

Abel Vagne
William Antivackis
Dylan Fournier



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'envoi 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'algorithme **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.

http_stats		Queue		Session rate			Sessions			Bytes		Denied		Errors		Warnings		Status	LastChk	Weight	Server				Downtime	Throttle	
Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LibTot	Last	In	Out	Req	Resp	Conn	Resp	Rate	Redis		Act	Back	Chk	Dwn			
Frontend				0	0	-	0	0	0	4 096	0	0	0	0	0	0	0	0	0					OPEN			
node_servers		Queue		Session rate			Sessions			Bytes		Denied		Errors		Warnings		Status	LastChk	Weight	Server				Downtime	Throttle	
Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LibTot	Last	In	Out	Req	Resp	Conn	Resp	Rate	Redis		Act	Back	Chk	Dwn			
server1	0	0	-	0	0	0	0	-	0	0	0	0	0	0	0	0	0	0	0	1m50s UP	1/1	Y	-	1	1	4s	-
server2	0	0	-	0	0	0	0	-	0	0	0	0	0	0	0	0	0	0	0	1m50s UP	1/1	Y	-	0	0	0s	-
Backend	0	0	0	0	0	0	0	410	0	0	0	0	0	0	0	0	0	0	0	1m50s UP	2/2	2	0	0	0	0s	-
stats		Queue		Session rate			Sessions			Bytes		Denied		Errors		Warnings		Status	LastChk	Weight	Server				Downtime	Throttle	
Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LibTot	Last	In	Out	Req	Resp	Conn	Resp	Rate	Redis		Act	Back	Chk	Dwn			
Frontend				2	2	-	2	2	4 096	2	0	0	0	0	0	0	0	0	0	OPEN							
Backend				0	0	0	0	0	410	0	0	0	0	0	0	0	0	0	0	1m50s UP	0/0	0	0	0	0	0	

FIG. 2 – Tableau Statistiques fourni par HAProxy

6 Simulation d'envoi 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'envoi 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.

Simuler une Charge

Nombre de requêtes :

Durée d'envoi (ms) :

Lancer la Simulation

FIG. 3 – Vue HTML /dashboard

7 Docker

L'ensemble du projet 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 :

```
h@hadooplan90:~/Documents/haproxy_loadbalancing$ docker build -t haproxy_loadbalancing .
(*) Building 40.3s (21/21) FINISHED
-> [internal] load build definition from Dockerfile
-> => transferring dockerfile: 1.29kB
-> [internal] load metadata for docker.io/library/ubuntu:latest
-> [auth] library/ubuntu:pull token for registry-1.docker.io
-> [internal] load .dockerignore
-> => transferring context: 2B
-> [internal] load build context
-> => transferring context: 180.54kB
-> [ 1/15] FROM docker.io/library/ubuntu:latest@sha256:72297848456d5d37d1262630108ab308d3e9ec7ed1c3286a32fe09856619a782
-> => resolve docker.io/library/ubuntu:latest@sha256:72297848456d5d37d1262630108ab308d3e9ec7ed1c3286a32fe09856619a782
-> CACHED [ 2/15] RUN apt-get update && apt-get install -y nodejs npm haproxy curl python3 python3-pip python3-venv && rm -rf /var/lib/apt/lists/*
-> CACHED [ 3/15] WORKDIR /app
-> CACHED [ 4/15] COPY backend/servers /app/backend/
-> CACHED [ 5/15] WORKDIR /app/backend
-> CACHED [ 6/15] RUN npm install
-> CACHED [ 7/15] WORKDIR /app
-> CACHED [ 8/15] COPY frontend /app/frontend/
-> CACHED [ 9/15] COPY config/haproxy.cfg /etc/haproxy/haproxy.cfg
-> [10/15] COPY src/request /app/src/request/
-> [11/15] RUN python3 -m venv /app/venv
-> [12/15] RUN /app/venv/bin/pip install --upgrade pip
-> [13/15] RUN /app/venv/bin/pip install -r /app/src/request/requirements.txt
-> [14/15] COPY entrypoint.sh /entrypoint.sh
-> [15/15] RUN chmod +x /entrypoint.sh
-> exporting to image
-> => exporting layers
-> => exporting manifest sha256:d5c135e9b319f2cf4f3c6ff1a35080d8f2b09557a08bc59cd6a3bc0d9aa345
-> => exporting config sha256:3fab01480e9793c2586264618770bb2965719d3a7f7c1c99d3fa2b11c2aaad0
-> => exporting attestation manifest sha256:179919c62cc05e902c7553dabeb6540d782c69cf9cb982946d0d8e9a6f508bac
-> => exporting manifest list sha256:51f20d05962ce0f1d78a5eadc14c50dd538fd9ef45cf875e2836576e58431f0
-> => naming to docker.io/library/haproxy_loadbalancing:latest
-> => unpacking to docker.io/library/haproxy_loadbalancing:latest
```

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

```
didoudebien@MSI:~/Documents/haproxy_loadbalancing$ docker run -d --name haproxy_test -p 8080:8080 -p 8404:8404 haproxy_loadbalancing
56a77e9a7aa2bf8d41a63ef6deaf6cbd800923693b4c19ceed0a393f366c3bcd
```

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

8.2 Exécution à partir de DockerHub

```
didoudebien@MSI:~$ docker run -d -p 8080:8080 -p 8404:8404 didoulamenace/haproxy_loadbalancing:latest
1342ecd8211ef21a0084f8bb097c87b2fe4578dd975f5dac54b82619dd829d86
```

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

```
didoudebien@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 :

```
1 # Configuration Globale
2 global
3     maxconn 4096 # Dfinition du nombre maximal de connexions simultanes 4096
4     daemon # Exccution de HAProxy en arriere-plan (mode daemon)
5
6 # Configuration par dfaut
7 defaults
8     log global # Utilisation des logs globaux
9     mode http # HAProxy fonctionne en mode HTTP (au lieu de TCP)
10    timeout connect 5000ms # Timeout pour tablir une connexion avec un backend (5 sec)
11    timeout client 50000ms # Timeout pour attendre les donnes du client (50 sec)
12    timeout server 50000ms # Timeout pour attendre la rponse dun serveur backend (50 sec)
13
14 # Configuration Frontend
15 frontend http_frontend
16     bind *:8080 # HAProxy coute sur le port 8080 (toutes les interfaces rseau)
17     default_backend node_servers # Par dfaut, toutes les requetes sont envoyes vers "
18         node_servers"
19
20     # Dfinition d'une ACL (Access Control List) pour les requetes API
21     acl is_api path_beg /hash /stats # Vrifie si l'URL commence par /hash ou /stats
22     use_backend node_servers if is_api # Si la requete correspond l'ACL, elle est envoye au
23         backend node_servers
24
25 # COnfiguration Backend
26 backend node_servers
27     balance roundrobin # Mthode d'quilibrage : Round Robin, Weighted Round Robin, Least
28         Connections et Weighted Least Connections
29     option httpchk GET /stats # Vrification de l'tat des serveurs en envoyant une requete
30         GET sur /stats
31     server server1 127.0.0.1:3000 check # Ajout du serveur "server1" sur le port 3000 avec
32         une vrification d'tat
33     server server2 127.0.0.1:3001 check # Ajout du serveur "server2" sur le port 3001 avec
34         une vrification d'tat
35
36 # Interface Statistiques HAProxy
37 listen stats
38     bind *:8404 # HAProxy coute sur le port 8404 pour l'interface des statistiques
39     stats enable # Activation de l'interface des statistiques
40     stats uri /stats # Accs aux statistiques via l'URL http://localhost:8404/stats
41     stats refresh 10s # Rafrachissement des statistiques toutes les 10 secondes
```