Abel Vagne
William Antivackis
Dylan Fournier

UQAC
Université du Québec
à Chicoutimi

Travaux Pratiques 2

Conception/architecture des systèmes infonuagique

Février 2025

Table des matières

1 Introduction **3**

2 Définition **3**

3 Serveurs Node.js **3**

4 Configuration de HAProxy **4**

5 Statistiques HAProxy **5**

6 Simulation d'envoie de requêtes **5**

7 Docker **6**

8 Exécution **6**
8.1 Exécution à partir du dépôt GitHub 6
8.2 Exécution à partir de DockerHub 7
8.3 Utilisation de l'application 7
8.4 Exécution du script python 8

Annexes **9**

1 Introduction

La question 2 du TP2 vise à mettre en place une infrastructure de répartition de charge entre différents serveurs Node.js, à l'aide de HAProxy.
Les objectifs du TP sont :

1. Installer Node.js et npm pour créer deux serveurs web simples.
2. Lancer ces serveurs sur des ports distincts.
3. Installer HAProxy et le configurer pour répartir les requêtes entrantes entre ces serveurs.
4. Tester le bon fonctionnement du load balancing.
5. Observer les performances via l'interface de statistiques de HAProxy.

Le code source de cette question est disponible sur le dépôt **GitHub**.

Le projet est aussi disponible sur **DockerHub** via ce lien.

2 Définition

Le **load balancing** permet d'optimiser l'utilisation des ressources, de maximiser le débit, de réduire le temps de réponse, et d'éviter qu'un seul serveur ne soit surchargé, ce qui dégraderait les performances globales.

HAProxy (High Availability Proxy) est un logiciel open source, largement utilisé pour la répartition de charge et la mise en place de serveurs proxy TCP ou HTTP. Il est conçu pour gérer des sites à très fort trafic, et est utilisé dans de nombreuses infrastructures critiques.

3 Serveurs Node.js

Pour le projet, deux serveurs web ont été développés à l'aide de Node.js. Chacun d'eux écoute sur un port distinct : le premier sur le port `3000`, le second sur le port `3001`.

L'objectif de ces serveurs est de pouvoir calculer le hash d'une chaîne de caractères. Pour cela, on a mis en place une interface graphique (1) depuis laquelle, un utilisateur peut saisir une chaîne de caractères et choisir un algorithme de hashages cryptographiques parmi *MD5*, *SHA-256* et *SHA-512*.

FIG. 1 – *Frontend de l'application Web*

4 Configuration de HAProxy

Dans cette partie, nous discutons de la configuration de HaProxy, disponible en **Annexe 8.4**.

Le fichier configuration est divisé en différentes parties :

- Section `Global` et `Defaults` : configuration des paramètres globaux et des valeurs par défaut.
- Section `Frontend` : configure le point d'entrée des requêtes.
- Section `Backend` : contient la liste des serveurs vers lesquels HAProxy répartit les requêtes.
- Section `Listen stats` : gère la visualisation des statistiques de HaProxy.

Dans notre projet, nous avons utilisé l'algortihme **RoundRobin** pour la répartition des charges. Mais d'autres algorithmes de répartition peuvent être utilisés :

- **Roundrobin** : distribue les requêtes de façon cyclique entre les serveurs, en les envoyant tour à tour à chaque serveur.
- **Weighted Roundrobin** : variante du roundrobin où chaque serveur reçoit un nombre de requêtes proportionnel à son poids (capacité), permettant de privilégier les serveurs plus puissants.
- **Least connection** : envoie chaque nouvelle requête au serveur ayant actuellement le moins de connexions actives, idéal pour équilibrer dynamiquement la charge.
- **Weighted least connection** : combine le principe du Least connection avec un système de poids, permettant de privilégier les serveurs les plus puissants tout en tenant compte de leur charge actuelle.

5 Statistiques HAProxy

HAProxy propose une interface de monitoring très pratique pour observer en temps réel l'état des serveurs backend et le comportement de la répartition de charge. Cette interface est accessible via l'URL `http://localhost:8404/stats`.

L'interface fournit un tableau de bord détaillé en indiquant le nom des serveurs, leur statut (UP/DOWN), le nombre de sessions actives, le nombre total de requêtes traitées et bien d'autres.

Le tableau est mise à jour automatiquement ce qui permet de vérifier la bonne répartition du trafic entre les serveurs, de détecter si un serveur cesse de fonctionner.

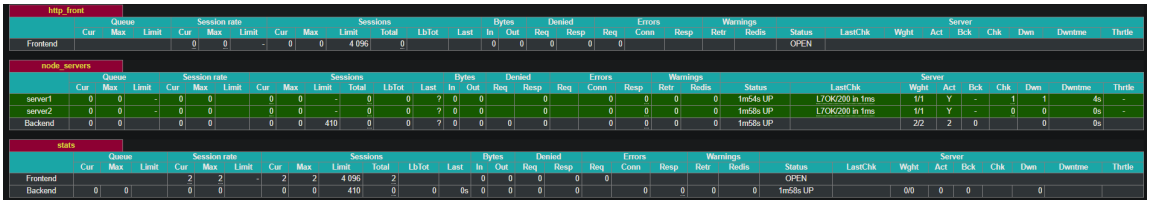

FIG. 2 – *Tableau Statistiques fourni par HAProxy*

6 Simulation d'envoie de requêtes

Nous avons également développé des programmes permettant de simuler l'envoi de requêtes.

Envoi progressif de requêtes via Python

Un script Python a été développé pour simuler l'envoie progressive la charge sur les serveurs, en envoyant les requêtes par vagues successives.

Mise en place d'un dashboard

Un tableau de bord a également été intégré à l'application pour lancer manuellement une simulation de requêtes.

Ce tableau de bord est disponible à l'adresse : `http://localhost/dashboard`.

Contrairement au script Python, qui envoie les requêtes de manière progressive, le dashboard utilise JavaScript pour lancer les requêtes en parallèle, ce qui permet de simuler une attaque plus brutale et instantanée. Cette distinction permet de comparer les effets de deux types de charges : l'une croissante, l'autre immédiate.

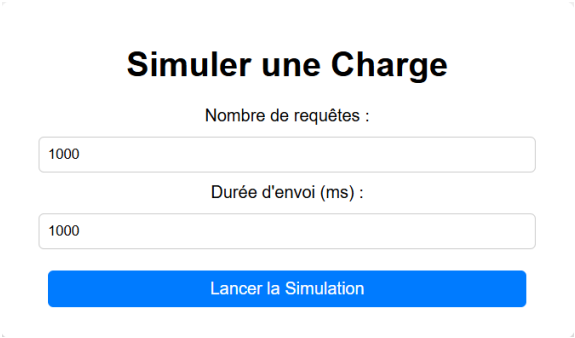

FIG. 3 – *Vue HTML /dashboard*

7 Docker

L'ensemble du projer a été conteneurisée à l'aide de Docker, ce qui permet de simplifier le déploiement, les tests ainsi que la portabilité de l'environnement sur différentes machines.

L'image Docker du projet est d'ailleurs disponible sur Docker Hub.

8 Exécution

8.1 Exécution à partir du dépôt GitHub

Après avoir *git clone* le projet. On obtient les exécutions suivantes :

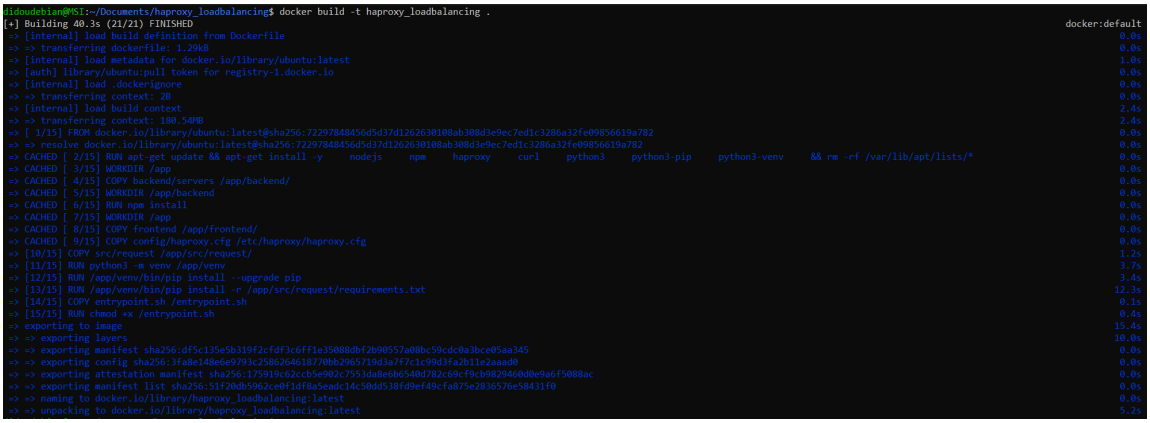

FIG. 4 – *docker build -t haproxy_loadbalancing .*

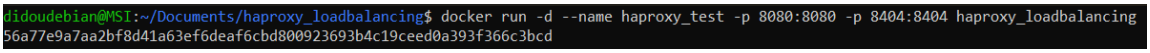

FIG. 5 – *docker run -d –name haproxy_test -p 8080:8080 -p 8404:8404 haproxy_loadbalancing*

8.2 Exécution à partir de DockerHub

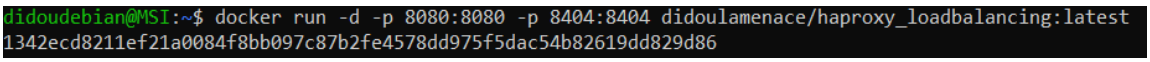

FIG. 6 – *docker run -d -p 8080:8080 -p 8404:8404 didoulamenace/haproxy_loadbalancing:latest*

8.3 Utilisation de l'application

Après avoir mis en route l'application avec Docker en local ou DockerHub, on peut se rendre aux URL suivantes :

- `http://localhost:8080` : Accès à l'application pour la génération de hash.
- `http://localhost:8080/dashboard` : interface dédiée à la simulation de requêtes.
- `http://localhost:8404` : interface de statistiques de HAProxy.

8.4 Exécution du script python

```
didoudebian@MSI:~/Documents/haproxy_loadbalancing$ docker exec -it haproxy_test bash
root@0e75450c3167:/app# source /app/venv/bin/activate
(venv) root@0e75450c3167:/app# python /app/src/request/request_progressive.py

 Envoi d'un lot de 100 requêtes...
Latence :
    Min : 2 ms | Max : 16 ms | Moyenne : 3 ms
Total cumulé : 100 | Échecs cumulés : 0

 Envoi d'un lot de 200 requêtes...
Latence :
    Min : 2 ms | Max : 4 ms | Moyenne : 2 ms
Total cumulé : 200 | Échecs cumulés : 0

 Envoi d'un lot de 300 requêtes...
Latence :
    Min : 2 ms | Max : 5 ms | Moyenne : 2 ms
Total cumulé : 300 | Échecs cumulés : 0

 Envoi d'un lot de 400 requêtes...
Latence :
    Min : 2 ms | Max : 4 ms | Moyenne : 2 ms
Total cumulé : 400 | Échecs cumulés : 0

 Envoi d'un lot de 500 requêtes...
Latence :
    Min : 2 ms | Max : 11 ms | Moyenne : 2 ms
Total cumulé : 500 | Échecs cumulés : 0

 Envoi d'un lot de 600 requêtes...
Latence :
    Min : 2 ms | Max : 14 ms | Moyenne : 2 ms
Total cumulé : 600 | Échecs cumulés : 0
```

FIG. 7 – *python /app/src/request/request_progressive.py*

Annexes

Annexe A : Contenu du fichier `haproxy.cfg`

Voici la configuration complète utilisée pour HAProxy dans le cadre du projet :

```
# Configuration Globale
global
    maxconn 4096 # Dfinition du nombre maximal de connexions simultanes  4096
    daemon # Excution de HAProxy en arrire-plan (mode daemon)

# Configuration par dfaut
defaults
    log global # Utilisation des logs globaux
    mode http # HAProxy fonctionne en mode HTTP (au lieu de TCP)
    timeout connect 5000ms # Timeout pour tablir une connexion avec un backend (5 sec)
    timeout client 50000ms # Timeout pour attendre les donnes du client (50 sec)
    timeout server 50000ms # Timeout pour attendre la rponse dun serveur backend (50 sec)

# Configuration Frontend
frontend http_front
    bind *:8080 # HAProxy coute sur le port 8080 (toutes les interfaces rseau)
    default_backend node_servers # Par dfaut, toutes les requtes sont envoyes vers "
        node_servers"

    # Dfinition d'une ACL (Access Control List) pour les requtes API
    acl is_api path_beg /hash /stats # Vrifie si l'URL commence par /hash ou /stats
    use_backend node_servers if is_api # Si la requte correspond  l'ACL, elle est envoye au
         backend node_servers

# COnfiguration Backend
backend node_servers
    balance roundrobin # Mthode d'quilibrage : Round Robin, Weighted Round Robin, Least
        Connections et Weighted Least Connections
    option httpchk GET /stats # Vrification de l'tat des serveurs en envoyant une requte
        GET sur /stats
    server server1 127.0.0.1:3000 check # Ajout du serveur "server1" sur le port 3000 avec
        une vrification d'tat
    server server2 127.0.0.1:3001 check # Ajout du serveur "server2" sur le port 3001 avec
        une vrification d'tat

# Interface Statistiques HAProxy
listen stats
    bind *:8404 # HAProxy coute sur le port 8404 pour l'interface des statistiques
    stats enable # Activation de l'interface des statistiques
    stats uri /stats # Accs aux statistiques via l'URL http://localhost:8404/stats
    stats refresh 10s # Rafrachissement des statistiques toutes les 10 secondes
```