

Livrable Final

Équipe N°8

Mai 2023

Livrable Final

Dans le cadre du projet 2CS

Option : Systèmes Informatiques (SIQ)

Mise en place d'un CDN (Content Delivery Network)

Réalisé par :

BELLI Bilal (CE)

ABOUD Rayane

HASSANI Mehdi

SAFTA Abir

BENABDELATIF Maya

RABAHSIDHOUM Mohamed Amine

Encadré par :

M. AMROUCHE Hakim

M. HAMANI Nacer

Promotion : 2022/2023

Table des matières

Introduction	4
Problématique	5
1 Architecture de notre solution CDN	7
1.1 Architecture de la région d'Alger	7
1.2 Architecture des régions d'Oran, Constantine et Ouargla	8
2 Solutions proposées	9
2.1 Première approche : HAProxy	9
2.1.1 Définition et fonctionnalités HAProxy	9
2.1.2 Utilisation courante dans les infrastructures réseau	9
2.1.3 Avantages généraux d'HAProxy	9
2.1.4 Inconvénients de HAProxy	9
2.1.5 Mécanismes de redirection dans HAProxy	10
2.2 Deuxième approche : Application avec Flask	10
2.2.1 Structure de l'application	11
2.2.2 Communication entre les composants	11
3 Outils utilisé pour la maintenance et la supervision	12
3.1 configuration et maintenance de NAS	12
3.2 monitoring et supervision des serveurs	12
4 Conception des tests	14
4.1 Critères de choix des tests	14
4.2 Plan de test	15
4.2.1 Partie 1 : Tests unitaires	15
4.2.2 Partie 1 : Tests unitaires	15
5 Réalisation des tests et évaluation des résultats	16
5.1 Plan de tests spécifique à chaque solution	16
5.1.1 Test de résolution de nom de domaine par le serveur DNS	16
5.1.2 Test de redirection de HAproxy	16
5.1.3 Test de redirection de l'application	18
5.1.4 Tests sur le NAS (Network Attach Storage)	23
5.2 Tests Intensifs	26
5.2.1 Test de résistance du serveur de redirection	26
5.2.2 Comparaison des temps de réponse de chaque solution	28
5.2.3 Test sur la répartition des requêtes selon leur adresses	29
6 Résumé du travaille effectué dans le projet	34
Conclusion	35

Table des figures

1.1	L'architecture finale des composants physiques de la solution	8
2.1	Distribution de la localisation de l'adresse IP tout au long du pays	11
3.1	GUI pour configuration du NAS	12
3.2	GUI pour monitoring du serveur	13
3.3	Exemple de vérification de la disponibilité du serveur à Oran	13
5.1	Résultat résolution de nom par nslookup de la solution HAproxy	16
5.2	Résultat du Test n°1 : client situé à Annaba	17
5.3	Résultat du Test n°2 : client situé à Sidi Bel Abbes	17
5.4	Résultat du Test n°3 : client situé à Gardaia	18
5.5	Résultat du Test n°4 : Disponibilité et de la charge des serveurs	18
5.6	Script de configuration (Parties 1 et 2)	19
5.7	Lancement de l'application de redirection	19
5.8	Fonctionnalité de la charges des serveurs	20
5.9	État du serveur load balancer DNS (lui-même étant l'origine)	20
5.10	Disponibilité de tous les serveurs	20
5.11	Adresse IP source (client)	21
5.12	Requête vers le serveur de redirection (cas 1)	21
5.13	Redirection vers le serveur d'Oran	21
5.14	Visualisation du score attribuée par l'application (cas 1)	21
5.15	Serveur d'Oran en panne (désactivé)	22
5.16	Requête vers le serveur de redirection (cas 2)	22
5.17	Redirection vers le serveur de Constantine	22
5.18	Visualisation du score attribuée par l'application (cas 2)	22
5.19	Visualisation des nouvelles scores attribuées par l'application	23
5.20	Serveur NAS monter sur serveur origine	23
5.21	Affichage du contenu du Serveur NAS	23
5.22	Script responsable de la sauvegarde du contenu dans le NAS	23
5.23	Contenu du serveur NAS (PDFs) avant de réaliser le test	24
5.24	Interface de connexion et d'importation d'un fichier PDF	24
5.25	Contenu du serveur NAS (PDFs) après avoir réalisé le test	24
5.26	Les trames qui sont transmises au NAS en écriture	24
5.27	lecture d'un fichier PDF	25
5.28	Les trames qui sont transmises au NAS en lecture	25
5.29	Résultats des deux solutions pour 10000 requêtes avec 100 en concurrence	26
5.30	Résultats des deux solutions pour 10000 requêtes avec 500 en concurrence	26
5.31	Résultats des deux solutions pour 10000 requêtes avec 1000 en concurrence	27
5.32	Courbes des temps de réponse des deux solutions (10 000 requêtes)	28
5.33	Adresses IP aléatoires	29
5.34	Redirection des clients vers des serveurs caches différents (cas a)	30
5.35	Redirection des clients vers des serveurs caches différents (cas b)	30
5.36	Redirection des clients vers des serveurs caches différents (cas c)	31
5.37	Redirection des clients vers des serveurs caches différents (cas d)	31

5.38	Redirection des clients vers des serveurs caches différents (cas e)	32
5.39	Redirection des clients vers des serveurs caches différents (cas f)	32

Introduction

Dans cette troisième partie de notre rapport, nous aborderons l'implémentation et la simulation de notre solution CDN pour l'optimisation de la diffusion de contenu. Nous décrirons les différentes étapes et outils utilisés pour mettre en œuvre notre plateforme.

La première étape consiste à déployer les répliques du CDN dans les universités partenaires. Nous avons sélectionné soigneusement les emplacements stratégiques pour ces répliques afin de garantir une couverture optimale. En utilisant des serveurs dédiés, nous avons configuré les répliques pour stocker en cache les ressources pédagogiques et les mettre à jour régulièrement pour assurer la pertinence des cours.

Une fois les répliques déployées, nous avons procédé à la configuration du système de gestion du CDN. Nous avons utilisé des outils et des logiciels spécialisés pour surveiller et contrôler la diffusion du contenu. Cela nous permet de gérer efficacement les demandes des utilisateurs, de surveiller les performances du réseau et de garantir une expérience fluide lors de l'accès aux cours en ligne.

Pour évaluer les performances de notre solution CDN, nous avons réalisé des simulations en utilisant des charges de trafic réalistes. Nous avons généré des demandes de ressources pédagogiques à partir de différents emplacements et mesuré les temps de réponse ainsi que la vitesse de diffusion du contenu. Ces simulations nous ont permis de valider l'efficacité de notre CDN et d'apporter les ajustements nécessaires pour améliorer les performances.

Nous avons également effectué des tests de charge pour évaluer la capacité du CDN à gérer des pics de demande, notamment lors de périodes d'affluence ou de téléchargements massifs de ressources. Les résultats de ces tests nous ont permis de dimensionner notre infrastructure de manière adéquate, en veillant à ce qu'elle puisse répondre aux besoins des utilisateurs dans des conditions de charge élevée.

Problématique

Nous nous confrontons à la problématique de définir le meilleur choix d'architecture détaillée à mettre en place pour optimiser les performances d'un CDN. De plus, nous devons élaborer une approche pratique pour implémenter cette solution et garantir une qualité optimale. Pour ce faire, il sera nécessaire de concevoir et réaliser des tests fondés sur des critères précis afin d'évaluer cette architecture, le plan général de notre rapport sera comme suit :

- **Définition de l'architecture détaillée du CDN :**

Dans cette section, nous examinerons les différents éléments de base nécessaires pour mettre en place une architecture détaillée du CDN. Cela inclut la sélection des serveurs d'origine, la configuration des serveurs de cache, ainsi que la mise en œuvre d'une gestion du routage des requêtes. Nous expliquerons également comment ces composants interagissent pour optimiser la performance globale du CDN.

- **Déploiement pratique de l'architecture du CDN :**

Une fois que l'architecture détaillée du CDN est conçue, il est essentiel de comprendre les différentes étapes et considérations pour un déploiement pratique. Nous ferons un passage pour expliquer les différents logiciels utilisés, la configuration réseau, la gestion des mises à jour et la sécurisation de notre infrastructure. Des exemples concrets et des bonnes pratiques seront fournis pour illustrer ces concepts.

- **Conception des tests basés sur des critères précis :**

Pour évaluer l'efficacité de l'architecture déployée, des tests rigoureux doivent être réalisés. Dans cette section, nous expliquerons comment concevoir des tests basés sur des critères précis tels que la latence, le taux de transfert, la répartition de charge, cohérence des données, la disponibilité, etc. Nous détaillerons également les outils et les méthodologies utilisés pour mesurer ces critères de manière précise et fiable.

- **Réalisation des tests et évaluation des résultats :**

Une fois les tests conçus, nous passerons à leur réalisation et à l'évaluation des résultats obtenus. Nous analyserons les mesures et les indicateurs de performance collectés lors des tests et les comparerons aux critères préalablement définis. Nous discuterons des résultats obtenus, des éventuelles limitations rencontrées et des recommandations pour améliorer l'architecture du CDN en fonction des résultats des tests.

- **Pour conclure :**

Nous soulignerons l'importance d'une architecture détaillée pour optimiser la performance d'un CDN. Nous mettrons en évidence les étapes essentielles pour déployer

cette architecture de manière pratique et expliquerons l'importance de mettre en place des tests afin de garantir un bon fonctionnement.

Chapitre 1

Architecture de notre solution CDN

Ce rapport présente en détail l'architecture du système de diffusion de contenu (CDN) de notre équipe. L'architecture a été conçue pour garantir une distribution efficace des ressources, une réduction de la latence et une haute disponibilité pour les utilisateurs finaux.

Il s'agit d'une architecture géographiquement distribuée, comprenant quatre régions principales : **Alger**, **Oran**, **Constantine** et **Ouargla**. Chaque région dispose de composants spécifiques qui contribuent au bon fonctionnement du CDN.

1.1 Architecture de la région d'Alger

Dans la région d'Alger, les composants clés de l'architecture comprennent :

- **Serveur d'origine** : Le serveur de sauvegarde est un site opérationnel en permanence qui surveille le serveur d'origine en effectuant des vérifications régulières. En cas de panne du serveur d'origine, le serveur de sauvegarde prend le relais pour assurer la continuité du service.
- **Serveur backup** : Le serveur d'origine héberge le site client (www.edustream.dz) et sert les utilisateurs qui se trouvent à proximité géographique. Il est responsable de fournir les fichiers, vidéos et documents demandés par les utilisateurs. L'application web est développée en utilisant le backend Python avec le framework Flask. Les fonctionnalités principales de l'application permettent aux étudiants d'accéder à leurs cours, documents et vidéos, tandis que les enseignants peuvent ajouter de nouveaux contenus à la plateforme. Le serveur d'origine intègre également Nginx, qui agit comme un hôte et un cache pour les vidéos et PDF stockés dans le NAS.
- **Network Area Storage (NAS)** : Le serveur de sauvegarde est un site opérationnel en permanence qui surveille le serveur d'origine en effectuant des vérifications régulières. En cas de panne du serveur d'origine, le serveur de sauvegarde prend le relais pour assurer la continuité du service.
- **Serveur de redirection** : Le serveur de redirection est essentiel pour le fonctionnement du CDN. Il est chargé de rediriger les requêtes des utilisateurs vers le serveur approprié en fonction de leur localisation géographique ainsi que l'état des serveurs. Deux solutions ont été envisagées pour cette fonction :

1. **Solution 1 :** HAProxy est un logiciel open-source qui agit en tant que serveur proxy et répartiteur de charge pour les applications web. Il permet de gérer le trafic réseau en distribuant les requêtes entrantes vers plusieurs serveurs d'application en fonction de règles de routage prédéfinies. L'utilisation de HAProxy présente plusieurs avantages, notamment la répartition de charge, la haute disponibilité, la sécurité et l'extensibilité. Cependant, la configuration de HAProxy peut être complexe et nécessite une expertise pour optimiser son utilisation. Il est important de noter que HAProxy représente un point de défaillance unique, et des mesures de haute disponibilité doivent être mises en place pour atténuer ce risque.
2. **Solution 2 :** Application de redirection personnalisée pour simplifier la configuration et répondre aux besoins spécifiques de notre système, notre équipe a développé une application de redirection personnalisée en Python. Cette application importe une base de données contenant les adresses IP et les localisations géographiques en coordonnées. Elle utilise ces informations pour rediriger les requêtes des utilisateurs vers les serveurs appropriés en fonction de leur position géographique. L'application prend également en compte la disponibilité des serveurs et leur charge pour assurer une distribution équilibrée des requêtes.

1.2 Architecture des régions d'Oran, Constantine et Ouargla

Ces régions abritent des serveurs caches situés dans différentes zones géographiques. Le serveur de redirection redirige les requêtes des utilisateurs vers ces serveurs en fonction de leur position géographique. Chaque serveur cache utilise Nginx, qui offre des fonctionnalités de mise en cache pour optimiser la distribution des ressources. Un fichier de configuration spécifique est utilisé pour contrôler le cache sur ces serveurs.

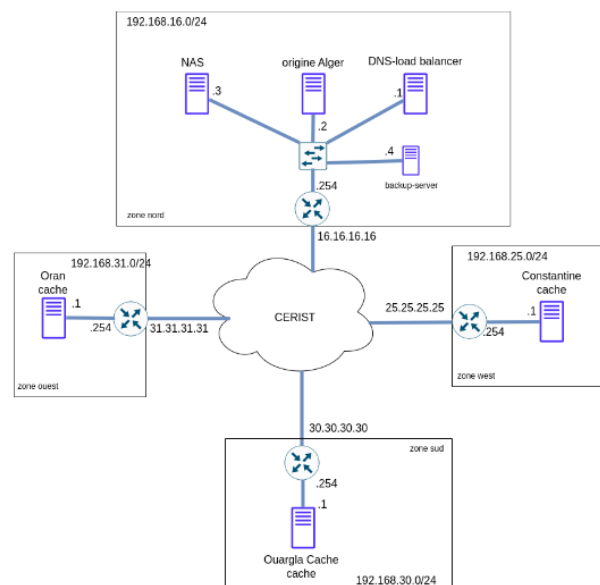


FIG. 1.1 : L'architecture finale des composants physiques de la solution

Chapitre 2

Solutions proposées

2.1 Première approche : HAProxy

2.1.1 Définition et fonctionnalités HAProxy

HAProxy est un logiciel open-source qui agit en tant que répartiteur de charge et proxy TCP/HTTP. Il permet d'équilibrer la charge sur les serveurs backend, d'améliorer la disponibilité, de garantir la scalabilité et de renforcer la sécurité d'une infrastructure réseau.

2.1.2 Utilisation courante dans les infrastructures réseau

HAProxy est largement utilisé dans les architectures à haute disponibilité, les environnements de cloud computing, les applications web à fort trafic et les infrastructures de microservices. Il s'intègre facilement avec différents outils et technologies du domaine des réseaux.

2.1.3 Avantages généraux d'HAProxy

HAProxy offre plusieurs avantages, notamment :

- Équilibrage de charge efficace pour une répartition optimale des demandes.
- Haute disponibilité grâce à la détection de pannes et au basculement automatique.
- Scalabilité permettant de faire face aux pics de trafic et d'ajouter/supprimer dynamiquement des serveurs.
- Sécurité renforcée grâce à la protection contre les attaques DDoS et au contrôle d'accès.
- Haute performance assurant un traitement rapide des requêtes et une optimisation du trafic.

2.1.4 Inconvénients de HAProxy

Malgré ses nombreux avantages, HAProxy présente certains inconvénients, à savoir :

- Complexité de configuration initiale : L'initialisation d'HAProxy peut être complexe, nécessitant une compréhension approfondie de sa structure de configuration.
- Limites de performances dans certaines configurations : Dans des scénarios très spécifiques, HAProxy peut rencontrer des limitations en termes de performances.

- Dépendance à un seul point de défaillance : Dans sa configuration de base, HAProxy peut constituer un point de défaillance unique, nécessitant la mise en place de mesures de redondance pour éviter toute interruption du service.
- Besoin de compétences techniques pour la gestion et le dépannage : La gestion et le dépannage d'HAProxy nécessitent des connaissances approfondies en réseaux et une expertise technique.

2.1.5 Mécanismes de redirection dans HAProxy

HAProxy propose plusieurs mécanismes de redirection, dont on cite :

1. **Redirection basée sur l'adresse IP source** : Permet de rediriger les requêtes en fonction de l'adresse IP source du client.
2. **Redirection basée sur le nom d'hôte (DNS)** : Permet de rediriger les requêtes en fonction du nom d'hôte spécifié dans la requête DNS.
3. **Redirection basée sur le chemin d'URL** : Permet de rediriger les requêtes en fonction du chemin d'URL demandé par le client.

Dans notre cas, on s'est basé sur la redirection selon l'adresse **IP source** en entrant **manuellement** les intervalle d'adresse IP pour chaque région.

2.2 Deuxième approche : Application avec Flask

nous avons développé une application structurée en **deux sous-applications** distinctes.

- La première application, appelée **l'application de redirection**, fonctionne sur le port 5000 et est chargée de recevoir les requêtes provenant de différentes régions du pays.

Cette application effectue une comparaison entre l'adresse IP source et les adresses IP stockées dans la base de données importé du site : <https://lite.ip2location.com/algeria-ip-address-ranges>. Si l'utilisateur est connecté depuis l'Algérie, son adresse IP correspond à une entrée existante dans la base de données.

L'application effectue ensuite une **cartographie** entre l'adresse **IP source** et la **localisation géographique correspondante**, en attribuant un **score** initial à chaque serveur.

- La deuxième application est l'application de **monitoring**, qui met à jour **régulièrement** l'état des serveurs du réseau, selon une **période définie** par le configurateur. Elle vérifie la disponibilité et la surcharge des serveurs en envoyant périodiquement des **requêtes SSH** à chaque serveur. Les états sont ensuite **mis à jour en utilisant deux variables partagées**.

2.2.1 Structure de l'application

Chaque application est exécutée sur un **thread séparé**, ce qui leur permet de fonctionner de manière **autonome** et **indépendante** l'une de l'autre.

L'application de redirection redirige les requêtes en fonction de la **géolocalisation de l'adresse IP source**, en utilisant les informations stockées dans la base de données. Elle ajoute également l'état des serveurs (disponible ou non disponible) ainsi que le niveau de surcharge, qui est mis à jour en utilisant des variables partagées.

2.2.2 Communication entre les composants

Les deux sous-applications communiquent entre elles pour garantir un fonctionnement cohérent du CDN. L'application de monitoring met à jour l'état des serveurs pour que l'application de redirection utilise les états pour ajouter le score aux serveurs dont il veut rediriger. De même, l'application de monitoring envoie des **requêtes SSH** régulières à chaque serveur pour vérifier leur disponibilité et leur surcharge, et met à jour les variables partagées en conséquence.

Note sur l'identification géographique : La base de données effectue une cartographie entre l'adresse ip et la localisation géographique si on plot la localisation géographique on aura la figure suivante :



FIG. 2.1 : Distribution de la localisation de l'adresse IP tout au long du pays

Comme il est évident, il n'est pas toujours facile de déterminer les localisations géographiques en fonction de la distance. C'est pourquoi nous avons adopté une approche de division des positions géographiques et avons assigné plusieurs positions à chaque serveur, afin de garantir une répartition équilibrée de la charge. En outre, cette méthode nous permet également de prendre en compte d'autres critères importants.

Chapitre 3

Outils utilisé pour la maintenance et la supervision

3.1 configuration et maintenance de NAS

Nous avons opté pour l'utilisation d'OpenMediaVault en tant que solution pour la gestion du stockage sur le système de stockage en réseau (NAS). OpenMediaVault nous permet de mettre en place une structure RAID (Redundant Array of Independent Disks) pour répartir et sécuriser les données stockées. Cette approche nous offre une meilleure résilience et une meilleure performance en cas de défaillance d'un disque dur.

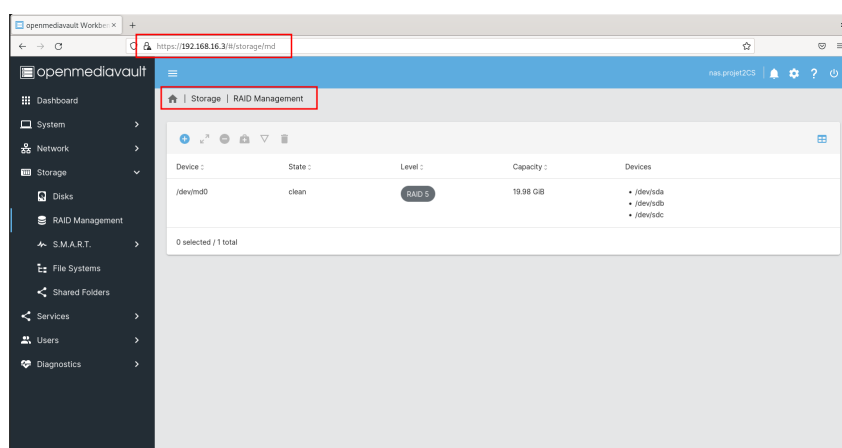


FIG. 3.1 : GUI pour configuration du NAS

3.2 monitoring et supervision des serveurs

Pour fournir une interface graphique qui supervise les serveurs l'équipe a opté d'utiliser le logiciel Check_mk est sa capacité à surveiller en temps réel les différents composants d'un système, tels que les serveurs, les applications, les bases de données et les réseaux.

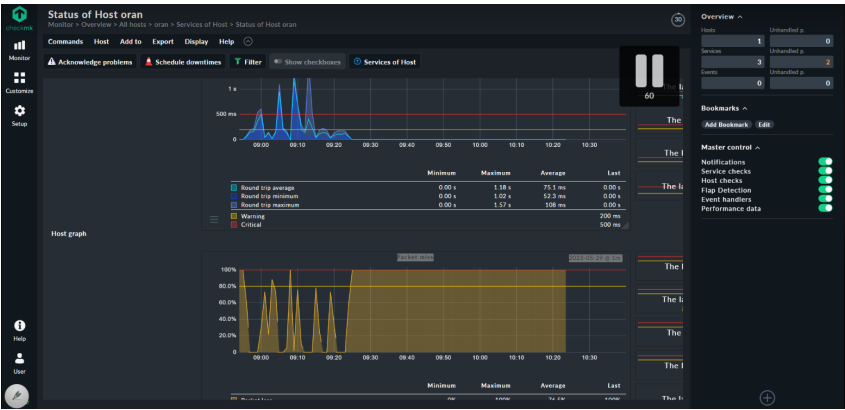


FIG. 3.2 : GUI pour monitoring du serveur

Voici un exemple de vérification de la disponibilité d'un serveur à Oran :

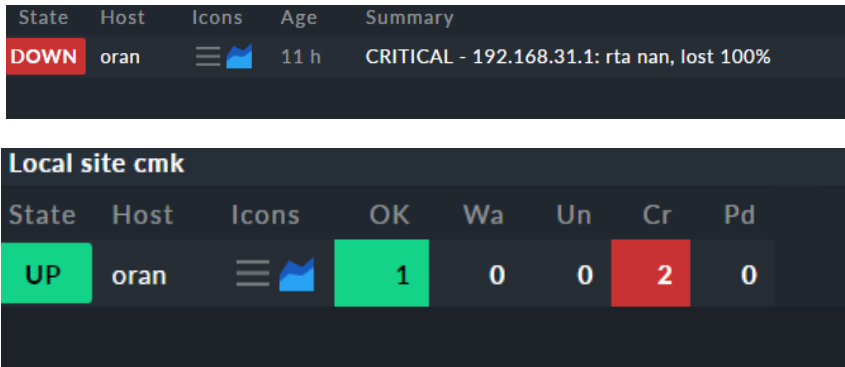


FIG. 3.3 : Exemple de vérification de la disponibilité du serveur à Oran

Chapitre 4

Conception des tests

4.1 Critères de choix des tests

Nous avons élaboré un plan de tests détaillé visant à évaluer l'efficacité de notre architecture mise en place. Ce plan de tests repose sur plusieurs critères essentiels qui sont expliqués ci-dessous, à savoir la **latence**, le **taux de transfert**, la **répartition de charge**, la **cohérence des données** et la **disponibilité**. Ces éléments clés nous permettront d'obtenir une évaluation approfondie de notre architecture CDN, en fournissant des informations précises sur ses performances et sa fiabilité.

La latence est l'un des indicateurs les plus importants lorsqu'il s'agit d'évaluer les performances d'un CDN. Elle mesure le délai de réponse entre une requête émise par un utilisateur et la réception de la réponse correspondante. Un temps de latence élevé peut entraîner une expérience utilisateur médiocre, tandis qu'une latence réduite permet une diffusion rapide du contenu.

Le taux de transfert est un autre paramètre crucial pour mesurer l'efficacité d'un CDN. Il représente la vitesse à laquelle les données sont transférées entre les serveurs du CDN et les utilisateurs finaux. Un taux de transfert élevé garantit une livraison rapide du contenu, ce qui est essentiel pour une expérience utilisateur fluide et agréable.

Une répartition de charge efficace permet d'éviter les engorgements et de maintenir des performances optimales même en cas de demande élevée. Nous évaluerons la capacité de notre architecture à gérer efficacement la répartition de charge afin de garantir une disponibilité continue du contenu.

L'importance de la cohérence survient lorsqu'il s'agit de diffuser du contenu dynamique ou en constante évolution. Nous nous assurerons que notre architecture CDN maintient une cohérence appropriée des données à travers les serveurs, ce qui garantit une expérience utilisateur homogène et exempte d'incohérences.

Enfin, la disponibilité est un aspect clé pour un CDN, c'est peut-être même le plus important, l'architecture doit être opérationnelle en permanence pour répondre aux demandes des utilisateurs. Nous réaliserons des tests pour évaluer la disponibilité de notre architecture et nous assurer que nos serveurs sont robustes et résistants aux pannes.

En combinant ces critères de performance dans notre plan de tests, nous serons en mesure d'évaluer de manière approfondie l'efficacité de notre architecture CDN et de prendre des mesures pour l'optimiser davantage. Les résultats de ces tests nous permettront de prendre des décisions éclairées et de mettre en place les ajustements nécessaires pour offrir la meilleure expérience utilisateur possible.

4.2 Plan de test

Cette section présente en détail la méthodologie de test que nous avons élaborée pour évaluer les performances de notre CDN.

Il convient de souligner que la conception du plan de test se divise en **deux parties majeures**. La **première partie** comprend des tests **dédiés spécifiquement** à chaque solution proposée. La deuxième partie des tests est **commune** et s'applique indépendamment de l'approche choisie. En d'autres termes, les mêmes tests seront effectués sur l'architecture, quelle que soit la solution utilisée.

4.2.1 Partie 1 : Tests unitaires

Haproxy

- Test de résolution de nom de domaine par le serveur DNS
- Test de redirection de HAproxy

Application de redirection

- Test de redirection de l'application

Tests Commun

- Test du cache
- Test du NAS

4.2.2 Partie 1 : Tests unitaires

- Tests intensive sur pour tester la résistance du serveur au requêtes concurrentes
 - 10000 requêtes 100 entre elle sont concurrentes
 - 10000 requêtes 500 entre elle sont concurrentes
 - 10000 requêtes 1000 entre elle sont concurrentes
- Tests intensive pour tester le partitionnement de la redirection de la requêtes
 - 5370 requêtes sachant que tous les serveurs sont en état de marche
 - 5370 requêtes sachant que le serveur de oran est en panne
 - 5370 requêtes sachant que le serveur de oran et constantine est en panne

Chapitre 5

Réalisation des tests et évaluation des résultats

5.1 Plan de tests spécifique à chaque solution

5.1.1 Test de résolution de nom de domaine par le serveur DNS

Dans ce test nous allons tester la résolution de nom de domaine (www.edustream.com) par le serveur DNS dont l'outil BIND est le responsable de cette opération Depuis la machine de client :

```
root@debian:~# nslookup www.edustream.com
Server:      192.168.16.1
Address:     192.168.16.1#53

Name:   www.edustream.com
Address: 192.168.16.1

root@debian:~# █
```

FIG. 5.1 : Résultat résolution de nom par nslookup de la solution HAproxy

Interprétation : Le serveur DNS fournit l'adresse IP associée au nom de domaine www.edustream.com, qui est 192.168.16.1 (adresse IP du serveur DNS).

5.1.2 Test de redirection de HAproxy

Dans ce test nous allons tester la redirection de HAproxy selon la localisation géographique pour cela on va simuler 3 machines clientes de différentes régions du pays qui essayent d'accéder à l'application web

Syntaxe de la commande : curl -I http://www.edustream.com

(l'option -I permet de visualiser les entêtes de la réponse au requête du client et parmi ces en têtes on trouve l'en tête qui indique le cache d'origine)

- **Test n°1**

Accéder à l'application d'après un client de wilaya de Annaba ayant l'adresse ip 192.168.23.10

Interprétation : Le serveur DNS (HAproxy) a redirige la requête de ce client vers le cache de constantine car ce celui ci est le plus proche

```

root@debian:~# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.23.10 netmask 255.255.255.0 broadcast 192.168.23.255
    inet6 fe80::20c:29ff:fe7c:25ff prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:7c:25:ff txqueuelen 1000 (Ethernet)
    RX packets 130901 bytes 28367029 (27.0 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 198380 bytes 16675119 (15.9 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@debian:~# curl -I http://www.edustream.com
HTTP/1.1 200 OK
Server: nginx/1.20.1
Date: Mon, 29 May 2023 22:28:01 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 344
X-Origin-Server-Location: Alger
X-Cache-Location: Constantine
X-Cache-Status: HIT

```

FIG. 5.2 : Résultat du Test n°1 : client situé à Annaba

géographiquement à Annaba

- **Test n°2**

Accéder à l'application d'après un client de wilaya de Sidi Bel Abbes ayant l'adresse ip 192.168.22.10

```

root@debian:~# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.22.10 netmask 255.255.255.0 broadcast 192.168.22.255
    inet6 fe80::20c:29ff:fe7c:25ff prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:7c:25:ff txqueuelen 1000 (Ethernet)
    RX packets 131087 bytes 28383507 (27.0 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 198824 bytes 16713571 (15.9 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@debian:~# curl -I http://www.edustream.com
HTTP/1.1 200 OK
Server: nginx/1.20.1
Date: Mon, 29 May 2023 22:39:23 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 344
X-Origin-Server-Location: Alger
X-Cache-Location: Oran
X-Cache-Status: HIT

```

FIG. 5.3 : Résultat du Test n°2 : client situé à Sidi Bel Abbes

Interprétation : Le serveur DNS (HAproxy) a redirige la requête de ce client vers le cache d'Oran car ce celui ci est le plus proche géographiquement à Sidi Bel Abbes

- **Test n°3**

Accéder à l'application d'après un client de wilaya de Gardaaia ayant l'adresse ip 192.168.47.10

Interprétation : Le serveur DNS (HAproxy) a redirige la requête

```

root@debian:~# curl -I http://www.edustream.com
HTTP/1.1 200 OK
Server: nginx/1.20.1
Date: Tue, 30 May 2023 00:00:15 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 344
X-Origin-Server-Location: Alger
X-Cache-Location: Ouergla
X-Cache-Status: HIT

root@debian:~# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.47.10 netmask 255.255.255.0 broadcast 192.168.47.255
    inet6 fe80::20c:29ff:fe7c:25ff prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:7c:25:ff txqueuelen 1000 (Ethernet)
    RX packets 131151 bytes 28388795 (27.0 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 199120 bytes 16737114 (15.9 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

FIG. 5.4 : Résultat du Test n°3 : client situé à Gardaia

de ce client vers le cache d'Ouargla car ce celui ci est le plus proche géographiquement à Ghardaïa

- **Test n°4**

Accéder à l'application d'après un client de wilaya de Annaba ayant l'adresse ip 192.168.22.10 (Test 1) mais cette fois ci on éteint le cache de constantine

```

root@debian:~# curl -I http://www.edustream.com
HTTP/1.1 200 OK
Server: nginx/1.18.0
Date: Mon, 29 May 2023 22:15:31 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 344
X-Origin-Server-Location: Alger

```

FIG. 5.5 : Résultat du Test n°4 : Disponibilité et de la charge des serveurs

Interprétation : L'en-tête "X-Cache-Location : Constantine" n'est pas présent actuellement en raison de l'indisponibilité du serveur cache de Constantine. Dans cette situation, le serveur DNS (HA-proxy) redirige la requête vers l'un des autres serveurs disponibles (caches ou serveur d'origine) en fonction de leur charge. Dans ce cas, il a redirigé la requête vers le serveur d'origine d'Alger.

5.1.3 Test de redirection de l'application

Dans cette série de captures d'écran, nous allons démontrer la fonctionnalité de surveillance et de répartition de charge des serveurs dans notre application. Cela est mis en œuvre grâce au script de configuration des serveurs que nous avons développé.

1. Script de la fonctionnalité de la disponibilité des serveurs et leur charges

```
def check_servers():
    servers_status = []

    # Iterate through the list of servers
    for server_url in servers_urls:
        try:
            # Send a GET request to the server
            response = requests.get(server_url, timeout=1)

            # Check if the response indicates a successful connection (e.g., 200 status code)
            if response.status_code == 200:
                servers_status.append(1)
            else:
                servers_status.append(0)

        except requests.exceptions.RequestException as e:
            # If an exception occurs, consider the server as down
            servers_status.append(0)

    # Update the server status in your Flask application as needed
    # For example, you can store it in a database, update a global variable, etc.

    return(servers_status) # Print server status for demonstration purposes
```

```
def check_load_on_servers():
    scores = []
    global availability
    i = 0
    for ip_address in servers_ip:
        if availability[i] == 0:
            scores.append(0)
        else:
            try:
                conn = socket.create_connection((ip_address, 22), timeout=1)
                with conn:
                    mem_usage = psutil.virtual_memory()
                    cpu_percent = psutil.cpu_percent()
                    # print(f"CPU usage: {cpu_percent}%")
                    # print(f"Memory usage: {mem_usage.percent}%")
                    scores.append(1 - (0.5 * float(cpu_percent) / 100.0 + 0.5 * float(mem_usage.percent) / 100.0))
            except ConnectionRefusedError:
                print(f"Connection to {ip_address} refused")
                scores.append(0)
            except TimeoutError:
                # print(f"Connection to {ip_address} timed out")
                scores.append(0)
            except Exception as e:
                # print(f"Error connecting to {ip_address}: {str(e)}")
                scores.append(0)
        i=i+1
    return scores
```

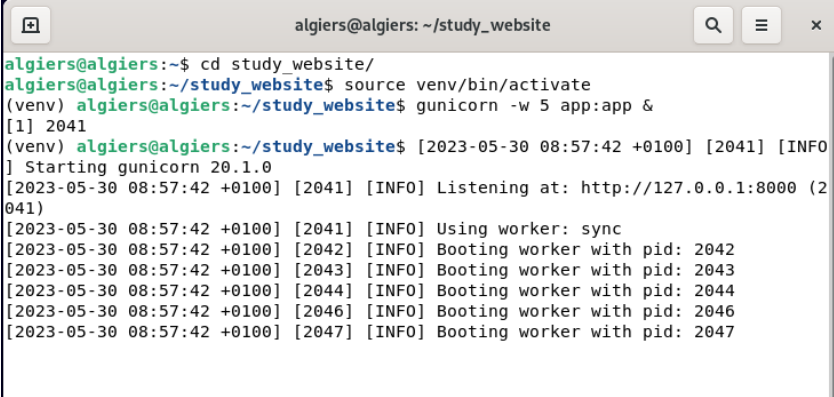
FIG. 5.6 : Script de configuration (Parties 1 et 2)

2. Une fois que l'application est lancée, un test de disponibilité est effectué pour chaque serveur :

```
dns-lb@dns-loadbalancer:~/lb_redirect_app$ python3 redirector.py
checking availability ..
* Serving Flask app 'redirector'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on http://192.168.16.1:5000
Press CTRL+C to quit
* Restarting with stat
checking availability ..
checking loads ..
availability update : [0, 0, 0, 0]
* Debugger is active!
* Debugger PIN: 114-460-294
checking loads ..
availability update : [0, 0, 0, 0]
```

FIG. 5.7 : Lancement de l'application de redirection

3. Lancement de l'application serveur sur le serveur d'origine :

A terminal window titled 'algiers@algiers: ~/study_website'. The user enters the following commands: 'cd study_website/', 'source venv/bin/activate', and 'gunicorn -w 5 app:app &'. The output shows gunicorn starting with 5 workers. Log messages include: '[2023-05-30 08:57:42 +0100] [2041] [INFO] Starting gunicorn 20.1.0', '[2023-05-30 08:57:42 +0100] [2041] [INFO] Listening at: http://127.0.0.1:8000 (2041)', '[2023-05-30 08:57:42 +0100] [2041] [INFO] Using worker: sync', and five subsequent messages for booting workers with pids 2042, 2043, 2044, 2046, and 2047.

```
algiers@algiers:~$ cd study_website/
algiers@algiers:~/study_website$ source venv/bin/activate
(venv) algiers@algiers:~/study_website$ gunicorn -w 5 app:app &
[1] 2041
(venv) algiers@algiers:~/study_website$ [2023-05-30 08:57:42 +0100] [2041] [INFO]
Starting gunicorn 20.1.0
[2023-05-30 08:57:42 +0100] [2041] [INFO] Listening at: http://127.0.0.1:8000 (2
041)
[2023-05-30 08:57:42 +0100] [2041] [INFO] Using worker: sync
[2023-05-30 08:57:42 +0100] [2042] [INFO] Booting worker with pid: 2042
[2023-05-30 08:57:42 +0100] [2043] [INFO] Booting worker with pid: 2043
[2023-05-30 08:57:42 +0100] [2044] [INFO] Booting worker with pid: 2044
[2023-05-30 08:57:42 +0100] [2046] [INFO] Booting worker with pid: 2046
[2023-05-30 08:57:42 +0100] [2047] [INFO] Booting worker with pid: 2047
```

FIG. 5.8 : Fonctionnalité de la charges des serveurs

4. État du serveur load balancer DNS

```
checking availability ..
checking loads ..
availability update : [1, 0, 0, 0]
■
```

FIG. 5.9 : État du serveur load balancer DNS (lui-même étant l'origine)

5. Après l'allumage de tous les serveurs :

```
checking loads ..
availability update : [1, 1, 1, 1]
```

FIG. 5.10 : Disponibilité de tous les serveurs

Lancement de l'application de redirection

1. Adresse IP source (client) pour le test :

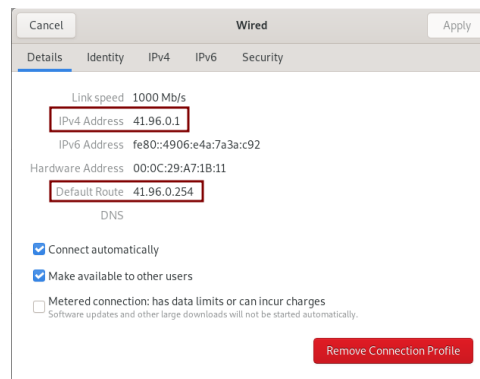


FIG. 5.11 : Adresse IP source (client)

2. Faire une requête vers le serveur de redirection :

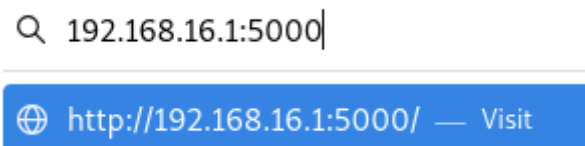


FIG. 5.12 : Requête vers le serveur de redirection (cas 1)

3. Résultat : redirection vers le serveur cache d'Oran

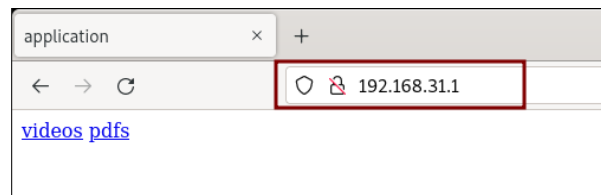


FIG. 5.13 : Redirection vers le serveur d'Oran

4. Visualisation du score

```
availability : [1, 1, 1, 1]
loads : [0.921, 0.7865, 0.4295, 0.858]
loads : [0.921, 0.7865, 0.4295, 0.858]
loads : [0.921, 0.7865, 0.4295, 0.858]
loads : [0.921, 0.7865, 0.4295, 0.858]
Final scores [0.42840000000000006, 0.7345999999999999, 0.2318, 0.4032]
41.96.0.1 - - [30/May/2023 15:57:46] "GET / HTTP/1.1" 302 -
```

FIG. 5.14 : Visualisation du score attribuée par l'application (cas 1)

5. Simulation du cas ou le serveur d'Oran est en panne :

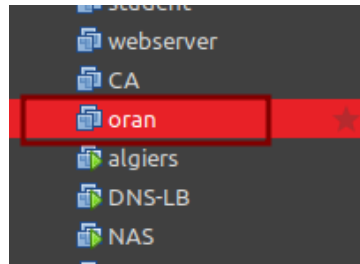


FIG. 5.15 : Serveur d'Oran en panne (désactivé)

6. Refaire la requête vers le serveur de redirection

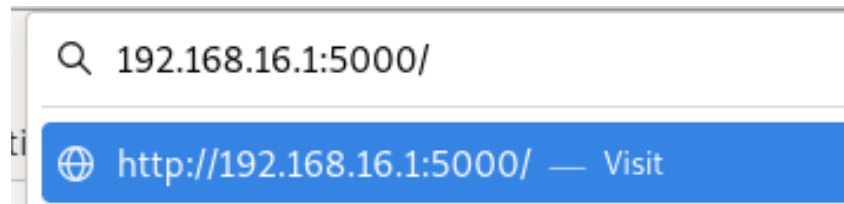


FIG. 5.16 : Requête vers le serveur de redirection (cas 2)

7. Resultat : redirection vers le serveur cache de Constantine



FIG. 5.17 : Redirection vers le serveur de Constantine

8. Visualisation des Scores

```
availability : [1, 0, 1, 1]
loads : [0.919, 0, 0.9295, 0.6515]
loads : [0.919, 0, 0.9295, 0.6515]
loads : [0.919, 0, 0.9295, 0.6515]
loads : [0.919, 0, 0.9295, 0.6515]
Final scores [0.42760000000000004, 0.0, 0.4318, 0.3206]
41.96.0.1 - - [30/May/2023 16:09:39] "GET / HTTP/1.1" 302 -
```

FIG. 5.18 : Visualisation du score attribuée par l'application (cas 2)

9. Réactivation du serveur d'Oran et répétition de la requête vers le serveur de redirection :

```
availability update : [1, 1, 1, 1]
loads : [0.927, 0.8460000000000001, 0.8295, 0.9295]
availability : [1, 1, 1, 1]
loads : [0.927, 0.8460000000000001, 0.8295, 0.9295]
loads : [0.927, 0.8460000000000001, 0.8295, 0.9295]
loads : [0.927, 0.8460000000000001, 0.8295, 0.9295]
loads : [0.927, 0.8460000000000001, 0.8295, 0.9295]
Final scores [0.4308, 0.7584, 0.3918000000000004, 0.4318]
```

FIG. 5.19 : Visualisation des nouvelles scores attribuées par l'application

5.1.4 Tests sur le NAS (Network Attach Storage)

1. Montage du Serveur NAS sur le serveur d'origine

```
algiers@algiers:~$ sudo mount -t nfs 192.168.16.3:/ /mnt/nas
[sudo] password for algiers:
```

FIG. 5.20 : Serveur NAS monter sur serveur origine

2. Contenu du serveur NAS destiné à notre application

```
algiers@algiers:~$ cd /mnt/nas/NFS01/
algiers@algiers:/mnt/nas/NFS01$ ls
pdfs  videos
```

FIG. 5.21 : Affichage du contenu du Serveur NAS

3. Implémentation du mécanisme de la sauvegarde au niveau de l'application (serveur origine)

```
@app.route('/videos/<video_id>')
def get_video(video_id):
    video = Video.query.get_or_404(video_id)
    video_file = os.path.join("/mnt/nas/NFS01/videos", video.video_path)
    return send_file(video_file)

@app.route('/pdfs/<pdf_id>')
def get_pdf(pdf_id):
    pdf = Pdf.query.get_or_404(pdf_id)
    pdf_file = os.path.join("/mnt/nas/NFS01/pdfs", pdf.pdf_path)
    return send_file(pdf_file)

def save_video(form video):
    random_hex = secrets.token_hex(8)
    file_ext=os.path.splitext(form video.filename)
    video_file_name = random_hex +file_ext
    video_path = os.path.join("/mnt/nas/NFS01/videos",video_file_name)
    form_video.save(video_path)
    return video_file_name

@app.route('/video/add', methods=['GET', 'POST'])
@login_required
def add_video():
    form=VideoForm()
    if form.validate_on_submit():
        video_file = save_video(form.video.data)
        video= Video(teacher=current_user,video_title=form.video_title.data,video_path=video_file)
        db.session.add(video)
        db.session.commit()
        return redirect(url_for('teacher_space'))
    return render_template('add_video.html', form=form)

def save_pdf(form document):
    random_hex = secrets.token_hex(8)
    file_ext=os.path.splitext(form document.filename)
    document_file_name = random_hex +file_ext
    picture_path = os.path.join("/mnt/nas/NFS01/pdfs",document_file_name)
    form_document.save(picture_path)
    return document_file_name
```

FIG. 5.22 : Script responsable de la sauvegarde du contenu dans le NAS

4. Visualisation du contenu du serveur NAS avant de réaliser le test

```
algiers@algiers:/mnt/nas/NFS01$ cd pdfs
algiers@algiers:/mnt/nas/NFS01/pdfs$ ls
4425070e4dd734e3.pdf  5ddef18640603494.pdf  70c2e8129029598e.pdf  928391dcae2f5a5.pdf  a1688517f593d7ca.pdf  ec288e83f79c843d.pdf
```

FIG. 5.23 : Contenu du serveur NAS (PDFs) avant de réaliser le test

5. Ajout d'un fichier PDF

login

192.168.31.1/login

Login admin

Username: admin

Password: [masked]

Login

application

192.168.31.1/pdf/add

pdf title: cour_demo

choose pdf from device: Browse...

cours_reseau.pdf

upload pdf

FIG. 5.24 : Interface de connexion et d'importation d'un fichier PDF

6. Visualisation du contenu du serveur NAS après avoir réalisé le test

```
algiers@algiers:/mnt/nas/NFS01/pdfs$ ls
4425070e4dd734e3.pdf  5ddef18640603494.pdf  70c2e8129029598e.pdf  928391dcae2f5a5.pdf  a1688517f593d7ca.pdf  ec288e83f79c843d.pdf
algiers@algiers:/mnt/nas/NFS01/pdfs$ ls
4425070e4dd734e3.pdf  5ddef18640603494.pdf  70c2e8129029598e.pdf  90f768d30e394fed.pdf  928391dcae2f5a5.pdf  a1688517f593d7ca.pdf  ec288e83f79c843d.pdf
algiers@algiers:/mnt/nas/NFS01/pdfs$
```

FIG. 5.25 : Contenu du serveur NAS (PDFs) après avoir réalisé le test

7. Visualisation des trames d'écriture au niveau du NAS

510	345.286485	192.168.16.2	192.168.16.3	NFS	374 V4 Call (Reply In 513) WRITE StateID: 0x5
511	345.286529	192.168.16.3	192.168.16.2	TCP	66 2049 → 749 [ACK] Seq=7869 Ack=410689 Win=
512	345.286622	192.168.16.3	192.168.16.2	TCP	66 2049 → 749 [ACK] Seq=7869 Ack=416789 Win=
513	345.286921	192.168.16.3	192.168.16.2	NFS	246 V4 Reply (Call In 510) WRITE
514	345.287256	192.168.16.2	192.168.16.3	NFS	298 V4 Call (Reply In 515) COMMIT FH: 0x57a4d
515	345.313959	192.168.16.3	192.168.16.2	NFS	174 V4 Reply (Call In 514) COMMIT
516	345.314402	192.168.16.2	192.168.16.3	NFS	322 V4 Call (Reply In 517) CLOSE StateID: 0x5
517	345.314685	192.168.16.3	192.168.16.2	NFS	254 V4 Reply (Call In 516) CLOSE
518	345.357495	192.168.16.2	192.168.16.3	TCP	66 749 → 2049 [ACK] Seq=417277 Ack=8345 Win=

FIG. 5.26 : Les trames qui sont transmises au NAS en écriture

8. Effectuer la lecture (ou demande) d'un fichier PDF

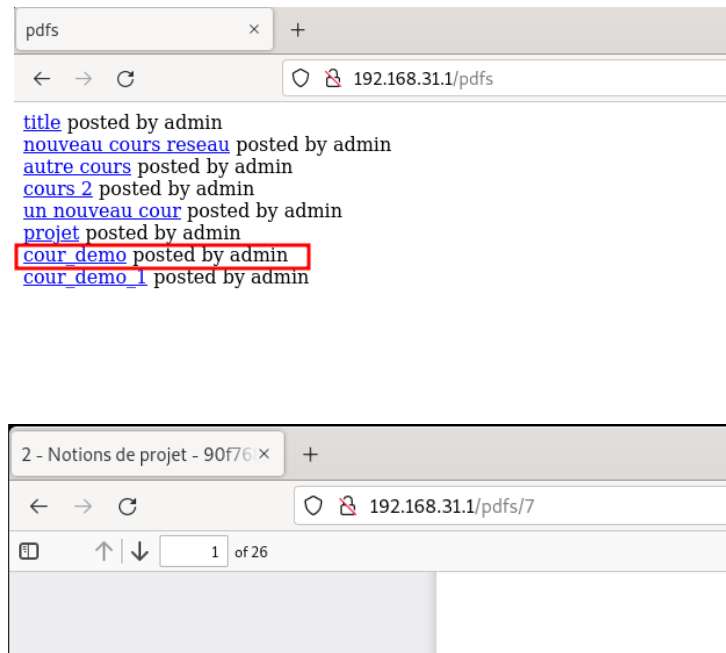


FIG. 5.27 : lecture d'un fichier PDF

9. Visualisation des trames de lecture au niveau du NAS

915	488.711932	192.168.16.2	192.168.16.3	TCP	66 749 → 2049 [ACK] Seq=419953 Ack=413761 Wi
916	488.711991	192.168.16.3	192.168.16.2	TCP	1514 2049 → 749 [ACK] Seq=416657 Ack=419953 Wi
917	488.712025	192.168.16.3	192.168.16.2	NFS	346 V4 Reply (Call In 896) READ
918	488.712110	192.168.16.2	192.168.16.3	TCP	66 749 → 2049 [ACK] Seq=419953 Ack=416657 Wi
919	488.712185	192.168.16.2	192.168.16.3	TCP	66 749 → 2049 [ACK] Seq=419953 Ack=418385 Wi
920	488.712510	192.168.16.2	192.168.16.3	NFS	306 V4 Call (Reply In 921) CLOSE StateID: 0x3
921	488.712939	192.168.16.3	192.168.16.2	NFS	182 V4 Reply (Call In 920) CLOSE
922	488.756219	192.168.16.2	192.168.16.3	TCP	66 749 → 2049 [ACK] Seq=420193 Ack=418501 Wi

FIG. 5.28 : Les trames qui sont transmises au NAS en lecture

5.2 Tests Intensifs

5.2.1 Test de résistance du serveur de redirection

1. commande : `ab -n 10000 -c 100 -s 90 http://edustream.com/`

```
Concurrency Level:      100
Time taken for tests:    88.312 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      5870001 bytes
HTML transferred:      3440000 bytes
Requests per second:    113.23 [#/sec] (mean)
Time per request:       883.123 [ms] (mean)
Time per request:       8.831 [ms] (mean, across all concurrent requests)
Transfer rate:          64.91 [Kbytes/sec] received
```

```
Concurrency Level:      100
Time taken for tests:    39.642 seconds
Complete requests:      10000
Failed requests:        0
Non-2xx responses:      10000
Total transferred:      4410000 bytes
HTML transferred:      2310000 bytes
Requests per second:    252.26 [#/sec] (mean)
Time per request:       396.415 [ms] (mean)
Time per request:       3.964 [ms] (mean, across all concurrent requests)
Transfer rate:          108.64 [Kbytes/sec] received
```

FIG. 5.29 : Résultats des deux solutions pour 10000 requêtes avec 100 en concurrence

2. commande : `ab -n 10000 -c 500 -s 90 http://edustream.com/`

```
Concurrency Level:      500
Time taken for tests:    95.271 seconds
Complete requests:      10000
Failed requests:        20
    (Connect: 0, Receive: 0, Length: 20, Exceptions: 0)
Non-2xx responses:      20
Total transferred:      5862500 bytes
HTML transferred:      3435320 bytes
Requests per second:    104.96 [#/sec] (mean)
Time per request:       4763.555 [ms] (mean)
Time per request:       9.527 [ms] (mean, across all concurrent requests)
Transfer rate:          60.09 [Kbytes/sec] received
```

```
Concurrency Level:      500
Time taken for tests:    61.249 seconds
Complete requests:      10000
Failed requests:        2534
    (Connect: 0, Receive: 0, Length: 2534, Exceptions: 0)
Non-2xx responses:      10000
Total transferred:      4425204 bytes
HTML transferred:      2320136 bytes
Requests per second:    163.27 [#/sec] (mean)
Time per request:       3062.460 [ms] (mean)
Time per request:       6.125 [ms] (mean, across all concurrent requests)
Transfer rate:          70.56 [Kbytes/sec] received
```

FIG. 5.30 : Résultats des deux solutions pour 10000 requêtes avec 500 en concurrence

3. commande : commande : ab -n 10000 -c 1000 -s 90 http ://edustream.com/

```
Concurency Level:      1000
Time taken for tests:  94.480 seconds
Complete requests:     10000
Failed requests:       77
  (Connect: 0, Receive: 0, Length: 77, Exceptions: 0)
Non-2xx responses:     77
Total transferred:     5841071 bytes
HTML transferred:      3421928 bytes
Requests per second:   105.84 [#/sec] (mean)
Time per request:      9448.039 [ms] (mean)
Time per request:      9.448 [ms] (mean, across all concurrent requests)
Transfer rate:         60.37 [Kbytes/sec] received
```

```
Concurency Level:      1000
Time taken for tests:  49.118 seconds
Complete requests:     10000
Failed requests:       3537
  (Connect: 0, Receive: 0, Length: 3537, Exceptions: 0)
Non-2xx responses:     10000
Total transferred:     4431222 bytes
HTML transferred:      2324148 bytes
Requests per second:   203.59 [#/sec] (mean)
Time per request:      4911.814 [ms] (mean)
Time per request:      4.912 [ms] (mean, across all concurrent requests)
Transfer rate:         88.10 [Kbytes/sec] received
```

FIG. 5.31 : Résultats des deux solutions pour 10000 requêtes avec 1000 en concurrence

5.2.2 Comparaison des temps de réponse de chaque solution

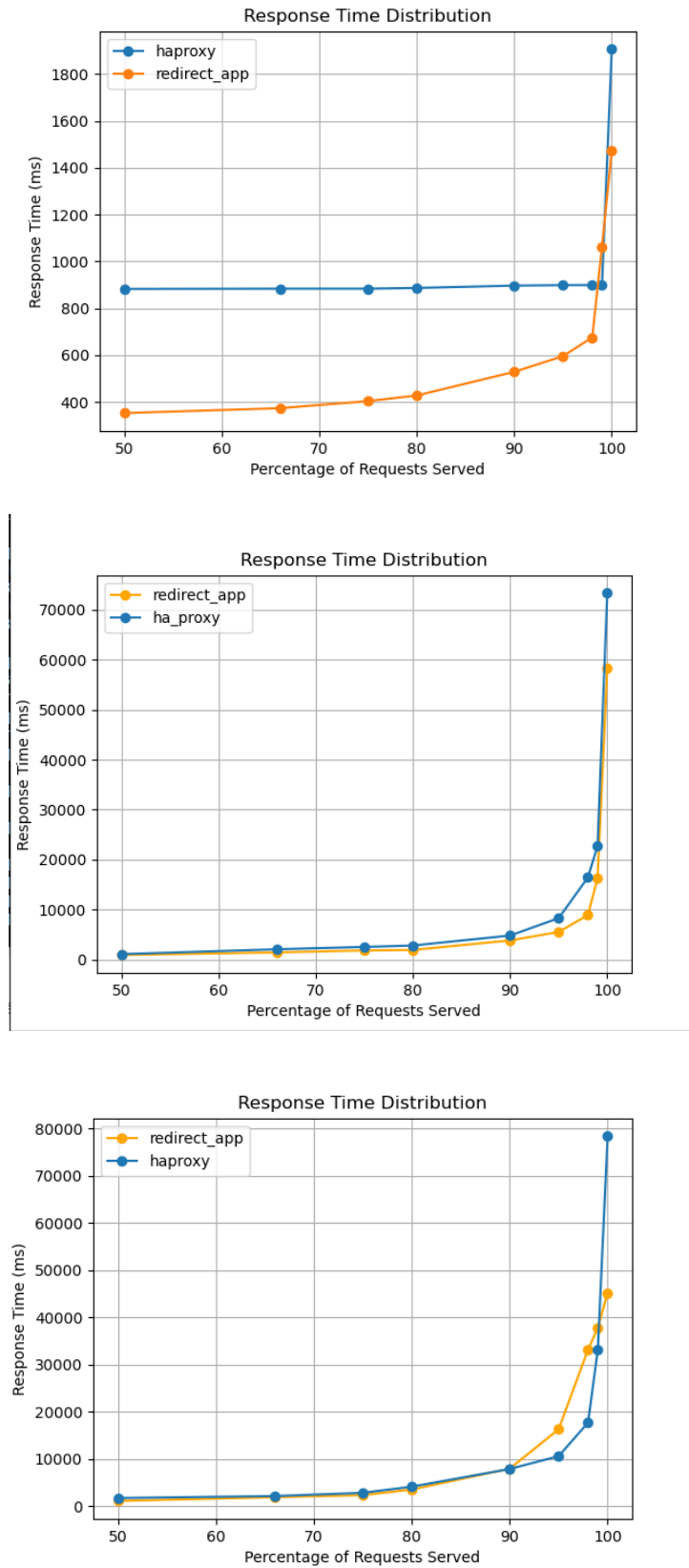


FIG. 5.32 : Courbes des temps de réponse des deux solutions (10 000 requêtes)

Interprétation et Synthèse

D'après ce test, on constate que en variant à chaque fois le degré de concurrence (c'est-à-dire le nombre de requêtes envoyées simultanément), le temps nécessaire pour effectuer le test, le nombre moyen de requêtes traitées par seconde, le débit de transfert et le temps moyen de traitement des requêtes concurrentes ne sont pas affectés de manière significative. Bien que le nombre de requêtes échouées augmente légèrement à mesure que le degré de concurrence augmente, le pourcentage reste toujours inférieur à 2%. La seule différence entre ces tests réside dans le temps moyen de traitement d'une requête, car certaines requêtes prennent plus de temps à être traitées que d'autres.

On remarque également que l'application HAproxy présente des temps de réponse supérieurs à ceux de l'application de redirection. En revanche, l'application de redirection privilégie la suppression des requêtes pour garantir des temps de réponse plus rapides.

5.2.3 Test sur la répartition des requêtes selon leur adresses

En utilisant la bibliothèque Request pour effectuer les tests de répartition de charges entre les serveurs

1. Génération d'adresses IP aléatoires à partir des plages d'adresses IP existantes dans la base de données associée au rôle.

```
193.194.64.1 193.194.95.255 819
193.41.146.1 193.41.147.255 51
195.24.80.1 195.24.87.255 204
195.39.218.1 195.39.219.255 51
196.20.64.1 196.20.127.255 163
196.29.40.1 196.29.43.255 102
196.41.224.1 196.41.255.255 80
197.112.0.1 197.119.255.255 520
197.140.0.1 197.143.255.255 2620
197.200.0.1 197.207.255.255 520
198.145.148.1 198.145.149.255 51
213.140.56.1 213.140.57.255 51
213.140.59.1 213.140.59.255 20
```

FIG. 5.33 : Adresses IP aléatoires

2. Effectuer 5000 requêtes vers le site redirigeur pour certains scénarios spécifiques :

(a) Application de redirection : Tous les serveur marchent.

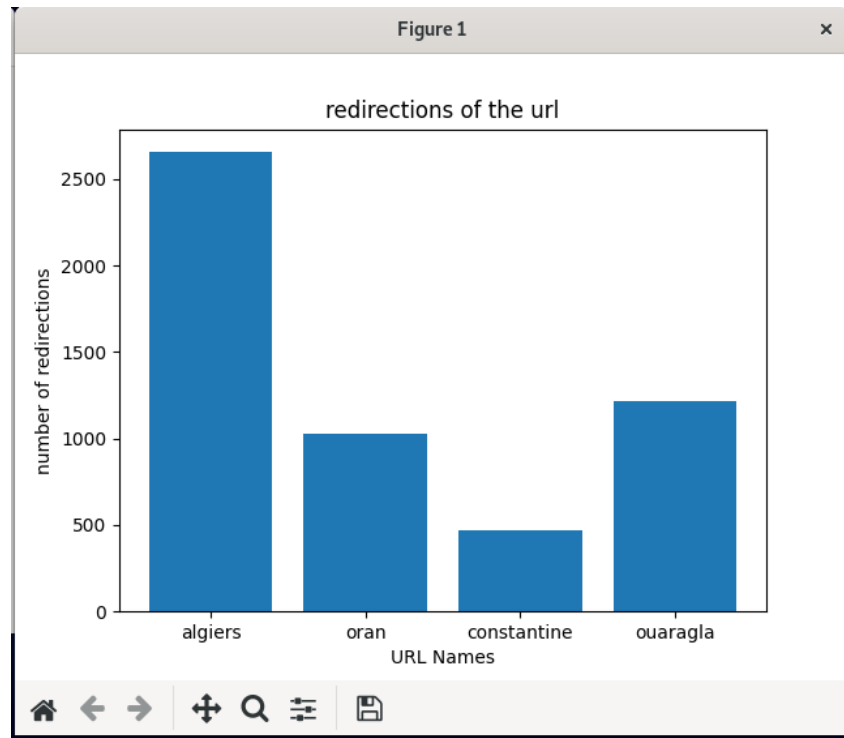


FIG. 5.34 : Redirection des clients vers des serveurs caches différents (cas a)

(b) HAProxy : Tous les serveur marchent.

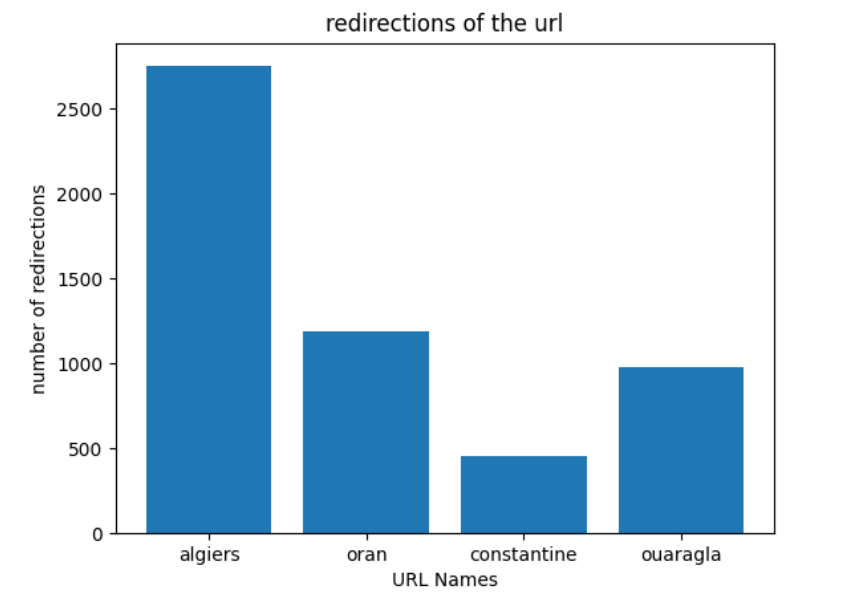


FIG. 5.35 : Redirection des clients vers des serveurs caches différents (cas b)

(c) Application de redirection : Site d'Oran en panne.

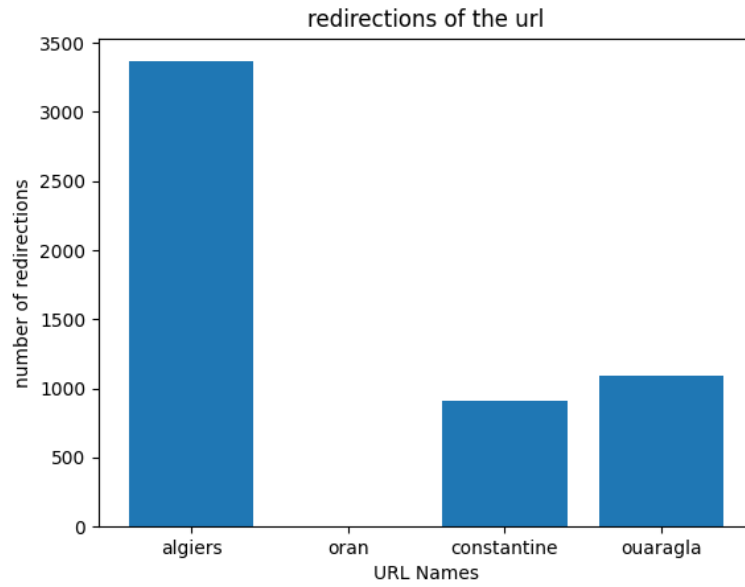


FIG. 5.36 : Redirection des clients vers des serveurs caches différents (cas c)

(d) HAProxy : Site d'Oran en panne.

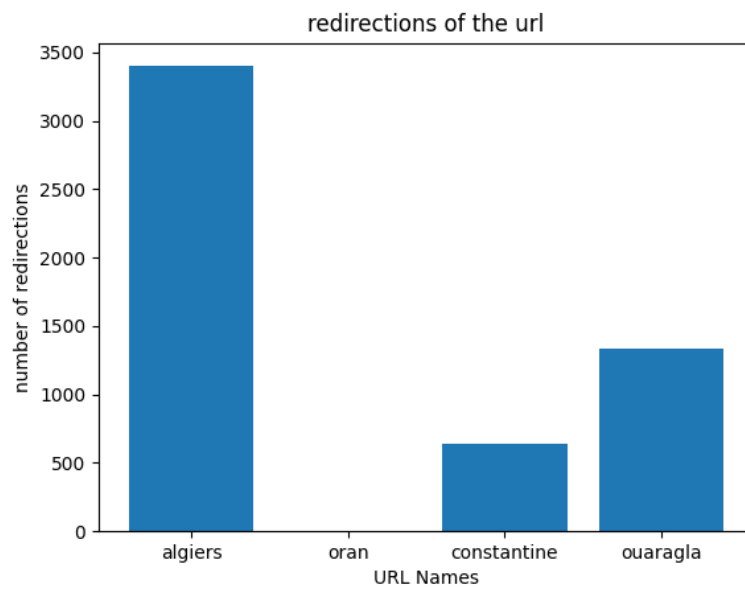


FIG. 5.37 : Redirection des clients vers des serveurs caches différents (cas d)

(e) Application de redirection : site d'Oran et Constantine en panne.

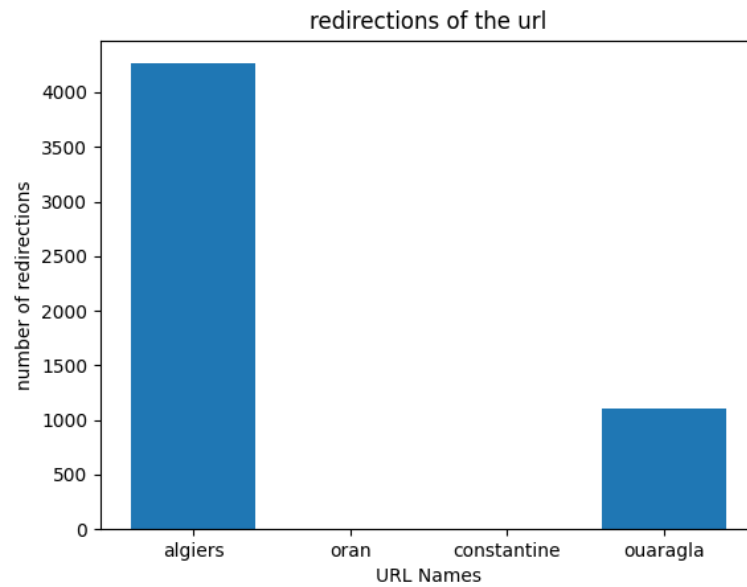


FIG. 5.38 : Redirection des clients vers des serveurs caches différents (cas e)

(f) HAProxy : site d'Oran et Constantine en panne

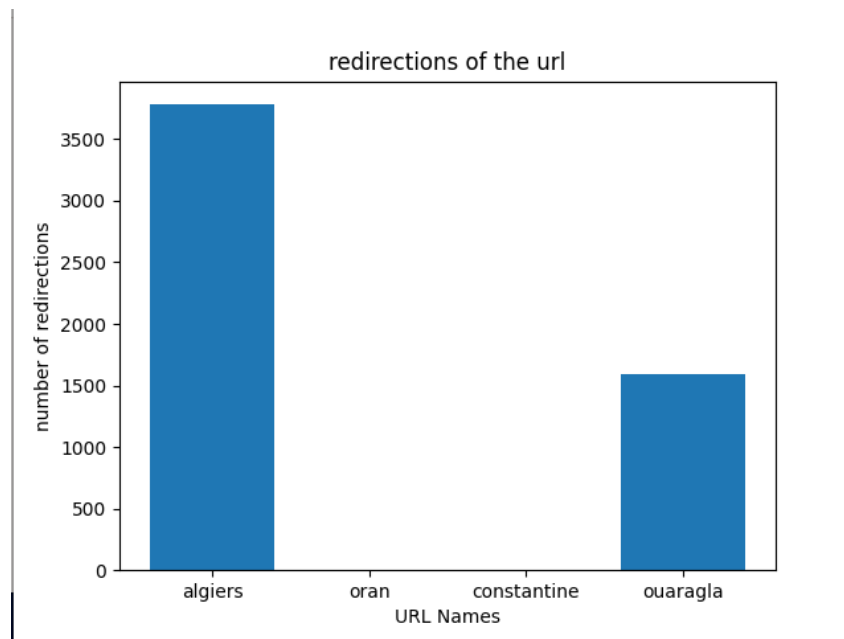


FIG. 5.39 : Redirection des clients vers des serveurs caches différents (cas f)

Interprétation et Synthèse

L'application de redirection se distingue par sa flexibilité et son efficacité dans la gestion des redirections, même avec un nombre élevé de requêtes. En plus de sa capacité à mettre à jour dynamiquement les paramètres de connexion, elle permet d'ajuster les coefficients et les priorités des serveurs en fonction du nombre de serveurs disponibles.

Ainsi, même en cas de défaillance d'un serveur, la répartition des requêtes s'ajuste automatiquement pour garantir une redirection efficace. Cette flexibilité de l'application de redirection lui confère un avantage en termes d'adaptabilité et de maintien de performances élevées. Elle peut gérer de manière dynamique les ressources disponibles et ajuster les priorités des serveurs en fonction de la charge et de l'état des serveurs individuels. Ces fonctionnalités offrent une solution robuste pour gérer efficacement les requêtes dans des environnements où les conditions peuvent varier rapidement.

L'application de redirection se révèle ainsi être un choix stratégique pour garantir une distribution équilibrée des requêtes, même lorsque les conditions opérationnelles changent ou en cas de défaillance d'un serveur. Elle offre une flexibilité et une adaptabilité indispensables pour maintenir des performances optimales et assurer la disponibilité continue du service.

Chapitre 6

Résumé du travail effectué dans le projet

- Partie 1
 1. Etude théorique de la notion du CDN
 2. Proposition d'une infrastructure initiale pour le projet, matériel utilisé et estimation du coût.
- Partie 2
 1. Confirmation de l'architecture du projet
 2. Choix des technologies et approches de la solution (HAProxy)
 3. Programmation du site web prototype (back-end sans liaison avec le NAS (stockage locale))
 4. Configuration du réseau simulateur de CERIST en utilisant MPLS au niveau du GNS3
 5. Configuration initiale de l'application de redirection (un bug qui ralentit l'application)
 6. Configuration initiale de HAProxy (en utilisant round-robin)
- Partie 3
 1. Finalisation de l'application avec le stockage et placement dans la topologie
 2. Fixage du bug de l'application de redirection
 3. Finalisation de la configuration des adresses pour la redirection de HAProxy
 4. Réalisation des tests unitaires et tests intensive

Conclusion

Ce projet a été une expérience enrichissante à bien des égards. Tout au long de sa réalisation, nous avons fait face à des défis, mais ceux-ci nous ont permis d'apprendre et de grandir. Sur le plan théorique, nous avons acquis une compréhension approfondie de l'utilité d'un réseau CDN et des principes sous-jacents à sa mise en œuvre. Nous avons exploré les différentes technologies utilisées pour créer et simuler un réseau CDN, ce qui nous a permis de mieux appréhender les enjeux et les solutions possibles.

Cependant, la véritable valeur de cette expérience réside dans notre apprentissage pratique. Nous avons eu l'opportunité d'installer et de configurer divers éléments clés d'un réseau CDN, tels qu'un reverse proxy, un serveur web, un serveur cache, un serveur NAS et un serveur de redirection. Ces activités nous ont donné une perspective concrète sur la façon dont ces composants interagissent et contribuent à la performance globale du réseau.

De plus, nous avons pu travailler avec des outils de simulation avancés tels que GNS3 et VMware. Cette expérience nous a permis de reproduire des environnements complexes et de tester différentes configurations, ce qui a renforcé notre compréhension des interactions entre les différents éléments du réseau.

En résumé, ce projet nous a offert l'opportunité de développer à la fois nos connaissances théoriques et nos compétences pratiques. Nous avons acquis une compréhension approfondie de la conception et de la mise en œuvre d'un réseau CDN, ainsi que de l'utilisation des outils de simulation. Ces compétences seront certainement précieuses dans notre parcours professionnel futur, car les réseaux CDN jouent un rôle de plus en plus crucial dans la diffusion efficace des contenus sur Internet.

En conclusion, nous sommes fiers des résultats que nous avons obtenus grâce à ce projet. Il a été une occasion exceptionnelle de développer nos compétences techniques et de nous familiariser avec des technologies clés du domaine des réseaux. Nous sommes confiants que les connaissances acquises et les expériences vécues tout au long de ce projet nous seront bénéfiques à l'avenir.