

10 DECEMBRE 2018

LABO 3
LOAD BALANCING

DEJVID MUAREMI, ROMAIN SILVESTRI
HEIG-VD
TS – 3ème

1. Table des matières

1.	TABLE DES MATIÈRES	1
2.	INTRODUCTION	2
2.1.	REMARQUE	2
3.	TÂCHE 1 : INSTALLER LES OUTILS	3
3.1.	VAGRANT	3
3.2.	JMETER	3
3.3.	GIT	3
3.4.	MANIPULATIONS	3
3.5.	RÉPONSES AUX QUESTIONS	4
4.	TÂCHE 2 : STICKY SESSIONS	7
4.1.	RÉPONSES AUX QUESTIONS	7
5.	TÂCHE 3 : DRAIN MODE	11
5.1.	RÉPONSES AUX QUESTIONS	11
6.	TÂCHE 4 : ROUND ROBIN IN DEGRADED MODE	13
6.1.	RÉPONSES AUX QUESTIONS	14
7.	TÂCHE 5 : BALANCING STRATEGIES	17
7.1.	RÉPONSES AUX QUESTIONS	17
8.	CONCLUSION	20
9.	TABLE DES ILLUSTRATIONS	21
10.	TABLE DES RÉFÉRENCES	22

2. Introduction

Pour ce laboratoire, nous allons déployer une application web dans une architecture à deux tiers afin de tester notre scalabilité. Le travail sera réalisé en 5 étapes, dans chacune d'entre-elles nous allons regarder différents points de la configuration d'un proxy, ici HAProxy.

2.1. Remarque

Comme indiqué dans le README.md, nous allons travailler sur la version 1.5 de HAProxy, par conséquent, ce rapport ne tiendra pas compte des autres versions, et il faudrait le réadapter si vous voulez travailler avec la version 1.8, la dernière sortie, ou 1.9, toujours en développement, à ce jour.

3. Tâche 1 : Installer les outils

Pour cette première étape, nous allons installer les outils de base nécessaire pour la réalisation de ce laboratoire.

3.1. Vagrant

Multiplateforme et écrit en Ruby, il s'agit d'un logiciel libre et open source, conçu pour créer et configurer des environnements de développement virtuel.

Depuis la version 1.6, Vagrant fourni également un support natif des conteneurs Docker, cela permet de réduire le coût en ressource.

Pour l'installer, il suffit d'aller sur le site officiel et de prendre la dernière version disponible, aucune option particulière n'a été utilisée et la documentation officielle suffit amplement.

3.2. JMeter

Développé par Apache Software Foundation, Apache JMeter est un logiciel libre écrit en Java, par conséquent multiplateforme, qui permet d'effectuer des tests de performance d'applications et de serveurs selon différents protocoles ainsi que d'effectuer des tests fonctionnels.

Le point intéressant de ce logiciel est qu'il permet de simuler un accès multiple sur une application web ce qui sera très utile pour tester notre load balancer.

3.3. Git

Encore un logiciel libre et open source, git est particulièrement utile pour cloner et travailler sur le repo, il simplifie également le travail en équipe. La documentation officielle suffit pour l'installer et l'utiliser, aucune option particulière n'a été mise en place mis à part l'utilisation des clé SSH post-installation.

3.4. Manipulations

Pour cette première partie, il nous est demandé de lancer le Vagrant et d'essayer d'utiliser l'application web pour voir comment celle-ci réagit avant de répondre aux questions suivantes.

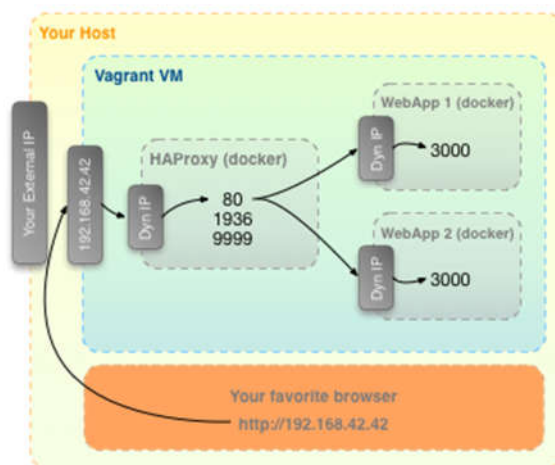


FIGURE 1: ARCHITECTURE

```
{
  "hello": "world!",
  "ip": "172.17.0.7",
  "host": "2b277f0fe8da",
  "tag": "s1",
  "sessionViews": 1,
  "id": "pdoSpuStaotz04us2l_uYArG0w6S57eV"
}
```

FIGURE 2 : CONTENU DE LA RÉPONSE

L'adresse de l'application web est la suivante : <http://192.168.42.42> ou, dans certains cas particuliers, <http://localhost:8080>.

3.5. Réponses aux questions

3.5.1. Question 1

En lançant l'application, on se rend compte que à chaque fois que l'on rafraîchit la page, on change de serveur. On se rend compte qu'il y a un problème étant donné que le serveur nous donne un cookie mais celui-ci n'a pas l'air d'être utilisé pour garder notre session et l'on reçoit à chaque fois une nouvelle comme on peut le voir dans la prochaine capture.

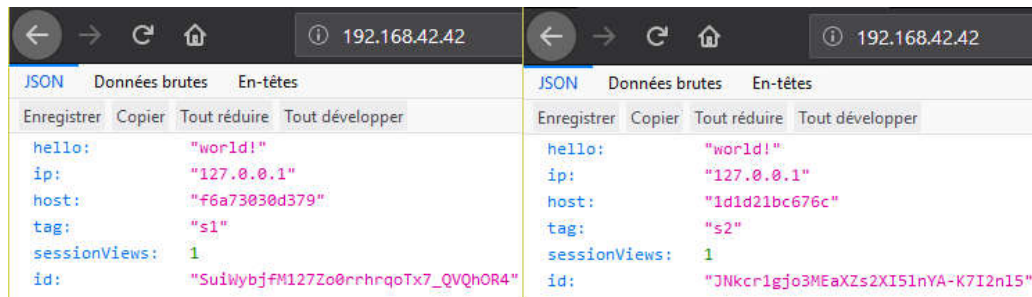


FIGURE 3 : RECHARGEMENT D'UNE PAGE

3.5.2. Question 2

Normalement, si le load balancer fonctionne correctement, on devrait garder la même session et pour se faire, il y a deux possibilités.

1. La session est gardée par le même serveur aussi longtemps que possible, et donc, l'utilisateur ne change pas de serveur même avec des rafraîchissements de page.
2. Le load balancer transmet la session d'un serveur à l'autre, ainsi l'utilisateur garde sa session même lorsqu'il y a un changement de serveur.

3.5.3. Question 3

Lors d'une première connexion suivie d'un rafraîchissement, on peut observer le comportement suivant :

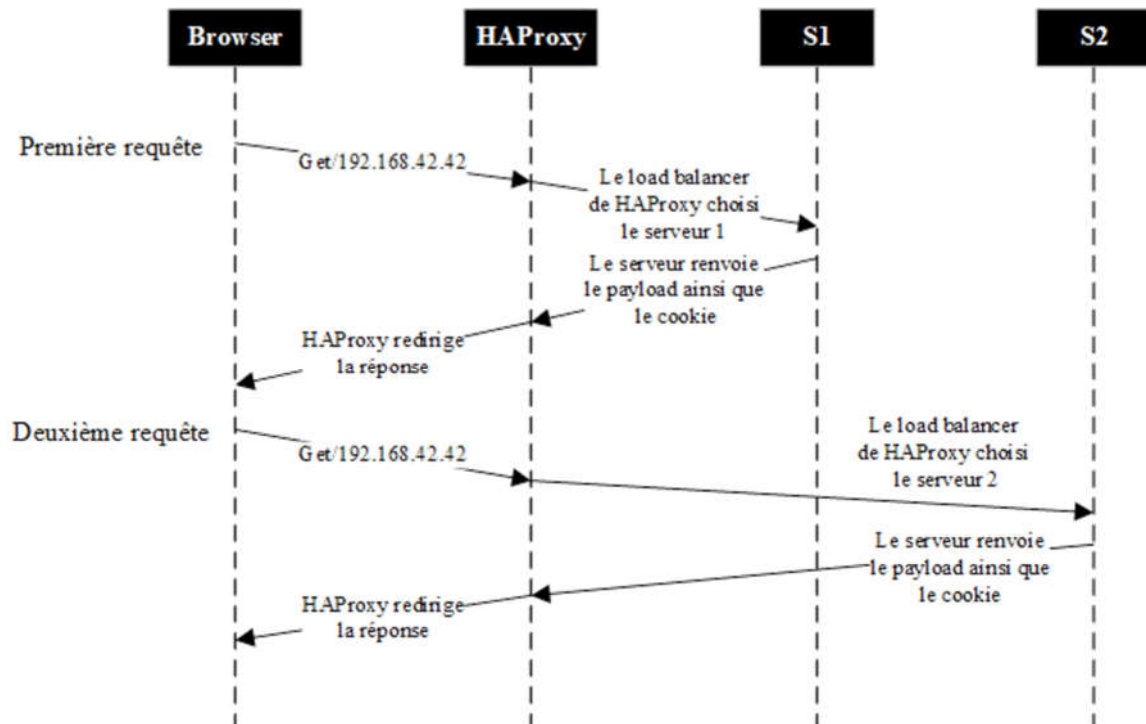


FIGURE 4 : DIAGRAMME DE SÉQUENCE, RECHARGEMENT DE PAGE

3.5.4. Question 4

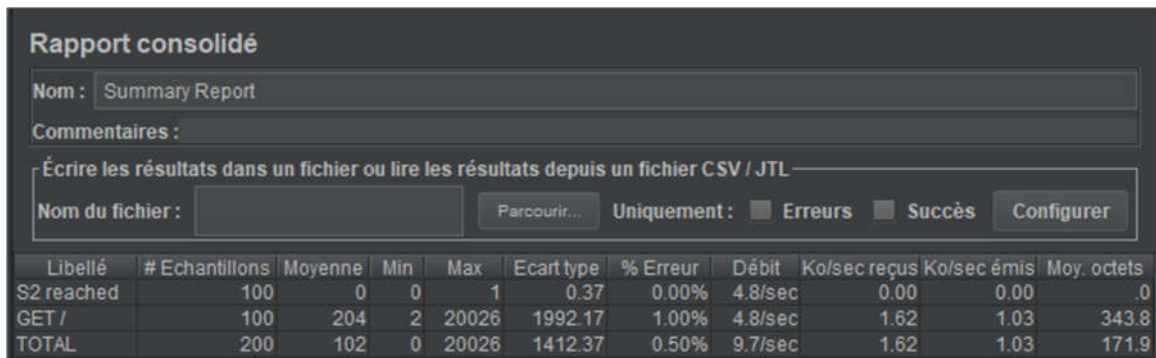
Comme on peut le voir sur l'image suivante, d'après le rapport JMeter, le load balancer fait en sorte que les requêtes soient distribuées équitablement entre les deux serveurs.

Rapport consolidé											
Nom : Summary Report											
Commentaires :											
Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL											
Nom du fichier : <input type="text"/> Parcourir... Uniquement : <input type="checkbox"/> Erreurs <input checked="" type="checkbox"/> Succès Configurer											
Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets	
GET /	100	14	3	194	24.33	0.00%	40.2/sec	18.40	8.62	468.7	
S1 reached	50	0	0	1	0.46	0.00%	22.6/sec	0.00	0.00	.0	
S2 reached	50	0	0	1	0.48	0.00%	25.4/sec	0.00	0.00	.0	
TOTAL	200	7	0	194	18.67	0.00%	80.3/sec	18.37	8.61	234.3	

FIGURE 5 : JMETER, COMPORTEMENT PAR DÉFAUT

3.5.5. Question 5

En tuant un serveur, le load balancer prend un petit moment, environ 20 secondes, pour se rendre compte qu'il n'a plus qu'un serveur à qui envoyé les requêtes.



Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : ☐ Erreurs ☒ Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
S2 reached	100	0	0	1	0.37	0.00%	4.8/sec	0.00	0.00	.0
GET /	100	204	2	20026	1992.17	1.00%	4.8/sec	1.62	1.03	343.8
TOTAL	200	102	0	20026	1412.37	0.50%	9.7/sec	1.62	1.03	171.9

FIGURE 6 : JMETER, AVEC UN SERVEUR DOWN

Comme on ne change plus de serveur, on garde la même session active comme on peut le voir sur le navigateur, le serveur reste le même et le compteur de vue augmente.

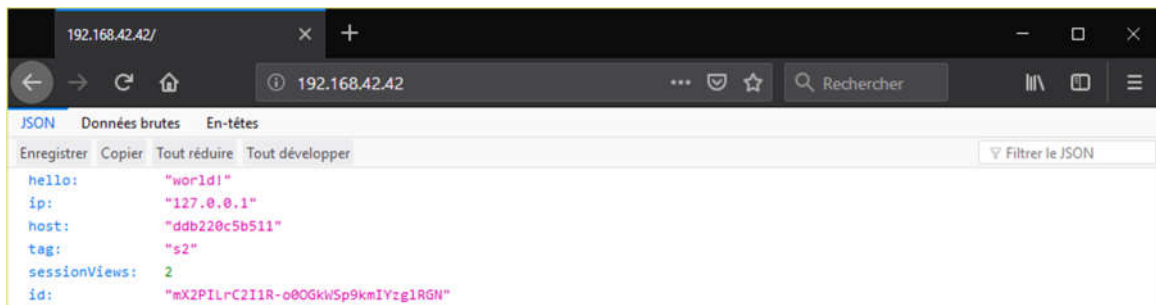


FIGURE 7 : NAVIGATION INTERNET, AVEC UN SERVEUR DOWN

4. Tâche 2 : Sticky sessions

Dans cette tâche, notre but sera de réparer la gestion des sessions par HAProxy en modifiant les fichiers de configurations.

4.1. Réponses aux questions

4.1.1. Question 1

4.1.1.1. SERVERID

Dans ce cas, nous injectons un cookie dans le navigateur du client avec l'ID du serveur qui l'a traité. Il y a donc deux cookies dans le navigateur : celui de l'application et celui du serveur. La prochaine fois que le client accède au site, il fournira le cookie du serveur qui dira au load balancer sur quel serveur il doit l'envoyer.

Lors de la première connexion du client, le HAProxy lui enverra cet en-tête : "Set-Cookie : SERVERID=s1" si le serveur choisi est le s1.

Pour les requêtes suivantes, le client aura cet en-tête : "Cookie : SERVERID=s1" à chaque demande, en indiquant au load balancer sur quel serveur il doit l'envoyer.

4.1.1.2. NOSESSID

Dans ce cas, au lieu d'injecter un cookie dans le navigateur, nous utilisons la configuration des cookies du serveur d'application. Il y a donc qu'un seul cookie, celui de l'application, qui est préfixé par le serveur traitant la requête.

Lors de la première connexion, le serveur placera le cookie qui ressemblera à ceci : "Set-Cookie : NODESESSID = s1~i12KJF23JKJJ1EKJ21213KJ"

Le cookie a été préfixé par la valeur du cookie du serveur (ici S1). Le ~ est utilisé comme séparateur entre les informations du serveur et la valeur du cookie.

Pour les requêtes suivantes, le client aura cet en-tête : "Cookie : NODESESSID = s1~i12KJF23JKJJ1EKJ21213KJ" à chaque requête, il indiquera ainsi au load balancer sur quel serveur il doit l'envoyer.

4.1.2. Question 2

Pour activer la gestion des sticky sessions, nous avons choisi d'ajouter un cookie dans le navigateur plutôt que d'utiliser celui de l'application.

Nous avons donc ajouté la ligne suivante dans le fichier haproxy.cfg après "back_end node" :

```
cookie SERVERID insert indirect nocache
```

Cette ligne dit à HAProxy d'injecter dans le browser un cookie appelé SERVERID si l'utilisateur n'en a pas déjà un. L'argument nocache ajoute le "Cache-Control : nocache" à l'en-tête puisque nous ne voulons pas garder de cookie personnel dans la cache.

Nous avons ensuite remplacé la déclaration des deux serveurs par les lignes ci-dessous :

```
server s1 <s1>:3000 check cookie s1
server s2 <s2>:3000 check cookie s2
```

Ces deux lignes déclarent les serveurs et indique au proxy qu'il doit vérifier la valeur du cookie et choisir un serveur en fonction de celle-ci.

4.1.3. Question 3

Lorsque nous ouvrons l'url pour la première fois, HAProxy injecte un cookie dans notre navigateur contenant le serveur qui nous a reçu.

```
{"hello":"world!","ip":"127.0.0.1","host":"af5cb239c578","tag":"s2","sessionViews":1,"id":"PuQ8yKhp8boUttIfyS06i8D8xuqu6H6"}
```

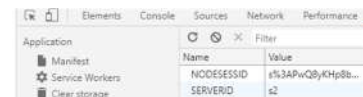


FIGURE 8 : SESSIONS, PREMIÈRE CONNEXION

À partir de ce point, chaque fois que nous accédons à cette page à partir de ce navigateur, nous serons dirigés vers le même serveur. C'est le cas tant que le cookie se trouve dans le navigateur.

```
{"hello":"world!","ip":"127.0.0.1","host":"af5cb239c578","tag":"s2","sessionViews":25,"id":"hmqinyRfheV2QrBuCrD9Bj5KIXszNw0H"}
```

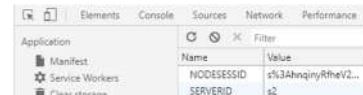


FIGURE 9 : SESSIONS, CONNECTIONS MULTIPLE

Si vous fermez le navigateur et accédez ensuite à la page à partir d'un nouveau navigateur, vous n'aurez plus votre cookie. Le serveur pourra changer par rapport au dernier et le nombre de vues sera réinitialisé. Vous recevez un nouveau cookie pour recommencer la sticky-session.

4.1.4. Question 4

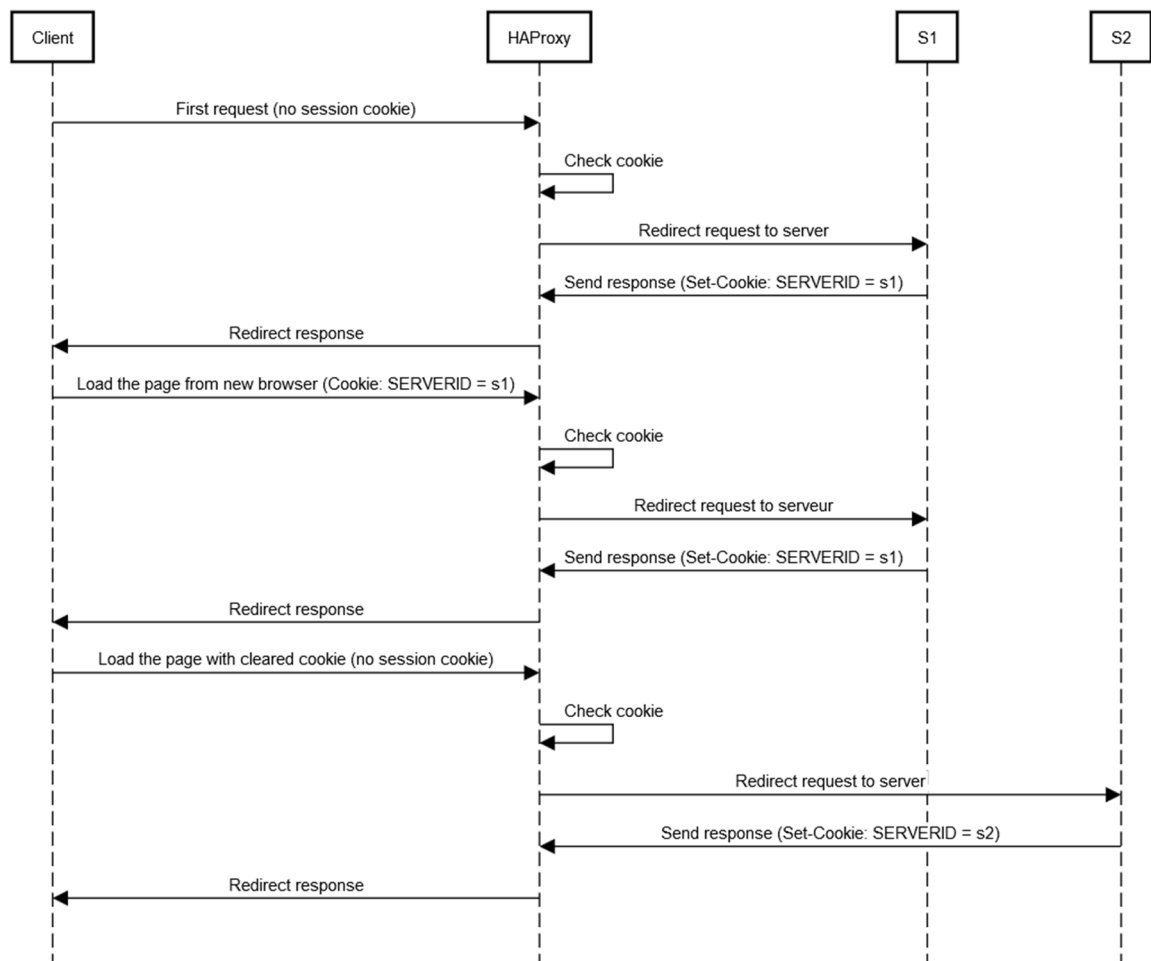


FIGURE 10 : SESSIONS, DIAGRAMME DE SÉQUENCE

4.1.5. Question 5

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ka/sec reçus	Ka/sec émis	Moy. octets
GET /	200	17	4	378	38.08	0.00%	63.9/sec	21.58	14.22	345.6
S1 reached	100	0	0	2	0.50	0.00%	58.8/sec	0.00	0.00	.0
S2 reached	100	0	0	2	0.48	0.00%	38.5/sec	0.00	0.00	.0
TOTAL	400	8	0	378	28.26	0.00%	127.8/sec	21.56	14.21	172.8

FIGURE 11 : SESSIONS, RAPPORT JMeter

Il n'y a pas de différence visible entre le run actuel et celui de la tâche 1. Les requêtes sont distribuées de façon équitable entre les deux serveurs. La seule différence est les serveurs sur lesquels les threads s'exécutent. En effet, parce-que nous avons mis des sticky-sessions, les threads effectueront toutes leurs requêtes sur le même serveur. Si, par exemple, le thread 1 s'exécute en entier puis le thread 2, alors un serveur recevra 100 requêtes puis le deuxième en recevra 100.

Ce comportement diffère de la tâche 1 où, dans l'exemple ci-dessus, la moitié des requêtes du thread 1 serait sur le serveur 1 et l'autre sur le serveur 2. De même pour le thread 2.

4.1.6. Question 6

Voir la capture de la question 5

Dans ce run, le load-balancer équilibre les requêtes entre les deux serveurs. La différence avec le premier run est que les requêtes effectuées par un thread atteindront toujours le même serveur. Donc, si le thread 1 effectue toutes ses requêtes avant le thread 2, le serveur 1 (on suppose que le thread 1 est redirigé sur le serveur 1) recevra alors toutes les requêtes et le serveur 2 n'en recevra aucune. Puis quand viendra le tour du thread 2 de s'exécuter, l'inverse se produira. Il y aura toujours un équilibre par rapport au nombre de requêtes sur chaque serveur mais on saura par contre que les requêtes qu'un serveur a reçu proviennent toutes du même thread.

5. Tâche 3 : Drain mode

5.1. Réponses aux questions

5.1.1. Question 1

nodes																													
		Queue		Session rate			Sessions				Bytes		Denied		Errors		Warnings		Server										
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Conn	Resp	Retr	Redts	Status	LastChk	Wght	Act	Blk	Chk	Dwn	Downtme	Thrls	
s1	0	0	-	0	52		0	1	-	110	1	3s	33 859	44 519	0	0	0	0	0	8m29s UP	1	Y	-	0	0	0	0s	-	
s2	0	0	-	0	58		0	1	-	104	1	2m57s	24 791	38 490	0	0	0	0	0	8m29s UP	1	Y	-	0	0	0	0s	-	
Backend	0	0	-	0	110		0	2	200	223	2	3s	58 650	82 518	0	0	0	0	0	8m29s UP	2	2	0	0	0	0	0s	-	

FIGURE 12 : DRAIN MODE, NOEUD DE RÉPONSE

On peut voir dans l'image ci-dessus que le nœud répondant aux requêtes de notre browser est le s1.

(N.B. la capture a été prise après avoir lancé le test JMeter d'où le grand nombre de requêtes).

5.1.2. Question 2

nodes		Queue		Session rate			Sessions			Bytes		Denied		Errors		Warnings		Status		LastChk		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redts		Wght	Act	Blk	Chk	Dwn	Downtme	Thrls
s1	0	0	-	0	52		0	1	-	119	1	2m11s	33 859	44 519	0	0	0	0	0	0	0	8s DRAIN	1	Y	-	0	0	0s	-
s2	0	0	-	0	58		0	1	-	104	1	5m5s	24 791	38 490	0	0	0	0	0	0	0	11m37s UP	1	Y	-	0	0	0s	-
Backend	0	0	-	0	110		0	2	200	223	2	2m11s	58 650	82 518	0	0	0	0	0	0	0	11m37s UP	1	1	0	0	0	0s	-

FIGURE 13 : DRAIN MODE, CONFIGURATION

On peut voir que le serveur 1 est maintenant en "drain mode"

5.1.3. Question 3

nodes																															
		Queue			Session rate			Sessions			Bytes		Denied		Errors		Warnings				Status		LastChk		Server						
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redts	Wght	Act	Blk	Chk	Dwn	Downtme	Thrls			
s1	0	0	-	0	52		0	1	-	135	1	8s	45 125	49 904	0	0	0	0	0	0	0	1m34s DRAIN	1	Y	-	0	0	0s	-		
s2	0	0	-	0	58		0	1	-	104	1	6m31s	24 791	38 490	0	0	0	0	0	0	0	13m3s UP	1	Y	-	0	0	0s	-		
Backend	0	0	-	0	110		0	2	200	239	2	6s	69 916	88 400	0	0	0	0	0	0	0	13m3s UP	1	1	0	0	0	0s	-		

FIGURE 14 : DRAIN MODE, COMPORTEMENT

On peut voir que lorsque l'on recharge la page, on reste quand même sur le serveur 1 malgré le fait qu'il soit en mode drain. C'est le comportement normal de cet état. En effet, les requêtes actives sur le serveur restent sur le serveur. Par contre, toutes les nouvelles requêtes sont redirigées sur le serveur 2.

5.1.4. Question 4

On est redirigé sur le serveur 1. C'est dû au fait que le cookie est toujours dans le browser donc nous avons toujours une session active avec le serveur 1. Il ne s'agit pas d'une nouvelle requête.

5.1.5. Question 5

Après avoir effacer les cookies, nous ne sommes plus en mesure d'atteindre le serveur 1. Toutes nos requêtes sont redirigées sur le serveur 2. C'est aussi le cas lorsque l'on effectue les requêtes depuis un nouveau browser. C'est un comportement normal.

Il n'est pas possible d'atteindre le serveur 1 car il est en mode "drain" et que nous n'avons pas de connexion active avec celui-ci. La seule façon d'être redirigé sur ce serveur est d'avoir, avant le changement de mode, déjà une connexion avec celui-ci. La connexion sera conservée. Le mode fonctionne donc correctement.

5.1.6. Question 6

Nodes		Queue			Session rate			Sessions					Bytes		Denied		Errors		Warnings		Status	LastChk	Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LibTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Rele			Rele	Weight	Act	Blk	Chk	Down	Downtime	Throttle
s1	0	0	-	1	52		0	1	-	154	2	1s	53 975	57 011	0	0	0	0	0	0	0	1m31s UP	L7CHK/2000 in 4ms	1	Y	-	0	0	0m45s	-
s2	0	0	-	0	58		0	1	-	140	0	51s	45 140	52 631	0	0	0	0	0	0	0	22m49s UP	L7CHK/2000 in 5ms	1	Y	-	0	0	0s	-
Backend	0	0	-	1	110		0	2	200	294	0	1s	99 115	109 642	0	0	0	0	0	0	0	22m49s UP		2	2	0	0	0	0s	-

FIGURE 15 : DRAIN MODE, READY MODE

Avant d'effacer les cookies, toutes nos requêtes restent sur le serveur 2. C'est du au fait que les sticky-sessions sont implémentées. En effet, le cookie qui a été injecté dans le browser indique que le serveur 2 est notre interlocuteur donc nous sommes toujours redirigés dessus. C'est aussi le cas lors de l'utilisation d'un nouveau browser.

Par contre, après avoir effacé les cookies, nous sommes redirigés sur le serveur 1 qui est de nouveau disponible.

5.1.7. Question 7

nodes		Queue			Session rate			Sessions			Bytes		Denied		Errors		Warnings		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LibTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Rele	Rele	Status	LastChk	Wght	Act	Blk	Chk	Down	Downtime	Thrtle
s1	0	0	-	0	52	0	1	-	154	2	3m21s	53 975	57 011	0	0	0	0	0	0	0	5s MAINT		1	Y	-	0	1	0m45s	-	
s2	0	0	-	0	58	0	1	-	140	0	4m11s	45 140	52 631	0	0	0	0	0	0	0	26m0s UP	L7CHK/2000 in 5ms	1	Y	-	0	0	0s	-	
Backend	0	0	-	0	110	0	2	200	294	0	3m21s	99 115	109 642	0	0	0	0	0	0	0	26m0s UP		1	1	0	0	0	0s	-	

FIGURE 16 : DRAIN MODE, MAINT MODE

Dans ce cas-là, nous sommes redirigés directement sur le serveur 2 même si nous avons une session ouverte avec le serveur 1. De plus, il n'est plus possible d'accéder au serveur 1 même en effaçant les cookies.

Contrairement au mode "Drain", le mode "Maint" empêche la redirection sur le serveur même si une session a été ouverte avec celui-ci.

6. Tâche 4 : Round robin in degraded mode

Dans cette partie, nous allons mettre à l'épreuve la configuration round robin mise en place précédemment. Nous allons particulièrement nous intéresser au comportement des serveurs lorsqu'une application web commence à avoir un comportement étrange. Dans le cas présent, le serveur s1 va avoir un temps de réponse allant de 0 à 2500 ms.

La modification du délai est très facile, elle peut être faite via un simple curl de la manière suivante :

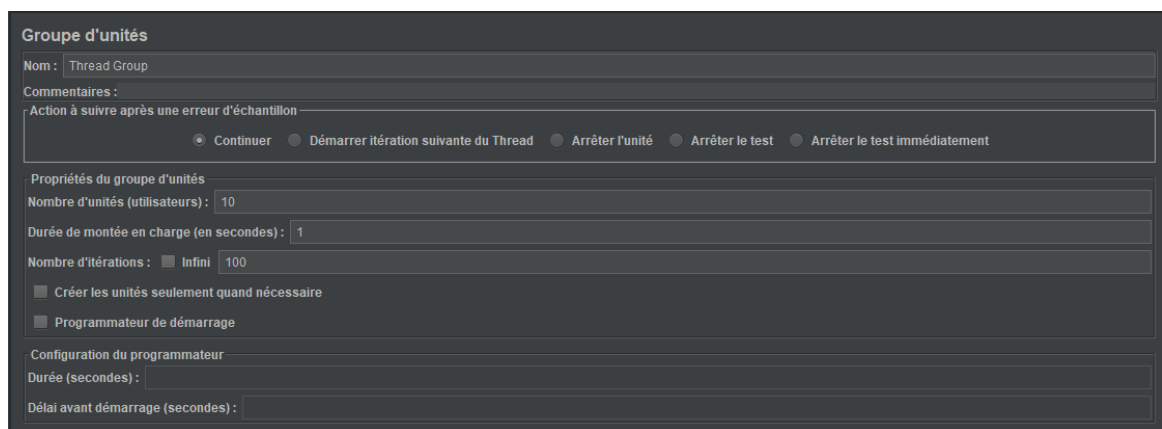
vagrant ssh

```
docker inspect --format '{{ .NetworkSettings.IPAddress }}' <containerName>
```

```
curl -H "Content-Type: application/json" -X POST -d '{"delay": X}' http://<containerIp>:3000/delay
```

La valeur X étant le délai voulu. Pour remettre les valeurs par défaut, il faut définir le délai à 0.

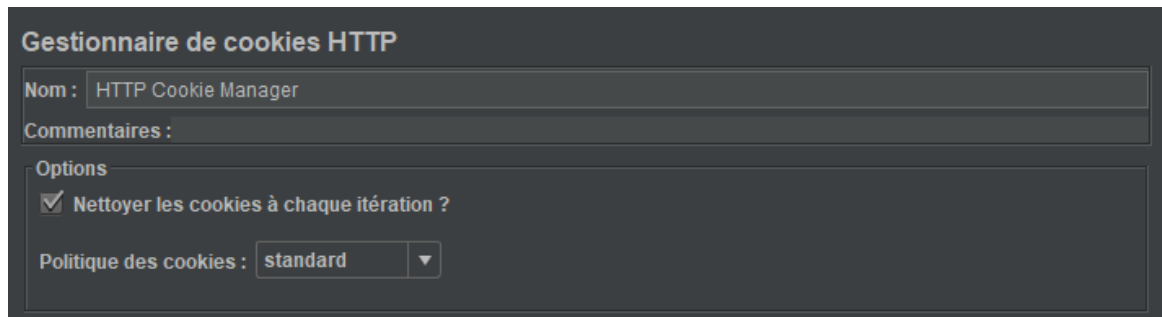
Ensuite nous allons configurer notre plan de test JMeter en mettant 10 utilisateurs dans le Thread Group :



The screenshot shows the 'Groupe d'unités' (Thread Group) configuration window in JMeter. The 'Nom' (Name) is 'Thread Group'. Under 'Action à suivre après une erreur d'échantillon' (Action to follow after a sample error), the 'Continuer' (Continue) option is selected. In the 'Propriétés du groupe d'unités' (Thread Group Properties) section, 'Nombre d'unités (utilisateurs) : 10' (Number of units (users) : 10) is set. 'Durée de montée en charge (en secondes) : 1' (Ramp-up time (in seconds) : 1) is set. 'Nombre d'itérations : 100' (Number of iterations : 100) is set with the 'Infini' (Infinite) checkbox unchecked. The 'Créer les unités seulement quand nécessaire' (Create units only when necessary) checkbox is checked, and the 'Programmeur de démarrage' (Startup script) checkbox is unchecked. The 'Configuration du programmeur' (Startup script configuration) section is empty.

FIGURE 17 : JMETER, NOMBRE D'UTILISATEURS

Pour la question 6, nous allons également changer la politique de gestion des cookies afin qu'ils soient supprimés entre chaque itérations :



The screenshot shows the 'Gestionnaire de cookies HTTP' (HTTP Cookie Manager) configuration window in JMeter. The 'Nom' (Name) is 'HTTP Cookie Manager'. Under 'Options', the 'Nettoyer les cookies à chaque itération ?' (Clean cookies at each iteration?) checkbox is checked. The 'Politique des cookies : standard' (Cookie policy : standard) dropdown menu is set to 'standard'.

FIGURE 18 : JMeter, GESTION DES COOKIES

6.1. Réponses aux questions

6.1.1. Question 1

Dans un premier temps, nous allons mettre le délai à 0 sur nos serveurs afin d'avoir des valeurs indicatives :

```
vagrant@ubuntu-14:~$ docker inspect --format '{{ .NetworkSettings.IPAddress }}' s1
172.17.0.2
vagrant@ubuntu-14:~$ docker inspect --format '{{ .NetworkSettings.IPAddress }}' s2
172.17.0.3
vagrant@ubuntu-14:~$ curl -H "Content-Type: application/json" -X POST -d '{"delay": 0}' http://172.17.0.2:3000/delay
{"message": "New timeout of 0ms configured."}
vagrant@ubuntu-14:~$ curl -H "Content-type: application/json" -X POST -d '{"delay": 0}' http://172.17.0.3:3000/delay
{"message": "New timeout of 0ms configured."}
vagrant@ubuntu-14:~$
```

FIGURE 19 : SERVEURS, CHANGEMENT DE DÉLAI

Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : ☐ Erreurs ☒ Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	1000	11	2	48	6.43	0.00%	447.6/sec	151.11	101.26	345.7
S1 reached	500	0	0	1	0.31	0.00%	233.2/sec	0.00	0.00	.0
S2 reached	500	0	0	1	0.30	0.00%	236.3/sec	0.00	0.00	.0
TOTAL	2000	5	0	48	7.33	0.00%	894.9/sec	151.04	101.21	172.8

FIGURE 20 : ROUND ROBIN, VALEUR INDICATIVES

Comme on peut le voir, les requêtes sont parfaitement distribuées entre nos deux serveurs.

6.1.2. Question 2

Cette fois-ci, nous allons regarder le comportement de nos serveurs lorsque S1 réponds avec un délai de 250ms.

Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : ☐ Erreurs ☒ Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	1000	721	4	3073	713.88	0.00%	7.0/sec	2.35	1.58	345.7
S1 reached	500	0	0	3	0.48	0.00%	3.5/sec	0.00	0.00	.0
S2 reached	500	0	0	8	0.57	0.00%	103.6/sec	0.00	0.00	.0
TOTAL	2000	360	0	3073	620.33	0.00%	13.9/sec	2.35	1.58	172.8

FIGURE 21 : ROUND ROBIN, DÉLAI DE 250MS

Comme on peut le voir sur la capture ci-dessus, un délai de 250ms n'est pas suffisant pour perturber nos serveurs, et par conséquent, les requêtes sont parfaitement distribuées entre eux. Cependant, on peut remarquer une perte de performance.

6.1.3. Question 3

On recommence après avoir augmenter le délai à 2500ms.

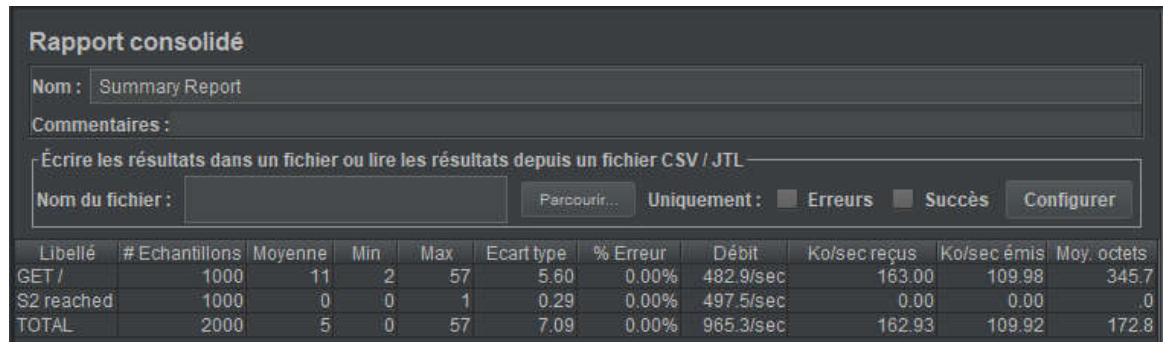


FIGURE 22 : ROUND ROBIN, DÉLAI DE 2500MS

Dans ce cas-ci, le délai est beaucoup trop lent et le premier serveur est évité par les requêtes, par conséquent, il sera rarement, voir jamais, atteint comme on peut le voir sur la capture ci-dessus.

6.1.4. Question 4

Il est normal qu'il n'y ait pas d'erreur, HAProxy redirige ses requêtes d'un serveur à l'autre selon le round robin, cependant il attend de recevoir une réponse de la part du serveur. Dans notre cas, il envoie une requête à S1 et pendant que celui-ci la traite, le serveur S2 va s'occuper de toutes les suivantes, ensuite quand S1 sera de nouveau disponible, il prendra la prochaine requête.

6.1.5. Question 5

Il faut mettre les lignes suivantes dans le fichier de configuration de HAProxy :

```
server s1 <s1>:3000 weight 2 check cookie s1
server s2 <s2>:3000 weight 1 check cookie s2
```


6.1.6. Question 6

Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : ☐ Erreurs ☐ Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	1000	1372	3	2335	901.45	0.00%	5.1/sec	1.71	1.16	345.7
S1 reach...	700	0	0	11	0.72	0.00%	3.6/sec	0.00	0.00	.0
S2 reach...	300	0	0	3	0.48	0.00%	144.6/sec	0.00	0.00	.0
TOTAL	2000	686	0	2335	936.50	0.00%	10.1/sec	1.71	1.16	172.8

FIGURE 23 : ROUND ROBIN, 250MS AVEC COOKIES

Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : ☐ Erreurs ☐ Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	1000	134	3	4267	538.14	0.00%	68.0/sec	34.92	7.77	525.6
S1 reach...	43	0	0	3	0.79	0.00%	3.5/sec	0.00	0.00	.0
S2 reach...	957	0	0	2	0.40	0.00%	65.7/sec	0.00	0.00	.0
TOTAL	2000	67	0	4267	386.41	0.00%	136.1/sec	34.92	7.77	262.8

FIGURE 24 : ROUND ROBIN, 250MS SANS COOKIES

Le changement est assez majeur, on peut voir que l'utilisation des cookies avec un poids et un serveur lent peut énormément ralentir les performances globales de notre load balancer. On passe de 10 requêtes par secondes à plus de 130 requêtes par seconde sans les cookies soit une amélioration de 13 fois.

7. Tâche 5 : Balancing strategies

Lors de ce labo, nous avons principalement travaillé en mode round robin, cependant, HAProxy propose également bien d'autres modes que nous allons regarder plus en détails à partir de maintenant.

7.1. Réponses aux questions

7.1.1. Question 1

En lisant la documentation de HAProxy, on se rend compte qu'il y a beaucoup d'algorithmes mais dans le cadre du laboratoire, nous en avons choisi les deux suivants :

Leastconn : Le choix du serveur se fait selon le nombre de connections sur chacun d'entre eux. Si plusieurs serveurs peuvent être choisis, on applique le round robin afin de tous les utiliser au moins une fois.

Source : Le choix du serveur se fait en hachant l'adresse ip source et en effectuant un modulo dessus selon le nombre de serveur total.

7.1.2. Question 2

Pour changer la configuration du load balancer de HAProxy, il faut changer l'algorithme du mot clé « balance » de la partie « backend nodes » du fichier haproxy.cfg. Dans l'exemple suivant, on utilise le mode leastconn, pour changer d'algorithme, il suffit de commenter et décommenter les lignes voulues.

```
94      # Define the balancing policy
95      # http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#balance
96      balance leastconn
97      #balance source
98      #balance roundrobin
```

FIGURE 25 : HAPROXY, CHANGEMENT DE LOAD BALANCER

Maintenant, on va faire des essais avec le script JMeter fourni sur nos deux nouveaux algorithmes. Nous avons choisi de faire les tests avec et sans les cookies comme dans la partie 4 afin de voir s'il y aurait des différences.

Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : ☐ Erreurs ☐ Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	1000	37	5	111	19.58	0.00%	213.9/sec	72.21	48.43	345.7
S1 reached	500	0	0	7	0.70	0.00%	114.1/sec	0.00	0.00	.0
S2 reached	500	0	0	6	0.49	0.00%	114.0/sec	0.00	0.00	.0
TOTAL	2000	18	0	111	23.04	0.00%	427.7/sec	72.19	48.42	172.8

FIGURE 26 : LEASTCONN AVEC COOKIES

Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : ☐ Erreurs ☐ Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	1000	19	3	71	9.32	0.00%	324.8/sec	166.69	37.11	525.6
S1 reached	521	0	0	2	0.37	0.00%	171.0/sec	0.00	0.00	.0
S2 reached	479	0	0	2	0.39	0.00%	157.9/sec	0.00	0.00	.0
TOTAL	2000	9	0	71	11.81	0.00%	649.4/sec	166.64	37.10	262.8

FIGURE 27 : LEASTCONN SANS COOKIES

Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : ☐ Erreurs ☐ Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	1000	29	3	84	12.90	0.00%	252.1/sec	85.09	57.02	345.7
S2 reached	1000	0	0	6	0.35	0.00%	253.3/sec	0.00	0.00	.0
TOTAL	2000	14	0	84	17.42	0.00%	504.0/sec	85.07	57.01	172.8

FIGURE 28 : SOURCE AVEC COOKIES

Rapport consolidé

Nom : Summary Report

Commentaires :

Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier : Parcourir... Uniquement : ☐ Erreurs ☐ Succès Configurer

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	1000	23	4	68	10.30	0.00%	279.7/sec	143.55	31.96	525.5
S2 reached	1000	0	0	2	0.35	0.00%	282.7/sec	0.00	0.00	.0
TOTAL	2000	11	0	68	13.76	0.00%	559.3/sec	143.51	31.95	262.8

FIGURE 29 : SOURCE SANS COOKIES

Comme on peut le voir ci-dessus, avec l'algorithme leastconn, nous avons une bonne répartition des charges. Sans les cookies, il arrive que le résultat ne soit pas parfait comme on peut le voir mais il reste plus que correct pour nos besoins.

L'algorithme source quand a lui redirige toute nos requêtes vers le serveur s2, soit le serveur choisi par son algorithme de hash par défaut. Ce comportement nous montre les problèmes que l'on peut rencontrer lorsque l'on utilise cet algorithme et que nos utilisateurs ont la même adresse IP. Un exemple concret serait l'HEIG-VD où chaque étudiant possède la même adresse IP public en sortie, nous serions tous redirigé vers le même serveur ce qui peut poser énormément de soucis et empêcherait le load balancer de faire son travail correctement.

7.1.3. Question 3

Finalement, ce que l'on peut retenir des deux algorithmes, c'est que selon nos besoins ils ont chacun leur avantages et désavantages et qu'il faut bien les choisir afin d'optimiser nos ressources au maximum.

Leastconn : Cet algorithme est très utile dans le cas où les sessions sont longues, par exemple avec sessions SQL ou LDAP cependant, dans le cas présent, comme les session http sont relativement courte il n'est pas très adapté. En revanche il s'agit de la meilleure option que l'on possède parmi les trois que l'on a vu lors de ce laboratoire.

Source : Cette méthode garanti qu'un client sera toujours redirigé vers le même serveur, jusqu'à ce qu'on ajoute ou retire un serveur. Dans le cas ou le nombre de serveur change, il faudra recalculer les hash et rediriger tous les clients vers un nouveau serveur, ceci peut prendre un certain temps s'il y en a beaucoup. De plus comme dit lors de la question précédente, il n'est pas adapté si plusieurs utilisateurs passent par un NAT, comme à la HEIG-VD.

8. Conclusion

Ce laboratoire nous a permis de découvrir des outils fort intéressants tel que JMeter et surtout HAProxy, ce dernier est un proxy disposant de nombreuses options pour la gestion de serveurs d'application. Ici, nous nous sommes principalement concentrés sur son load balancer lors de plusieurs étapes.

Dans un premier temps, nous avons découvert les outils fournis et avons pu nous rendre compte que le load balancer n'avait pas été entièrement configuré.

Par la suite, nous avons cherché deux manières de configurer les sticky sessions, qui sont très importantes dans de nombreuses applications web. Une fois ces méthodes mises en place, nous les avons testées et documentées.

Une fois que nos sticky sessions étaient en état de marche, nous avons regardé les différentes options disponibles, à savoir, le drain, maint, et normal mode afin de voir l'utilité de chacun d'eux.

À partir de ce moment, nous avons un HAProxy fonctionnel, cependant, dans le web, il arrive parfois qu'un serveur soit surchargé par d'autres applications, ou qu'il soit simplement plus lent, nous avons donc simulé un ralentissement sur le serveur S1 et regarder comment HAProxy réagirait, les résultats étaient impressionnants. Nous avons constaté que bien que la stratégie de gestion soit en round robin, il arrive qu'il soit en partie ignoré afin de garantir le fonctionnement des serveurs.

Et pour finir, nous avons remplacé la stratégie round robin par deux autres que nous avons choisi dans parmi toutes celles dont HAProxy dispose. Nous avons constaté qu'il y en a beaucoup et qu'elles ont chacune leurs avantages et leurs défauts.

Ce laboratoire fut fort intéressant, il nous a permis de mettre en pratique ce que l'on a pu voir en cours, l'outil HAProxy s'avère être beaucoup plus puissant qu'on ne l'aurait imaginé et le fait de voir son comportement lors d'une montée en charge et d'accès concurrents fut très instructif.

9. Table des illustrations

FIGURE 1 : ARCHITECTURE	3
FIGURE 2 : CONTENU DE LA RÉPONSE	3
FIGURE 3 : RECHARGEMENT D'UNE PAGE	4
FIGURE 4 : DIAGRAMME DE SÉQUENCE, RECHARGEMENT DE PAGE	5
FIGURE 5 : JMETER, COMPORTEMENT PAR DÉFAUT	5
FIGURE 6 : JMETER, AVEC UN SERVEUR DOWN	6
FIGURE 7 : NAVIGATION INTERNET, AVEC UN SERVEUR DOWN	6
FIGURE 8 : SESSIONS, PREMIÈRE CONNEXION	8
FIGURE 9 : SESSIONS, CONNEXIONS MULTIPLE	8
FIGURE 10 : SESSIONS, DIAGRAMME DE SÉQUENCE	9
FIGURE 11 : SESSIONS, RAPPORT JMETER	9
FIGURE 12 : DRAIN MODE, NOEUD DE RÉPONSE	11
FIGURE 13 : DRAIN MODE, CONFIGURATION	11
FIGURE 14 : DRAIN MODE, COMPORTEMENT	11
FIGURE 15 : DRAIN MODE, READY MODE	12
FIGURE 16 : DRAIN MODE, MAINT MODE	12
FIGURE 17 : JMETER, NOMBRE D'UTILISATEURS	13
FIGURE 18 : JMETER, GESTION DES COOKIES	13
FIGURE 19 : SERVEURS, CHANGEMENT DE DÉLAI	14
FIGURE 20 : ROUND ROBIN, VALEUR INDICATIVES	14
FIGURE 21 : ROUND ROBIN, DÉLAI DE 250MS	14
FIGURE 22 : ROUND ROBIN, DÉLAI DE 2500MS	15
FIGURE 23 : ROUND ROBIN, 250MS AVEC COOKIES	16
FIGURE 24 : ROUND ROBIN, 250MS SANS COOKIES	16
FIGURE 25 : HAPROXY, CHANGEMENT DE LOAD BALANCER	17
FIGURE 26 : LEASTCONN AVEC COOKIES	18
FIGURE 27 : LEASTCONN SANS COOKIES	18
FIGURE 28 : SOURCE AVEC COOKIES	18
FIGURE 29 : SOURCE SANS COOKIES	18

10. Table des références

Site officiel de HAProxy <http://www.haproxy.org/>

Documentation officielle de HAProxy <http://www.haproxy.org/#docs>

Configuration de HAProxy <http://cbonte.github.io/haproxy-dconv/1.6/configuration.html>

Site officiel de vagrant <https://www.vagrantup.com/>

Site officiel de jmeter <http://jmeter.apache.org/>

Site officiel de git-scm <https://git-scm.com/>

Configuration SSH pour git <https://help.github.com/articles/connecting-to-github-with-ssh/>

Configuration des sticky sessions <https://www.haproxy.com/fr/blog/load-balancing-affinity-persistence-sticky-sessions-what-you-need-to-know/>