé comme un disque système. Le noeud client est connecté au noeud de stockage à l'aide de liens 10GbE.

4.2.2 Scénario expérimental

Nous rappelons que notre but est d'ajouter un module dans le projet python-swiftclient qui implémente nos algorithmes d'ordonnancement. La solution proposée vise à intégrer un mécanisme de planification de requêtes dans le projet python-swiftclient, en tenant compte des contraintes des utilisateurs dans le but de maximiser l'utilisation de la bande passante. La planification permet d'envoyer l'ensemble des requêtes GET à plusieurs créneaux temporels au serveur Swift. Ensuite, Swift les traite dans le même ordre d'arrivée. Dans une certaine mesure, notre méthode essaie de contrôler le transfert de données du serveur Swift en agissant du côté client.

- **Génération des fichiers de test et des requêtes des clients :**

  Dans ce travail, nous considérons le système de stockage en nuage Swift comme une boite noire et le projet Python-swiftclient en tant qu'un client de Swift. Nous avons développé un script shell permettant de générer des requêtes GET des utilisateurs vers Swift.

Tableau 4.1 Modèle d'accès au fichier. Source : Yahoo

Catégorie	Statistiques de Yahoo Research		Fichiers générés	
	Catégorie de la taille du fichier	Proportion	Taille moyenne	Nombre
1	<4Ko	21.71%	678o	217
2	<100Ko	18.78%	28Ko	187
3	<1Mo	12.28%	403Ko	123
4	<64Mo	31.28%	15Mo	313
5	<1Go	14.97%	203Mo	150
6	>1Go	0.96%	2.5Go	10

Les fichiers demandés par le client font un total de 1000 fichiers de tailles et de nombres variés comme le montre le tableau 4.1. Pour imiter un scénario réaliste, le choix des tailles et des nombres des fichiers était basé sur un ensemble de traces d'accès aux fichiers fournies par la communauté de recherche de Yahoo (Yahoo) récapitulé dans le tableau 4.1. Pour chaque catégorie de taille de fichier, nous avons choisi la taille moyenne comme une taille représentative. Par exemple, si nous considérons la première catégorie, 21,71% des fichiers accédés par les différents utilisateurs ont une taille moyenne de 678 octets avec un total de 217 pour la durée de l'expérience. Nous notons que les expériences sont effectuées en considérant trois scénarios différents. Ces scénarios se distinguent par des taux d'arrivée de requêtes différents extraits des données fournies par Yahoo afin de simuler une architecture d'infonuage réelle. Comme le montre la figure 4.2, nous avons choisi trois taux d'arrivée de requêtes variable d'un créneau temporel à l'autre avec un total de 1000 requêtes pour chaque scénario.

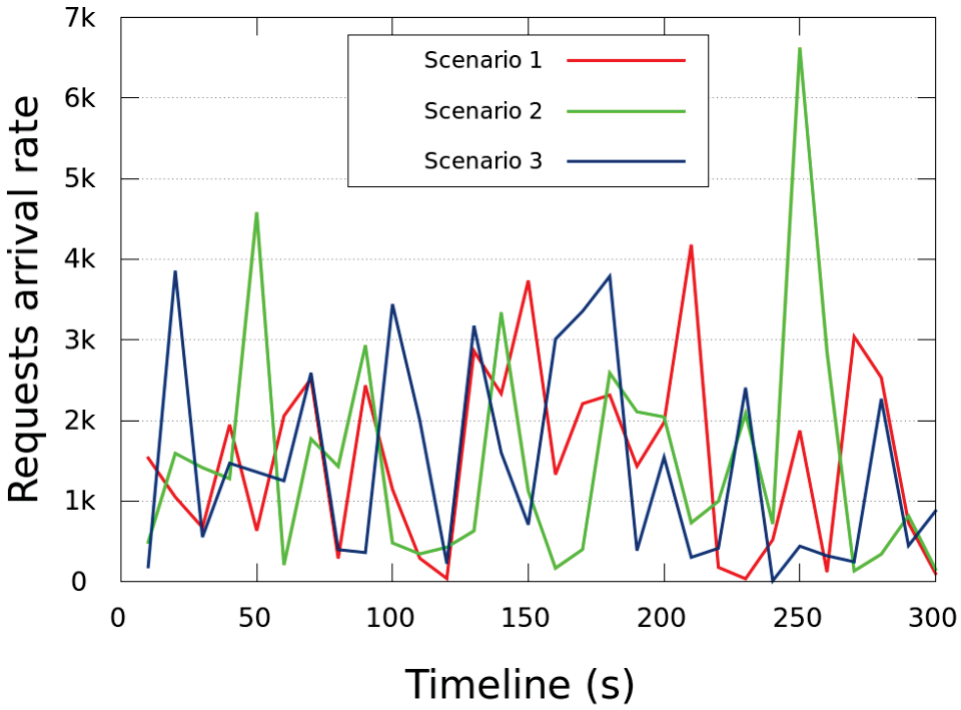

Figure 4.2 Taux d'arrivée des requêtes.

- **Opérations d'interactions avec Swift :**

  Une fois les fichiers de test ont été générés, nous nous sommes servis d'un ensemble de commandes qui facilitent l'interaction avec Swift afin de créer des conteneurs de données, d'y téléverser des fichiers et télécharger des fichiers selon les requêtes des utilisateurs. Le tableau 4.2 récapitule l'ensemble de ces commandes. Nous notons que *param_con* sont les paramètres de connexion au serveur Swift : « –A http ://adresseIPServeurSwift :8080/auth/v1.0 –U utilisateur -K motDePasse »

Tableau 4.2 Commandes utilisées

Opération	Description	Exemple
post	Met à jour les méta données pour le compte, le conteneur et l'objet. Post crée un conteneur s'il n'existe pas.	swift param_con post nom_conteneur
stat	Affiche les informations du compte, conteneur ou objet.	swift param_con stat nom_conteneur
delete	Supprime un conteneur ou les objets présents dans un conteneur.	swift param_con delete nom_conteneur nom_fichier
download	Télécharge les objets d'un conteneur.	swift param_con download nom_conteneur nom_fichier
upload	Téléverse les objets vers le conteneur.	swift param_con upload nom_conteneur nom_fichier
list	Liste les conteneurs du compte ou les objets d'un conteneur.	swift param_con list nom_conteneur

- **Spécification de l'échéance d'une requête :**

  Comme nous l'avons mentionné précédemment, les utilisateurs doivent spécifier une échéance pour récupérer une donnée. Nous proposons la formule suivante pour définir l'échéance d_i d'une requête i :

$$d_i = t_i + (1+\beta) \times r_i^{avg} \quad \forall 1 \leq i \leq N \tag{4.1}$$

  où t_i est l'instant auquel une requête i est générée et β est un pourcentage choisi par l'utilisateur qui spécifie combien de retard il peut tolérer pour une requête i. La variable r_i^{avg} représente le temps de transfert moyen enregistré en fonction des requêtes précédentes générées par les différents utilisateurs de l'entreprise. Cette formule permet de fixer un délai raisonnable supérieur au temps de transfert dans le cas idéal (où il n'y a pas de temps d'attente) et de satisfaire l'utilisateur qui peut tolérer un pourcentage de retard.

- **Surveillance de la bande passante disponible :**

  Afin de surveiller la bande passante disponible, nous avons codé un script en langage Python qui utilise la bibliothèque psutil pour mesurer le total d'octets envoyés et reçus au niveau du système de l'utilisateur.

  psutil [1] (i.e. utilitaires de processus et de système) : est une bibliothèque multi-plateforme permettant de récupérer des informations sur les processus en cours et l'utilisation du système (CPU, mémoire, disques, réseau, capteurs) en Python. Elle est utile principalement pour la surveillance du système, le profilage et la limitation des ressources de processus et la gestion des processus en cours. psutil implémente de nombreuses fonctionnalités offertes par les outils de ligne de commande UNIX tels que : ps, top, lsof, netstat, ifconfig, tty, df, kill, pmap, gentil, ionice, iostat, iotop, uptime, qui, taskset, gratuit.

4.3 Évaluation du succès et de l'échec des requêtes

Une requête de transfert de données peut échouer ou bien aboutir. L'échec d'une requête peut être expliqué par la surcharge du serveur Swift au cas où il se trouve surchargé par un grand nombre de requêtes. En fait, le proxy est le composant de Swift qui reçoit les requêtes entrantes, comme expliqué au niveau du premier chapitre. Ce dernier peut traiter simultanément un nombre bien déterminé de requêtes qui est spécifié dans le fichier de configuration du proxy. Une fois qu'une requête arrive au serveur proxy, elle est gardée dans une file d'attente jusqu'à ce qu'un noeud de stockage soit disponible pour la traiter. Les requêtes réussies peuvent aboutir à un transfert de données effectué avant ou bien après les échéances spécifiés par les utilisateurs.

1. https ://pypi.python.org/pypi/psutil/

4.3.1 Taux de requêtes réussies respectant leurs échéances

Dans cette partie, nous nous intéressons à évaluer le taux de requêtes respectant leurs échéances, dans le cas où les requêtes sont réussies (Figure 4.3). Nous remarquons que les algorithmes

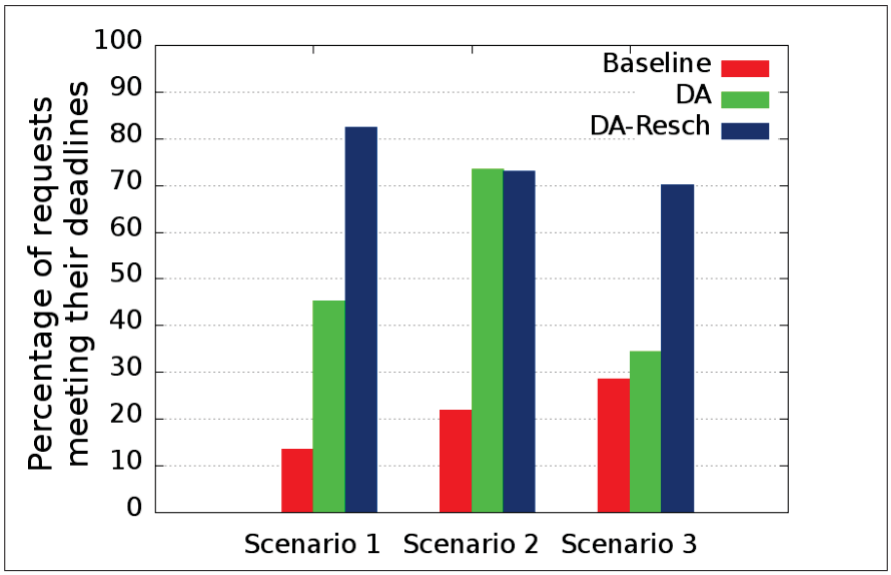

Figure 4.3 Pourcentage des requêtes respectant leurs échéances.

DA et DA-Resch surpassent significativement python-swiftclient (Baseline) en considérant les différents scénarios. En effet, python-swiftclient ne prend pas en compte la préférence des utilisateurs en termes d'échéance. Il utilise la stratégie PEPS afin de traiter les requêtes entrantes. De plus, comme l'algorithme DA-Resch permet de planifier de nouveau les requêtes au cas où elles peuvent être retardées, il est possible d'augmenter le nombre de requêtes respectant leurs échéances de plus de 30% par rapport à python-swiftclient et l'algorithme DA, comme le montre la figure 4.3.

4.3.2 Taux de requêtes échouées

Dans cette partie, nous étudions le nombre de requêtes échouées. Nous rappelons que l'échec survient lorsque Swift ne renvoie jamais la réponse à la requête puisqu'il est surchargé suite à la réception d'un grand nombre de requêtes.

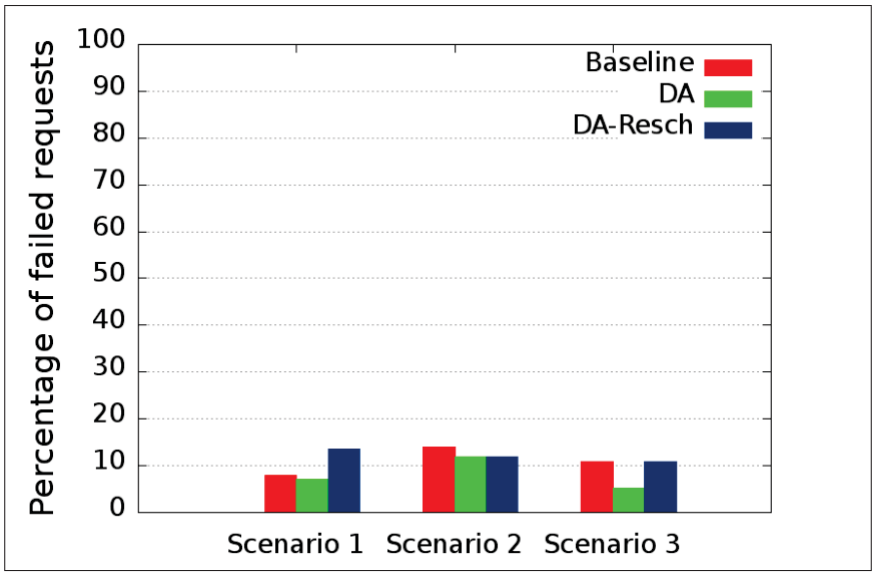

Figure 4.4 Pourcentage des requêtes échouées.

La figure 4.4 montre le pourcentage de requêtes échouées. Le pourcentage est d'environ 10% pour python-swiftclient et l'algorithme DA. Nous remarquons que DA a moins de requêtes échouées par rapport à python-swiftclient dans tous les scénarios car Swift n'est pas surchargé par les requêtes entrantes. En fait, l'algorithme d'ordonnancement DA retarde les requêtes et les soumet à Swift dans différents créneaux temporels plutôt que de les envoyer ensemble comme il est le cas avec python-swiftclient. Cela réduit la charge de travail de Swift et optimise l'utilisation de la bande passante. Nous remarquons également que DA-Resch engendre plus de requêtes échouées que DA. Ceci est dû au fait que DA-Resch pénalise certaines requêtes (en les retardant) afin de maximiser le nombre de requêtes qui respectent leurs échéances comme le montre la figure 4.3.

4.3.3 Taux de requêtes réussies ne respectant pas leurs échéances

Dans cette partie, nous étudions le nombre de requêtes qui ont réussi mais sans respecter leurs échéances. La figure 4.5 montre que plus de 50% des requêtes ont été réussies en utilisant python-swiftclient mais sans respecter leurs échéances. L'algorithme DA améliore légèrement

ce résultat mais DA-Resch parvient à minimiser le pourcentage de ces requêtes et augmente le nombre de requêtes qui réussissent et respectent l'échéance requise (Fig. 4.3).

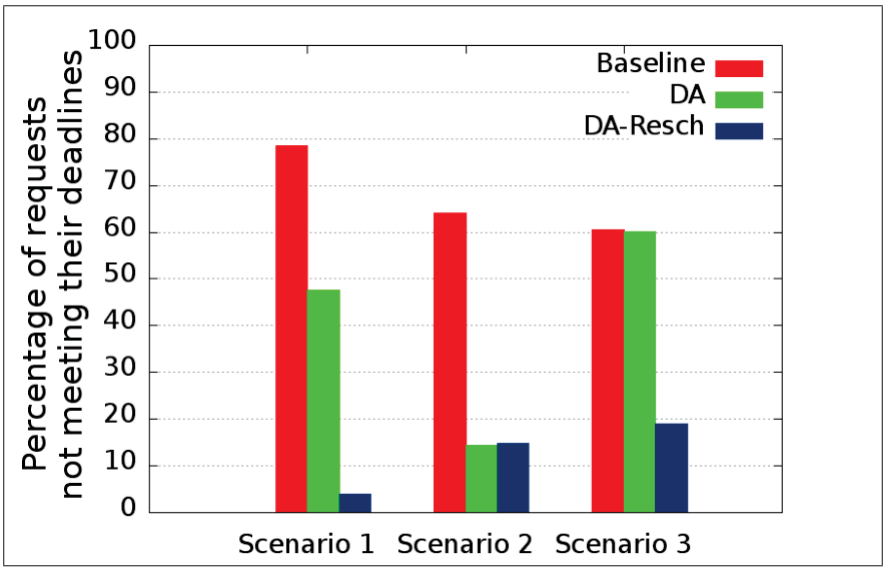

Figure 4.5 Pourcentage des requêtes ne respectant pas leurs échéances.

4.4 Évaluation de la consommation de bande passante disponible

Dans cette partie, nous évaluons la moyenne de la consommation de bande passante mesurée durant le transfert de données ainsi que la distribution de la consommation de bande passante dans le temps.

4.4.1 Évaluation de la consommation moyenne de bande passante disponible

La figure 4.6 montre que la consommation moyenne de bande passante est plus élevée pour DA et DA-Resch.

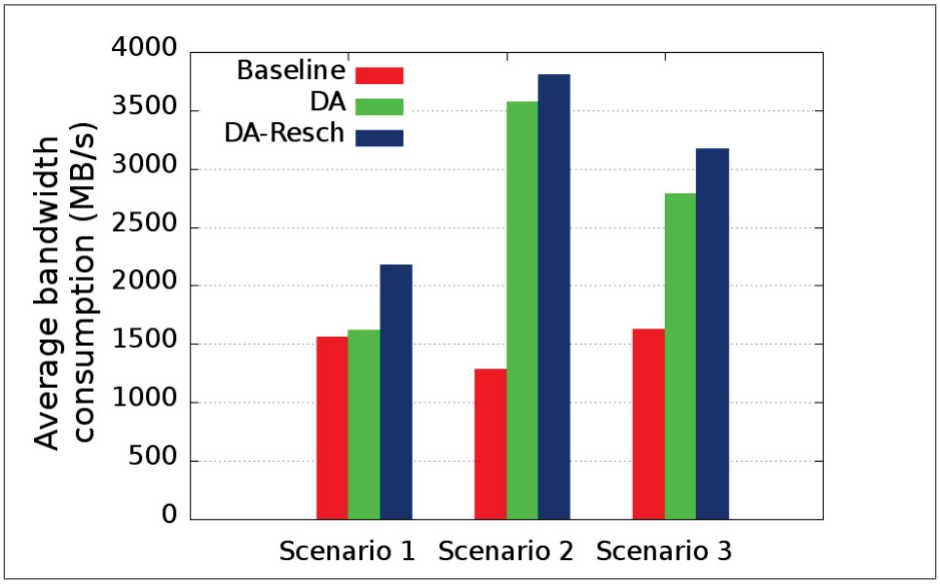

Figure 4.6 Consommation moyenne de bande passante.

Contrairement au projet python-swiftclient, le mécanisme de planification dans DA et DA-Resch permet de prioriser les requêtes en fonction des échéances tout en maximisant le nombre de threads en utilisant la bande passante disponible pendant chaque créneau temporel. Par conséquent, ces algorithmes permettent de maximiser l'utilisation de la bande passante.

4.4.2 Évaluation de la consommation de bande passante disponible au fil du temps

Les figures 4.7, 4.8 et 4.9 montrent la consommation de la bande passante au fil du temps. Nous pouvons clairement remarquer que, pour tous les scénarios, le temps de transfert de toutes les requêtes est aux alentours de 200s pour DA et DA-Resch par rapport à 1000s pour python-swiftclient. Cela implique une amélioration de 80% concernant le temps de transfert comparé au projet python-swiftclient de OpenStack. Nous pouvons également voir que DA et DA-Resch maximisent la consommation de la bande passante en transférant les données. Cela est dû à la maximisation du nombre de requêtes planifiées pour chaque créneau temporel sans

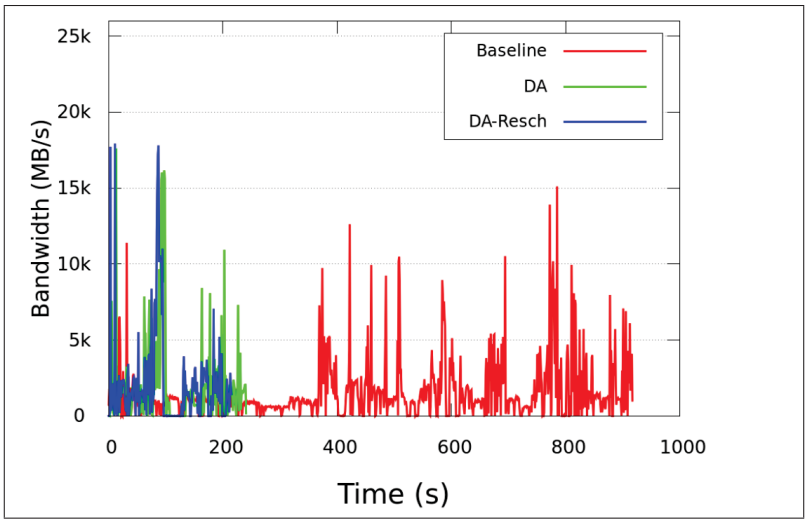

Figure 4.7 Consommation de bande passante au fil du temps - Scénario 1

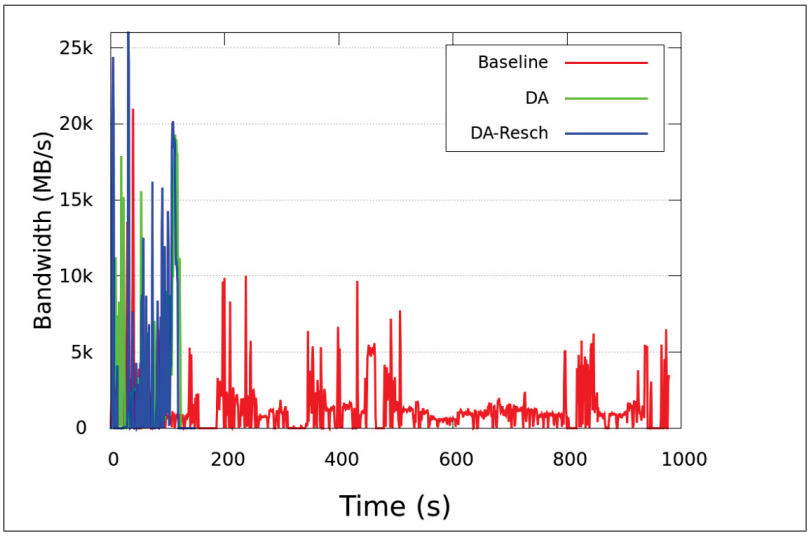

Figure 4.8 Consommation de bande passante au fil du temps - Scénario 2

dépasser la valeur maximale de bande passante.

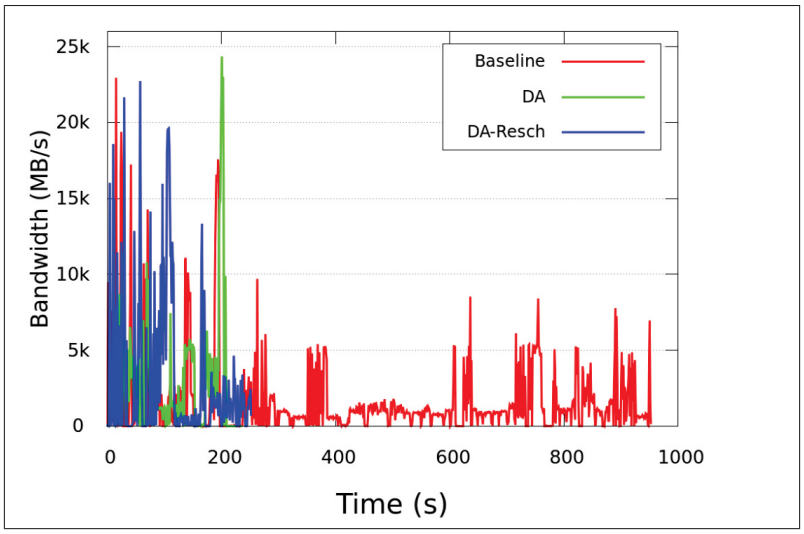

Figure 4.9 Consommation de bande passante au fil du temps - Scénario 3

4.5 Évaluation des retards enregistrés pour les différentes requêtes

Comme la figure 4.5 ne montre pas à quel point le temps de transfert de données est proche de l'échéance, nous présentons la fonction de répartition des retards illustrée par les figures 4.10, 4.11 et 4.12. Nous rappelons que le retard enregistré par une requête est la différence entre la fin de transmission des données demandées et l'échéance de la requête.

Les figures montrent que plus de 60% et 80% des retards de requête valent zéro ou moins (c'est-à-dire négatifs) pour DA et DA-Resch, respectivement, signifiant que la plupart des requêtes respectent leurs échéances. Nous pouvons également remarquer que les deux algorithmes, DA et DA-Resch, surpassent significativement l'algorithme de OpenStack dans tous les scénarios. Avec les deux algorithmes, le retard le plus élevé ne dépasse pas 200 secondes alors qu'il dépasse 800 secondes avec l'algorithme de OpenStack.

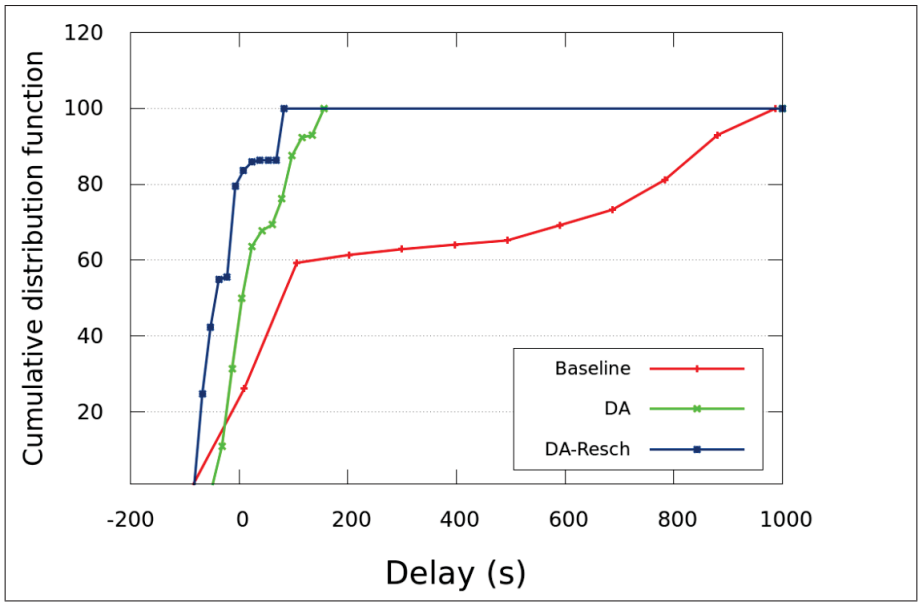

Figure 4.10 Fonction de répartition des retards - Scénario 1

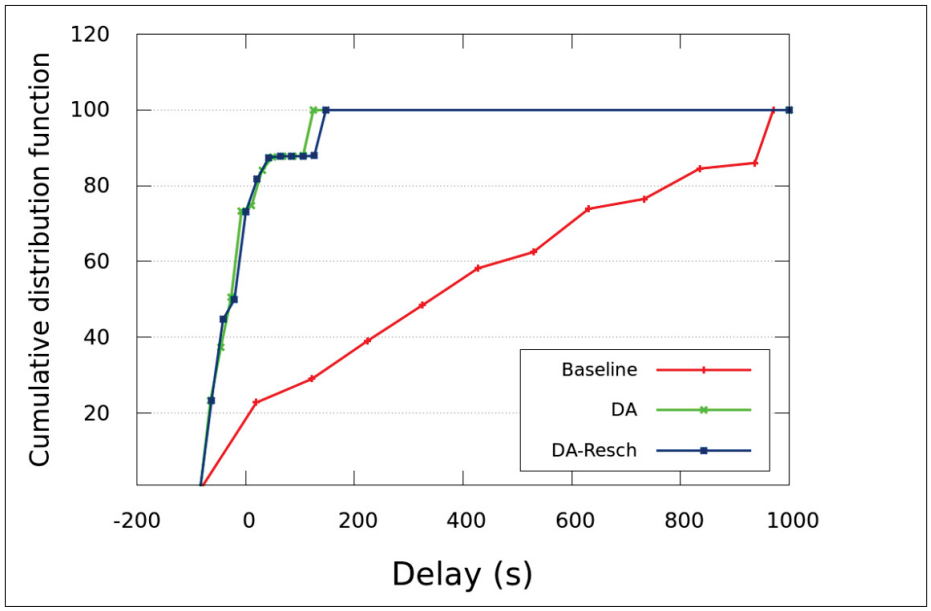

Figure 4.11 Fonction de répartition des retards - Scénario 2

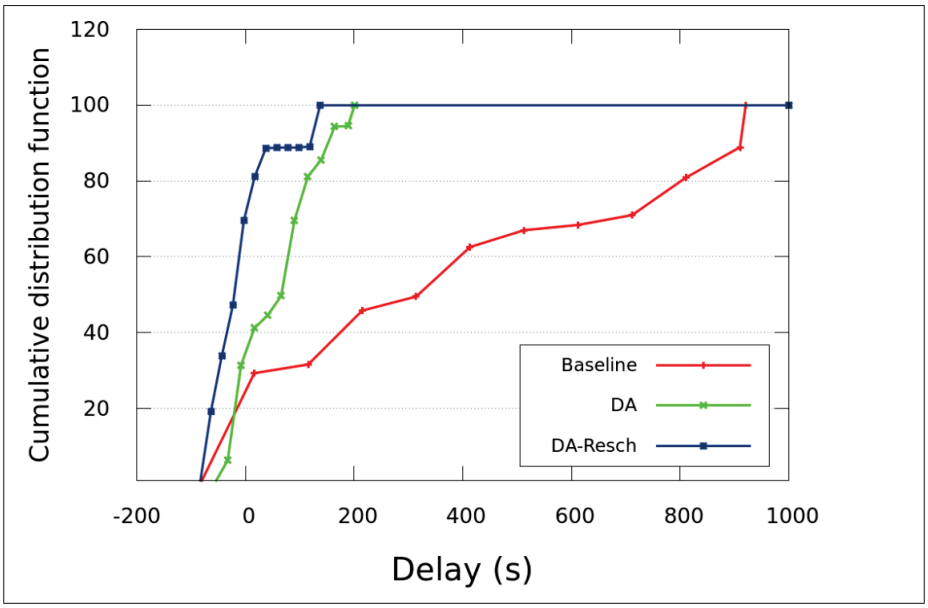

Figure 4.12 Fonction de répartition des retards - Scénario 3

4.6 Conclusion

Dans ce chapitre, nous avons présenté les expériences réalisées et avons discuté les résultats obtenus. Nous avons montré à travers des expérimentations réelles que les deux algorithmes surpassent la solution de OpenStack officielle pour les utilisateurs de Swift. En effet, nous avons trouvé que les données relatives à plus de 70% des requêtes sont récupérées par les utilisateurs avant leurs échéances en utilisant DA-Resch pour les différents scénarios considérés. Nous avons également prouvé que DA et DA-Resch permettent de maximiser la consommation de bande passante disponible en les comparant avec la solution de Swift dédiée pour les clients, baptisée pyhon-swiftclient. Ceci est du au fait que le mécanisme d'ordonnancement dans DA et DA-Resch permet de prioriser les requêtes en considérant leurs échéances tout en maximisant le nombre de threads utilisant la bande passante disponible pour chaque créneau temporel. Nos algorithmes réussissent également à réduire la durée de transfert des requêtes. Les expériences ont montré une réduction de 80% de la durée totale de transferts des données par rapport à python-swiftclient. Finalement, après avoir mesuré les retards enregistrés par chaque requête, nous avons trouvé qu'en utilisant nos algorithmes le retard le plus important ne dépasse pas 200s alors qu'il dépasse 800s avec python-swiftclient.

CONCLUSION ET PERSPECTIVES

La quantité de données générées par les différentes applications de l'infonuage est en croissance continue. Cela accentue la demande des utilisateurs du transfert de données massives à diverses fins telles que le stockage de données, le traitement et l'analyse. Ce besoin a orienté l'intérêt de la communauté des chercheurs travaillant sur la gestion des données vers un nouveau champs émergeant qui s'intéresse à l'ordonnancement de requêtes de données dans le but d'optimiser le transfert de données en termes de temps et de ressources.

Des travaux récents ont abordé la gestion du transfert de données dans les systèmes de stockage de l'infonuage. Cependant, les solutions existantes n'ont pas pris en considération les préférences des utilisateurs en termes de temps de transfert. Dans ce mémoire, nous avons proposé deux algorithmes d'ordonnancement de données DA et DA-Resch qui tiennent en compte les échéances des requêtes afin de satisfaire aux exigences des utilisateurs tout en maximisant l'utilisation de la bande passante disponible. Nous avons implémenté ces algorithmes en langage Python, par la suite les ont intégrés dans Swift (la solution de OpenStack pour le stockage des données) dans le but d'optimiser sa politique de transfert de données.

Notre solution vise à fournir le transfert de données entre les utilisateurs et les serveurs de stockage dans l'infonuage en tant que service où les utilisateurs peuvent planifier leurs requêtes à l'avance. Grâce à notre solution, les transferts de données sont désormais gérés par un composant intégré dans chaque client dans lequel les requêtes de transfert sont ordonnancées pour une meilleure performance en termes de consommation maximale de bande passante, de temps de transfert de données minimal et de nombre maximal de requêtes satisfaisant leurs échéances.

Nous avons montré à travers des expérimentations effectuées sur une plateforme réelle que les deux algorithmes surpassent la solution Swift de OpenStack. En effet, nous avons trouvé que les données relatives à plus de 70% des requêtes sont récupérées par les utilisateurs avant leurs échéances en utilisant DA-Resch pour les différents scénarios considérés. Nous avons

également prouvé que DA et DA-Resch permettent de maximiser la consommation de bande passante disponible en les comparant avec la solution de Swift dédiée pour les clients, baptisée pyhon-swiftclient. Cela est dû au fait que le mécanisme d'ordonnancement dans DA et DA-Resch permet de prioriser les requêtes en considérant leurs échéances tout en maximisant le nombre de transferts utilisant la bande passante disponible pour chaque créneau temporel. Nos algorithmes réussissent également à réduire la durée de transfert des requêtes. Les expériences ont montré une réduction de 80% de la durée totale de transferts des données par rapport à python-swiftclient. Finalement, nous avons trouvé qu'en utilisant nos algorithmes, le retard le plus important ne dépasse pas 200s alors qu'il dépasse 800s avec python-swiftclient.

Comme perspectives futures, nous pouvons nous orienter vers plusieurs directions prometteuses. Nous prévoyons proposer un système basé sur la négociation qui trouve un accord commun entre les utilisateurs et les fournisseurs de stockage afin d'attribuer l'échéance approprié à chaque requête selon la priorité de la donnée à récupérer en question. Une autre voie intéressante serait d'étudier l'effet de la variation du nombre de threads et de la durée du créneau temporel sur les performances visées (citées plus haut) des transferts dans le but d'ajuster ces paramètres dynamiquement avant les transferts de données. Nous pouvons également proposer un modèle de coût qui décide du montant dû à l'utilisateur selon la performance moyenne des transferts de données qui prend en considération, entre autres, la durée du transfert, le taux de requêtes satisfaisant leurs échéances.

BIBLIOGRAPHIE

Abonnés_Netflix. [Consulté en Septembre 2016]. Netflix nears 100 million subscribers. Repéré à http://money.cnn.com/2017/04/17/technology/netflix-subscribers/index.html.

Alhamad, M., Dillon, T. & Chang, E. (2010). Conceptual SLA framework for cloud computing. *Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on*, pp. 606–610.

Amar Kapadia,Sreedhar Varma, K. R. (2014). *Implementing Cloud Storage with OpenStack Swift*.

Amazon_S3. [Consulté en Octobre 2016]. Object Size Limit. Repéré à https://aws.amazon.com/blogs/aws/amazon-s3-object-size-limit/.

Amazon_S3. [Consulté en Janvier 2017]. SLA Amazon S3. Repéré à https://aws.amazon.com/fr/s3/sla/.

Amazon_S3. [Consulté en Août 2016]. (2013). Stockage d'objets conçu pour stocker et récupérer n'importe quelle quantité de données, n'importe où. Repéré à https://aws.amazon.com/fr/s3/.

Architecture_OpenStack. [Consulté en Janvier 2017]. Product Documentation for Red Hat OpenStack Platform. Repéré à https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform/2/html/Release_Notes/sect-OpenStackNotes-Intro_Chapter-Test_Section_3.html.

Azure. [Consulté en Décembre 2016]. SLA Microsoft Azure. Repéré à https://azure.microsoft.com/en-us/support/legal/sla/summary/.

Ballani, H., Costa, P., Karagiannis, T. & Rowstron, A. (2011). Towards predictable datacenter networks. *ACM SIGCOMM Computer Communication Review*, 41(4), 242–253.

Barr, J. [Consulté en Août 2016]. (2013). Amazon S3 – More Than 449 Billion Objects. Repéré à https://aws.amazon.com/fr/blogs/aws/amazon-s3-more-than-449-billion-objects/.

Chen, Y., Jain, S., Adhikari, V. K., Zhang, Z.-L. & Xu, K. (2011). A first look at inter-data center traffic characteristics via yahoo ! datasets. *INFOCOM, 2011 Proceedings IEEE*, pp. 1620–1628.

Documentation_Swift. [Consulté en Janvier 2017]. Swift components. Repéré à https://docs.openstack.org/swift/pike/admin/objectstorage-components.html#proxy-servers.

Dropbox. [Consulté en Septembre 2016]. Dropbox. Repéré à https://www.dropbox.com/.

ebay. [Consulté en Septembre 2016]. ebay. Repéré à www.ebay.com/.

Fang Liu,Jin Tong, J. M. [Consulté en Décembre 2016]. (2016). NIST Cloud Computing Reference Architecture. Repéré à https://www.nist.gov/publications/nist-cloud-computing-reference-architecture.

Garey, M. R. & Johnson, D. S. (2002). *Computers and intractability*. wh freeman New York.

Google_drive. [Consulté en Septembre 2016]. Google_drive. Repéré à https://drive.google.com/.

Guo, C., Li, Y. & Wu, Z. (2016). SLA-DO : A SLA-Based Data Distribution Strategy on Multiple Cloud Storage Systems. *Parallel and Distributed Systems (ICPADS), 2016 IEEE 22nd International Conference on*, pp. 602–609.

Guo, C., Lu, G., Wang, H. J., Yang, S., Kong, C., Sun, P., Wu, W. & Zhang, Y. (2010). Secondnet : a data center network virtualization architecture with bandwidth guarantees. *Proceedings of the 6th International COnference*, pp. 15.

Heidari, P. & Kanso, A. (2016). QoS Assurance through Low Level Analysis of Resource Utilization of the Cloud Applications. *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*, pp. 228–235.

Hong, C.-Y., Kandula, S., Mahajan, R., Zhang, M., Gill, V., Nanduri, M. & Wattenhofer, R. (2013). Achieving high utilization with software-driven WAN. *ACM SIGCOMM Computer Communication Review*, 43(4), 15–26.

Hou, B., Chen, F., Ou, Z., Wang, R. & Mesnier, M. (2017). Understanding I/O Performance Behaviors of Cloud Storage from a Client's Perspective. *ACM Transactions on Storage (TOS)*, 13(2), 16.

Instagram. [Consulté en Septembre 2016]. Instagram. Repéré à https://www.instagram.com/.

Janet, J., Balakrishnan, S. & Murali, E. (2016). Improved data transfer scheduling and optimization as a service in cloud. *IEEE Int. Conference on Information Communication and Embedded Systems (ICICES)*, pp. 1–3.

JOS. [Consulté en Octobre 2016]. Java OpenStack Storage. Repéré à https://github.com/javaswift/joss.

Joshi, G., Soljanin, E. & Wornell, G. (2015a). Efficient replication of queued tasks for latency reduction in cloud systems. *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, pp. 107–114.

Joshi, G., Soljanin, E. & Wornell, G. (2015b). Efficient replication of queued tasks for latency reduction in cloud systems. *IEEE Annual Allerton Conf. on Communication, Control, and Computing*.

Kandula, S., Menache, I., Schwartz, R. & Babbula, S. R. (2014). Calendaring for wide area networks. *ACM SIGCOMM computer communication review*, 44(4), 515–526.

Kosar, T., Arslan, E., Ross, B. & Zhang, B. (2013). Storkcloud : Data transfer scheduling and optimization as a service. *ACM workshop on Scientific cloud computing*.

Laoutaris, N., Sirivianos, M., Yang, X. & Rodriguez, P. (2011). Inter-datacenter bulk transfers with netstitcher. *ACM SIGCOMM Computer Communication Review*, 41(4), 74–85.

Lin, Y. & Wu, Q. (2013). Complexity analysis and algorithm design for advance bandwidth scheduling in dedicated networks. *IEEE/ACM Transactions on Networking (TON)*, 21(1), 14–27.

Linden, G. (2006). Make data useful. *Stanford CS345 Talk*.

Liu, G., Shen, H. & Yu, L. (2016). Towards Deadline Guaranteed Cloud Storage Services. *IEEE Int. Conf. on Cloud Computing (CLOUD)*.

Liu, W., Tieman, B., Kettimuthu, R. & Foster, I. (2010). A data transfer framework for large-scale science experiments. *ACM International Symposium on High Performance Distributed Computing*.

Netflix. [Consulté en Septembre 2016]. Netflix. Repéré à https://www.netflix.com/.

Noormohammadpour, M., Raghavendra, C. S., Rao, S. & Madni, A. M. (2016). RCD : Rapid Close to Deadline Scheduling for datacenter networks. *World Automation Congress (WAC), 2016*, pp. 1–6.

OpenStack. [Consulté en Janvier 2017]. What is OpenStack ? Repéré à https://www.openstack.org/software/.

OpenStack_Swift. [Consulté en Septembre 2016]. OpenStack Swift Object Storage Service. Repéré à http://swift.openstack.org/.

OpenStack_Swift_Community. [Consulté en Septembre 2016]. OpenStack Swift. Repéré à https://wiki.openstack.org/.

ovh. [Consulté en Décembre 2016]. ovh cloud storage. Repéré à https://www.ovh.com/ca/fr/cloud-public/storage/object-storage/.

Python-swiftclient. [Consulté en Octobre 2016]. Python-swiftclient. Repéré à https://docs.openstack.org/developer/python-swiftclient/introduction.html/.

Qiu, M. M., Zhou, Y. & Wang, C. (2013). Systematic analysis of public cloud service level agreements and related business values. *Services Computing (SCC), 2013 IEEE International Conference on*, pp. 729–736.

Rackspace. [Consulté en Septembre 2016]. Rackspace. Repéré à https://www.rackspace.com/.

RSwift. [Consulté en Novembre 2016]. RSwift. Repéré à https://github.com/pandemicsyn/RSwift/.

Sanjay, G., Howard, G. & Shun-Tak, L. (2003). The Google file system. *ACM SOSP*.

Saurabh. [Consulté en Décembre 2016]. CLOUD PIZAZZ. Repéré à https://sawalia.wordpress.com/2014/08/17/cloud-pizazz/.

Schurman, E. & Brutlag, J. (2009). *The User and Business Impact of Server Delays, Additional Bytes, and HTTP Chunking in Web Search*.

Sivaram, N. [Consulté en Décembre 2016]. OpenStack Cinder Volume Backups - Store and Restore Using Object Storage. Repéré à https://www.snia.org/sites/default/files/SDCIndia/2016/Presentations/Using%20Object%20Storage%20to%20store%20and%20restore%20Cinder%20Volume%20backups.pdf.

Swift. [Consulté en Janvier 2017]. Swift documentation. Repéré à https://docs.openstack.org/swift/latest/.

SwiftBox. [Consulté en Novembre 2016]. SwiftBox. Repéré à https://github.com/suniln/SwiftBox/.

Varalakshmi, P., Thangavel, M., Nithya, K., Priya, T. & Sakthya, D. (2013). EDSRPPC : An efficient data storage and retrieval through personalization and prediction in cloud. *Advanced Computing (ICoAC), 2013 Fifth International Conference on*, pp. 413–418.

Wang, D., Joshi, G. & Wornell, G. (2015). Using straggler replication to reduce latency in large-scale parallel computing. *ACM SIGMETRICS Performance Evaluation Review*, 43(3), 7–11.

Wang, Y., Wu, C. Q. & Hou, A. (2016). On periodic scheduling of bandwidth reservations with deadline constraint for big data transfer. *Local Computer Networks (LCN), 2016 IEEE 41st Conference on*, pp. 224–227.

Xie, D., Ding, N., Hu, Y. C. & Kompella, R. (2012). The only constant is change : Incorporating time-varying network reservations in data centers. *ACM SIGCOMM Computer Communication Review*, 42(4), 199–210.

Yahoo. [Consulté en Avril 2017]. Yahoo ! dataset de Webscope : Informations statistiques sur les fichiers et le modèle d'accès aux fichiers dans l'un des clusters Yahoo. Repéré à http://webscope.sandbox.yahoo.com/catalog.php?datatype=s/.

Yassine, A., Shirehjini, A. A. N. & Shirmohammadi, S. (2016). Bandwidth On-demand for Multimedia Big Data Transfer across Geo-Distributed Cloud Data Centers. *IEEE Transactions on Cloud Computing*.

Youtube. [Consulté en Août 2016]. Youtube. Repéré à https://www.youtube.com/.

Zhang, H., Chen, K., Bai, W., Han, D., Tian, C., Wang, H., Guan, H. & Zhang, M. (2017). Guaranteeing Deadlines for Inter-Data Center Transfers. *IEEE/ACM Transactions on Networking (TON)*, 25(1), 579–595.