AIT - Laboratoire N°3

Load Balancing


```
Auteurs:
Sacha Perdrizat
Alban Favre
Moïn Danai
```


Introduction

Dans ce laboratoire nous allons explorer les fonctionnalitées du `Load Balancer` HAproxy. Ce rapport découpe les livrables dans les plusieurs tâches.

Tâche 1

1. Explain how the load balancer behaves when you open and refresh the URL http://192.168.42.42 in your browser. Add screenshots to complement your explanations. We expect that you take a deeper a look at session management.

Réponse:

HAProxy fonctionne avec des *frontends* et *backends*.

2. Explain what should be the correct behavior of the load balancer for session management.

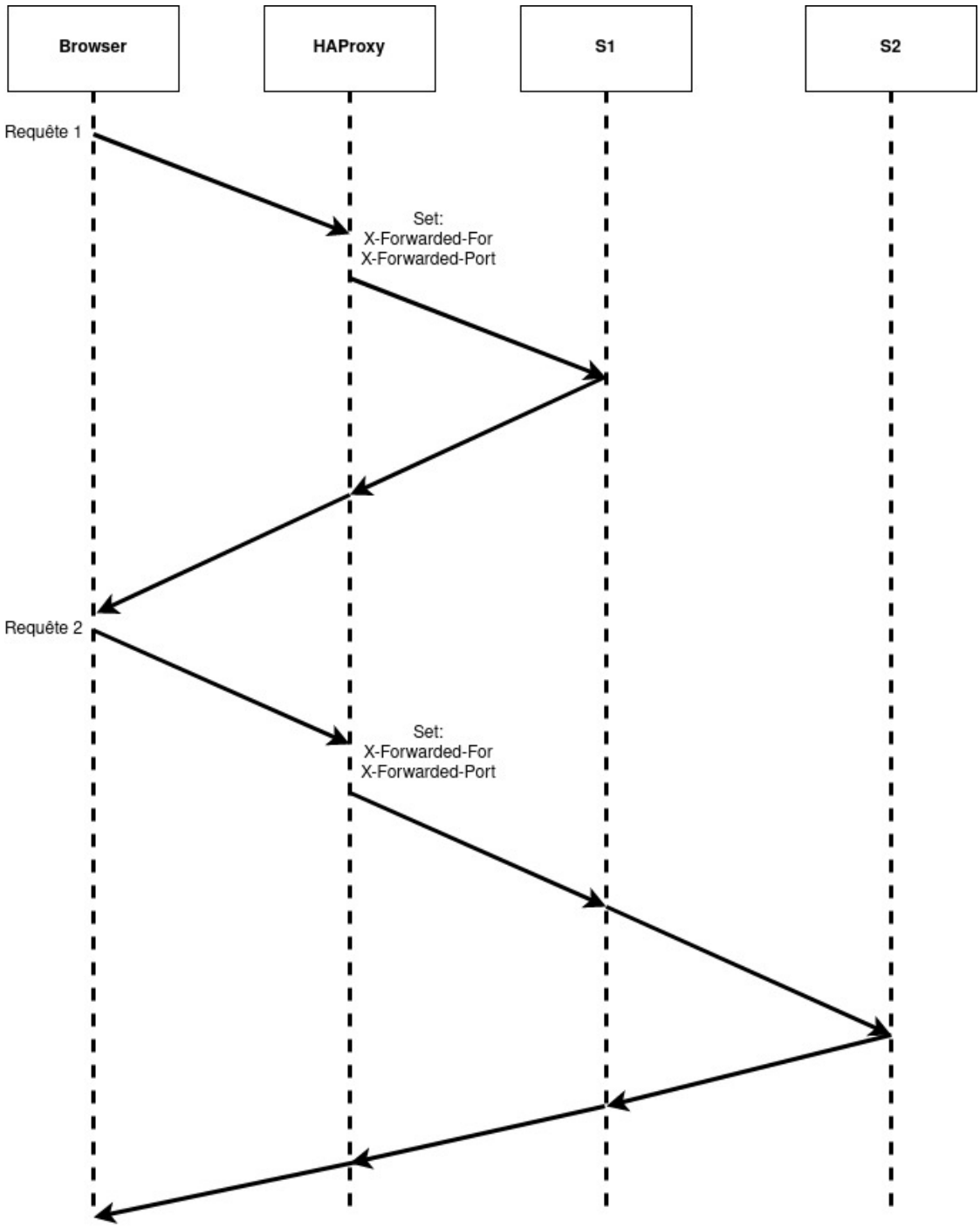
   Réponse:

   Pour une bonne gestion des sessions l'on s'attend à ce le load Balancer route les requêtes aux serveur en fonction d'un cookie de sessions

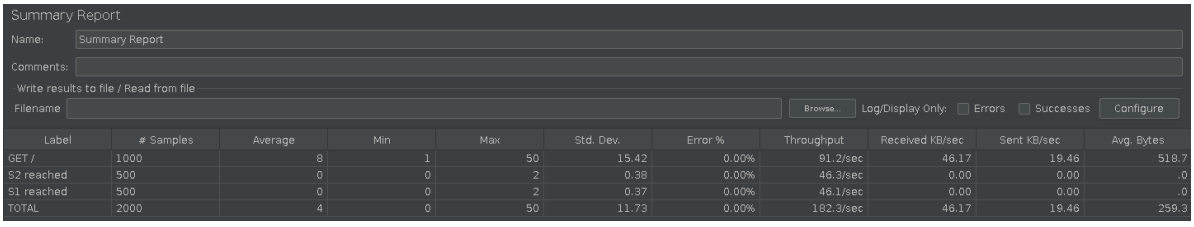
3. Provide a sequence diagram to explain what is happening when one requests the URL for the first time and then refreshes the page. We want to see what is happening with the cookie. We want to see the sequence of messages exchanged (1) between the browser and HAProxy and (2) between HAProxy and the nodes S1 and S2.

   Réponse

   On the flowchart below, we can see how HAProxy injects two headers that pertain to the proper identification of client requests being proxied. As the backend mode is round-robin, every request is sent to S1 while every other request is sent to S2.



4. Provide a screenshot of the summary report from JMeter.



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: | Errors | Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	1000	8	1	50	15.42	0.00%	91.2/sec	46.17	19.46	518.7
S2 reached	500	0	0	2	0.38	0.00%	46.3/sec	0.00	0.00	.0
S1 reached	500	0	0	2	0.37	0.00%	46.1/sec	0.00	0.00	.0
TOTAL	2000	4	0	50	11.73	0.00%	182.3/sec	46.17	19.46	259.3

5. Run the following command:

```
$ docker stop s1
```

Clear the results in JMeter and re-run the test plan. Explain what is happening when only one node remains active. Provide another sequence diagram using the same model as the previous one.

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

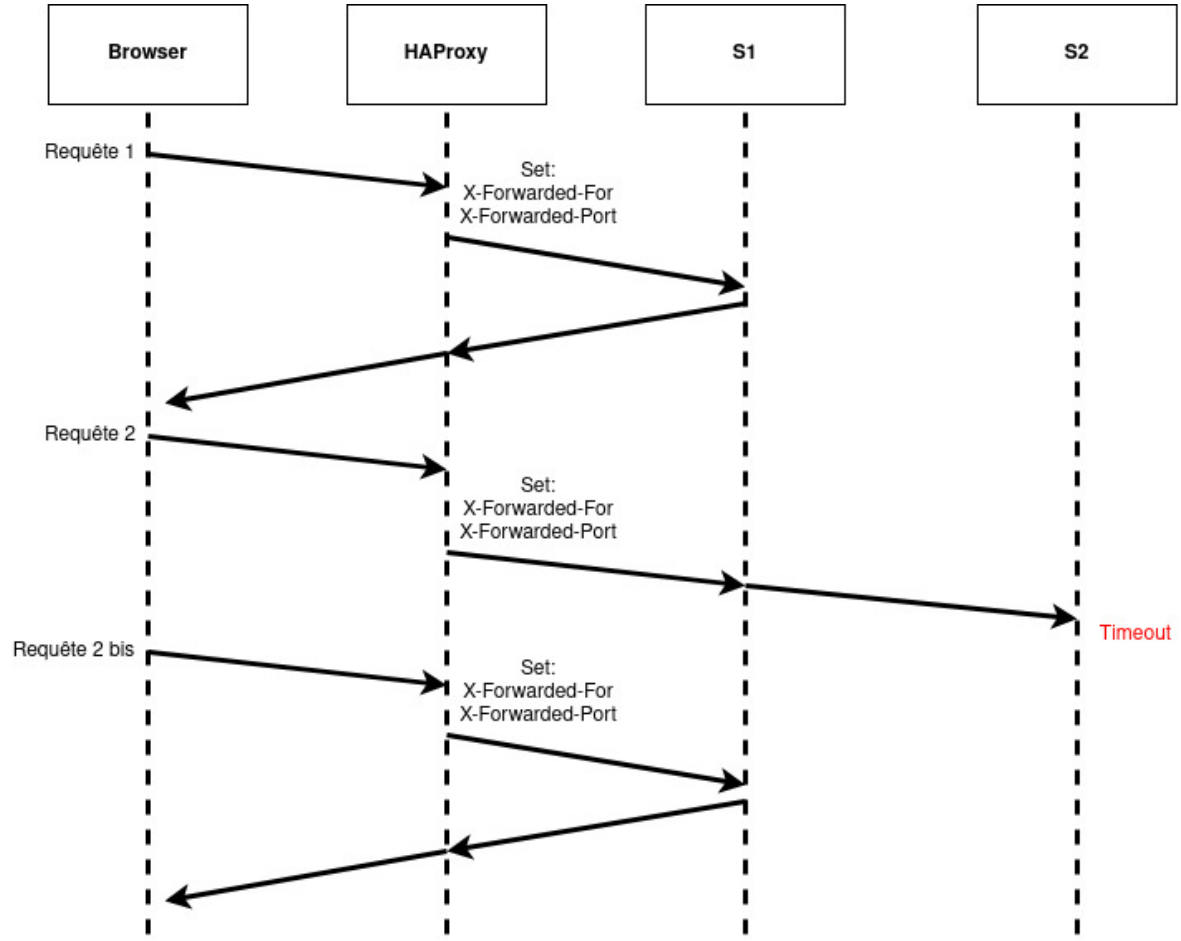
Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	1000	7	1	51	14.51	0.00%	113.3/sec	43.69	24.10	395.0
S2 reached	1000	0	0	2	0.35	0.00%	113.4/sec	0.00	0.00	.0
TOTAL	2000	3	0	51	10.91	0.00%	226.5/sec	43.69	24.10	197.5

We can see HAProxy spitting some warnings:

```
ha          | [WARNING] 329/152730 (10) : Server nodes/s1 is DOWN, reason:
Layer4 timeout, check duration: 2002ms. 1 active and 0 backup servers left. 0
sessions active, 0 requeued, 0 remaining in queue.
```

Indeed, HAProxy timed out trying to reach `/` on s1 so it skipped marked it DOWN and skipped it for future requests.

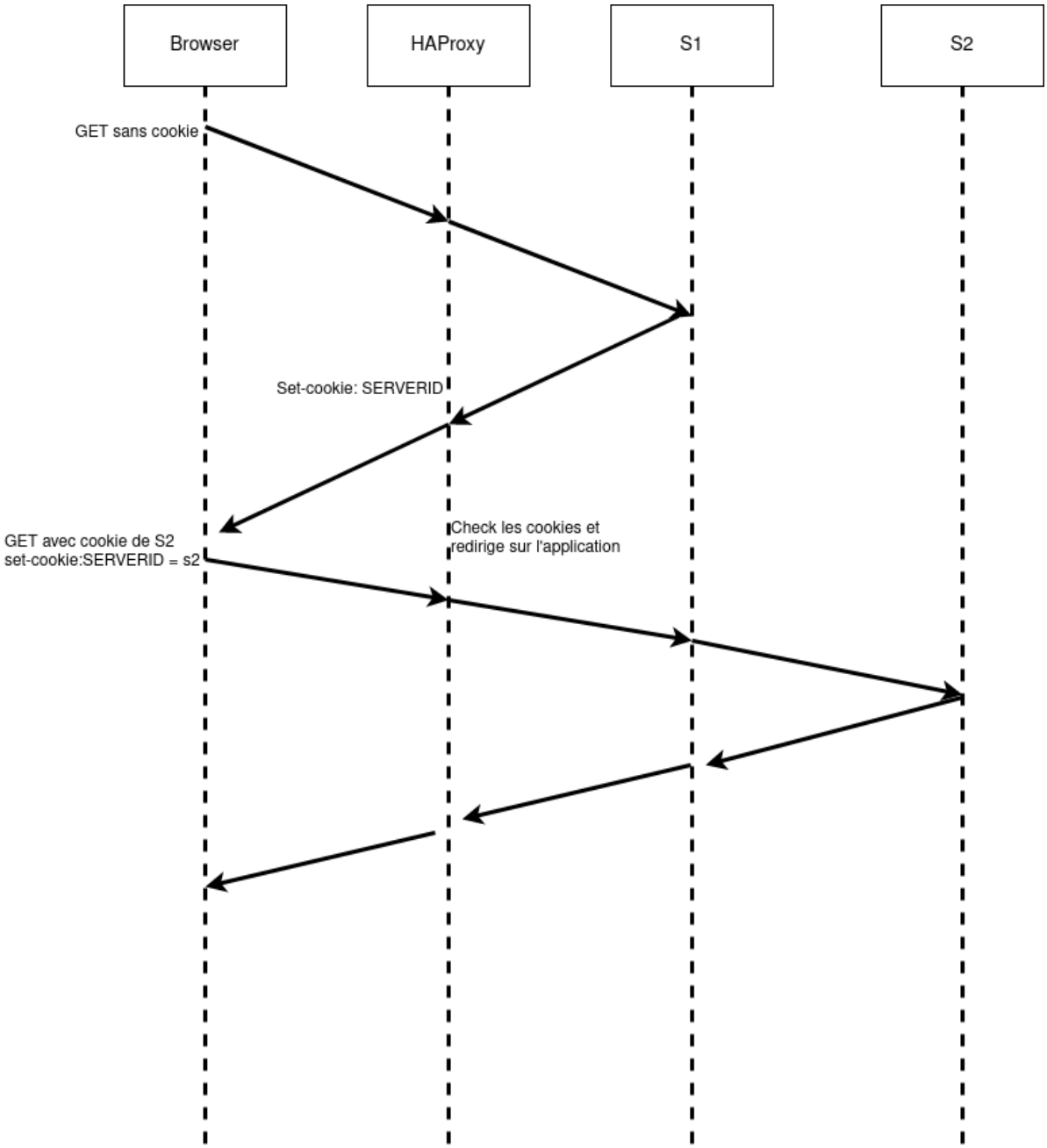


Tâche 2

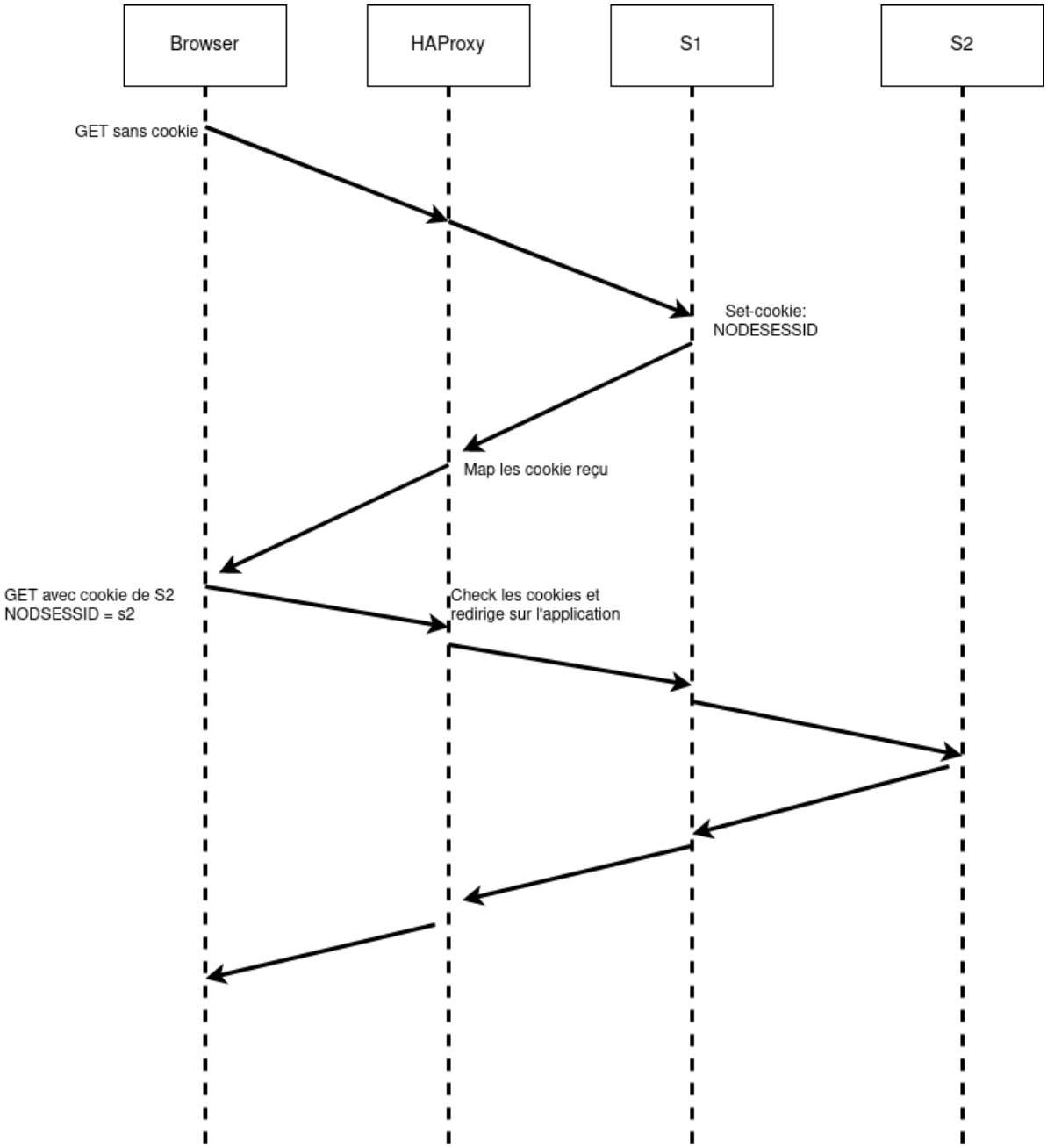
1. There is different way to implement the sticky session. One possibility is to use the SERVERID provided by HAProxy. Another way is to use the NODESESSID provided by the application. Briefly explain the difference between both approaches (provide a sequence diagram with cookies to show the difference).

- Choose one of the both stickiness approach for the next tasks.

Avec le SERVERID



Avec le NODESESSID



Il est recommandé d'utilisé le cookie SERVERID, comme il sont délivré par le load balancer il seront plus fiable. Il n'est pas garantit que les cookies délivré par les applications soit distinct

2. Provide the modified `haproxy.cfg` file with a short explanation of the modifications you did to enable sticky session management.

Pour utiliser le NODESESSID ou le SERVERID qui nous est donnée par l'application il convient d'ajouter ceci à la configuration

```
backend node

  cookie {NODESESSID | SERVERID} insert indirect nocache
  server s1 ${WEBAPP_1_IP}:3000 check cookie s1
  server s2 ${WEBAPP_2_IP}:3000 check cookie s2
```

Le tout en redémarrant le load balancer

NB: Nous utiliserons le SERVERID pour la suite des exemples

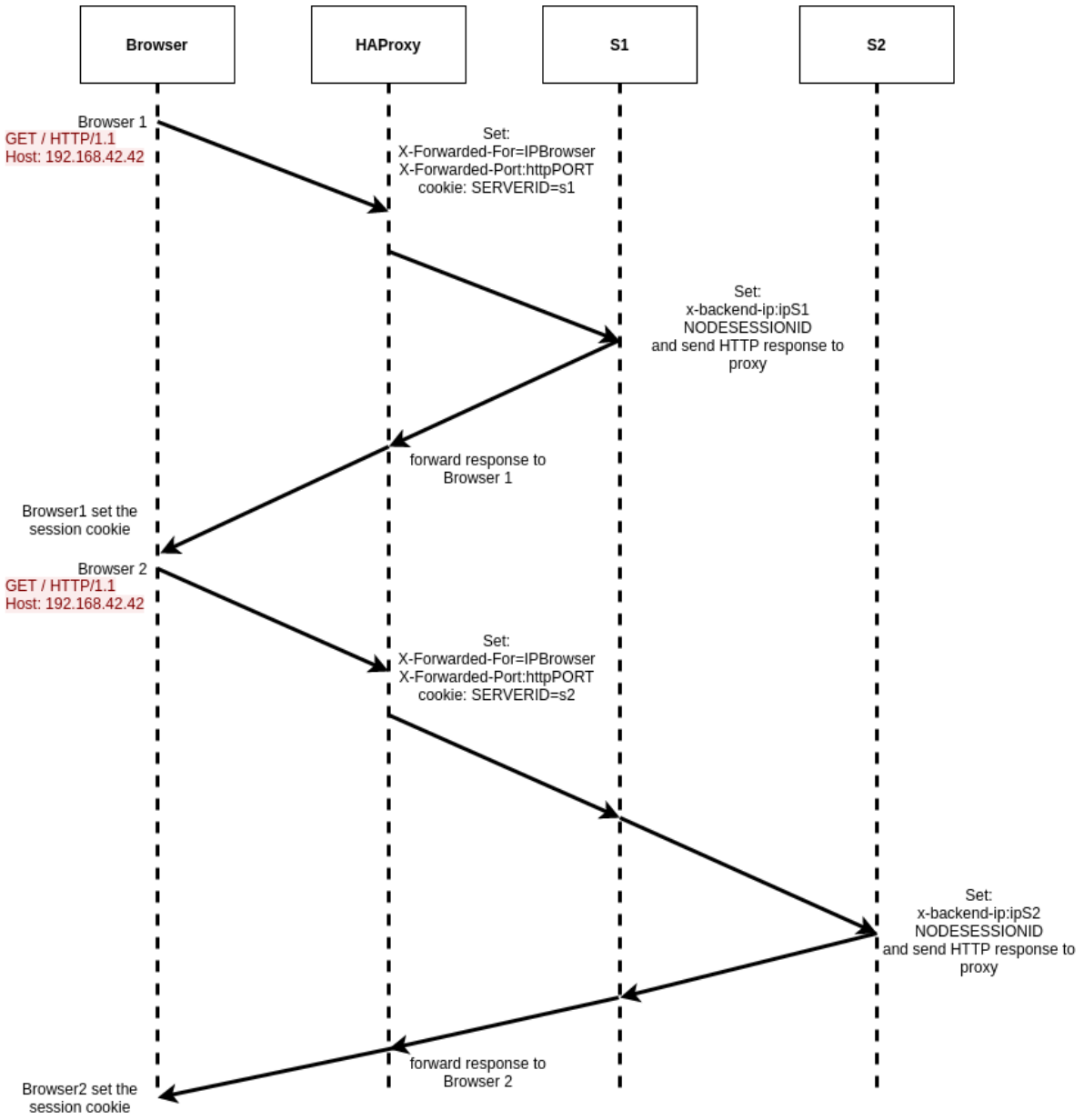
3. Explain what is the behavior when you open and refresh the URL http://192.168.42.42 in your browser. Add screenshots to complement your explanations. We expect that you take a deeper a look at session management.

Lorsque nous faisons une requête et plusieurs `refresh` nous observons que le load Balancer nous renvoie sur le même serveur.

Aperçu d'une capture sur Wireshark

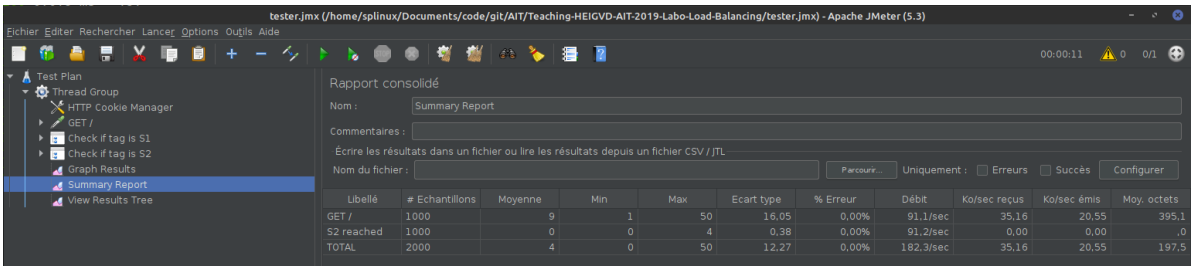
```
Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
    x-powered-by: Express\r\n
    x-backend-ip: 192.168.42.11\r\n
    content-type: application/json; charset=utf-8\r\n
    content-length: 129\r\n
        [Content length: 129]
    etag: W/"81-VrGks1g0NUpZwi+ckNKdYx7w5t0"\r\n
    # Préparation du NODESESSID ######
    set-cookie:
NODESESSID=s%3Ad83uu7wWdkEIyVFM0rx_wqNuyXvmoB8R.myRuFLo3DjR2ZBQC9GEL3GAVIvd4kKF
ce2TS7zGNIw0; Path=/; HttpOnly\r\n
    ##################################
    vary: Accept-Encoding\r\n
    date: Wed, 25 Nov 2020 15:18:29 GMT\r\n
    keep-alive: timeout=5\r\n
    # Préparation du SERVERID #########
    set-cookie: SERVERID=s1; path=/\r\n
    ##################################
    cache-control: private\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.005106200 seconds]
    [Request in frame: 39]
    [Request URI: http://192.168.42.42/]
    File Data: 129 bytes
```

4. Provide a sequence diagram to explain what is happening when one requests the URL for the first time and then refreshes the page. We want to see what is happening with the cookie. We want to see the sequence of messages exchanged (1) between the browser and HAProxy and (2) between HAProxy and the nodes S1 and S2. We also want to see what is happening when a second browser is used.



5. Provide a screenshot of JMeter's summary report. Is there a difference with this run and the run of Task 1?

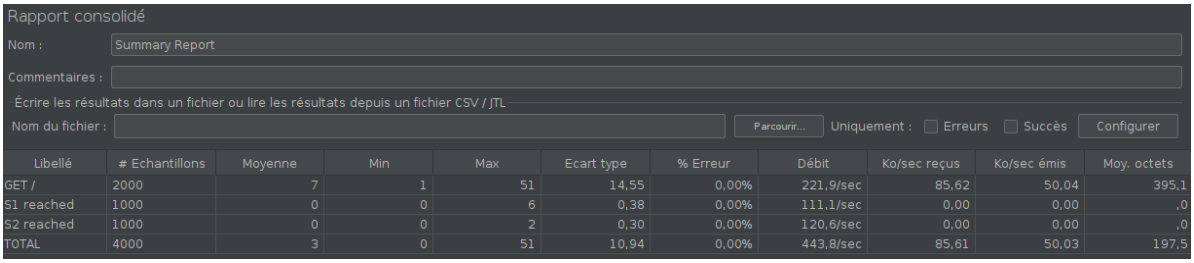
En observant la capture des requêtes sur Jmeter On remarque que toutes les requêtes ont été délivré sur le même serveur



- Clear the results in JMeter.
- Now, update the JMeter script. Go in the HTTP Cookie Manager and ~~uncheck~~verify that the box `Clear cookies each iteration?` is unchecked.
- Go in `Thread Group` and update the `Number of threads`. Set the value to 2.

7. Provide a screenshot of JMeter's summary report. Give a short explanation of what the load balancer is doing.

Le test JMeter ci-dessous va effectuer les requêtes sur 2 threads(users), on va donc observer que pour chaque utilisateur les session vont se répartir entre les serveur, 1000 sur s1 et 1000 sur s2


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 /	2000	7	1	51	14,55	0,00%	221,9/sec	85,62	50,04	395,1
S1 reached	1000	0	0	6	0,38	0,00%	111,1/sec	0,00	0,00	,0
S2 reached	1000	0	0	2	0,30	0,00%	120,6/sec	0,00	0,00	,0
TOTAL	4000	3	0	51	10,94	0,00%	443,8/sec	85,61	50,03	197,5

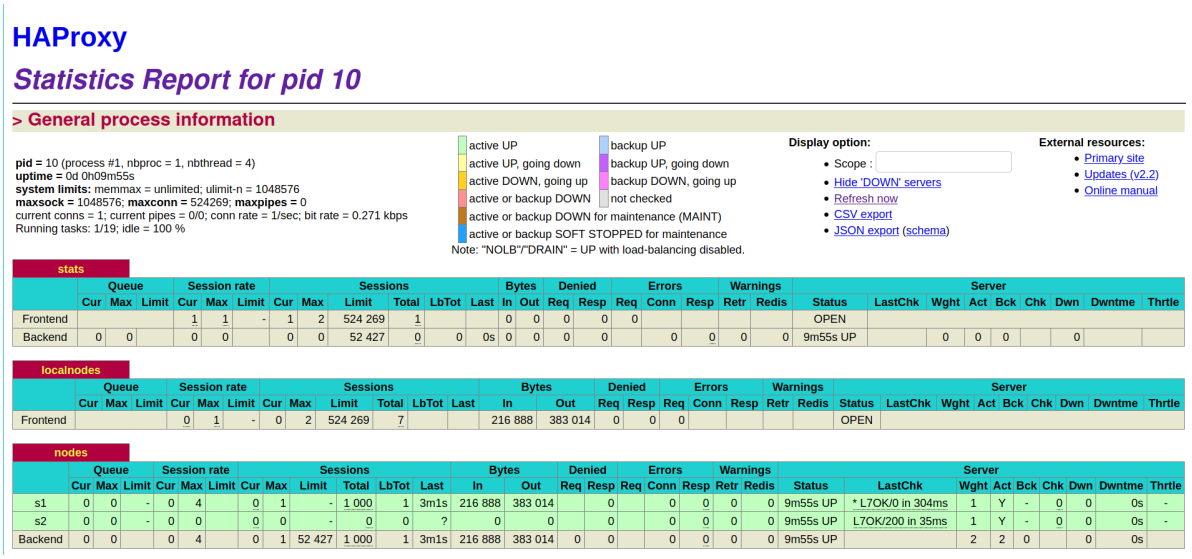
Tâche 3

1. Take a screenshot of the Step 5 and tell us which node is answering.
   In the screen we can see that onyl s2 is answering

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	1000	361	303	668	103,51	0,00%	2,7/sec	1,06	0,62	395,1
S1 reached	1000	0	0	6	0,54	0,00%	2,7/sec	0,00	0,00	,0
TOTAL	2000	181	0	668	195,04	0,00%	5,5/sec	1,06	0,62	197,5

2. Based on your previous answer, set the node in DRAIN mode. Take a screenshot of the HAProxy state page.

Answer


HAProxy

Statistics Report for pid 10

> General process information

pid = 10 (process #1, nbproc = 1, nbthread = 4)
uptime = 0d 0h09m55s
system limits: memmax = unlimited; ulimit-n = 1048576
maxsock = 1048576; **maxconn =** 524269; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec; bit rate = 0.271 kbps
Running tasks: 1/19; idle = 100 %

active UP
active UP, going down
active DOWN, going up
active or backup DOWN
active or backup DOWN for maintenance (MAINT)
active or backup SOFT STOPPED for maintenance
backup UP
backup UP, going down
backup DOWN, going up
not checked
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Refresh now
- CSV export
- JSON export (schema)

External resources:
- Primary site
- Updates (v2.2)
- Online manual

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				1	1	-	1	2	524 269	1			0	0	0	0	0					OPEN								
Backend	0	0		0	0		0	0	52 427	0	0	0s	0	0	0	0		0	0	0	0	9m55s UP		0	0	0		0		

localnodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	0	2	524 269	7			216 888	383 014	0	0	0					OPEN								

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	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
s1	0	0	-	0	4		0	1	-	1 000	1	3m1s	216 888	383 014		0		0	0	0	0	9m55s UP	* L7OK/0 in 304ms	1	Y	-	0	0	0s	-
s2	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	9m55s UP	L7OK/200 in 35ms	1	Y	-	0	0	0s	-
Backend	0	0		0	4		0	1	52 427	1 000	1	3m1s	216 888	383 014	0	0		0	0	0	0	9m55s UP		2	2	0		0	0s	

3. Refresh your browser and explain what is happening. Tell us if you stay on the same node or not. If yes, why? If no, why?

Answer

No, we are now getting replies from s1 only since s2 is being drain of requests and slowly set aside (i.e. not getting requests anymore, out of commission).

4. Open another browser and open `http://192.168.42.42`. What is happening?

Answer

We get the following:

```
{
  "hello":"world!",
  "ip":"192.168.42.11",
  "host":"0a228300f282",
  "tag":"s1",
  "sessionViews":11,
  "id":"1t-Yg2rLj4TQWpQyXFRTudDq4NFtbrBx"
}
```


upon each page refresh, except the `id` which changes each time.

5. Clear the cookies on the new browser and repeat these two steps multiple times. What is happening? Are you reaching the node in DRAIN mode?

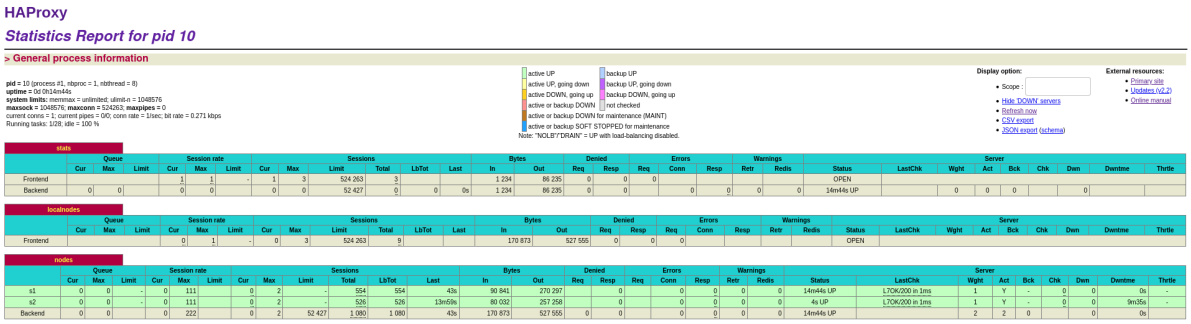
Answer

Still not, no. We always get a similar response (variable `id`).

6. Reset the node in READY mode. Repeat the three previous steps and explain what is happening. Provide a screenshot of HAProxy's stats page.

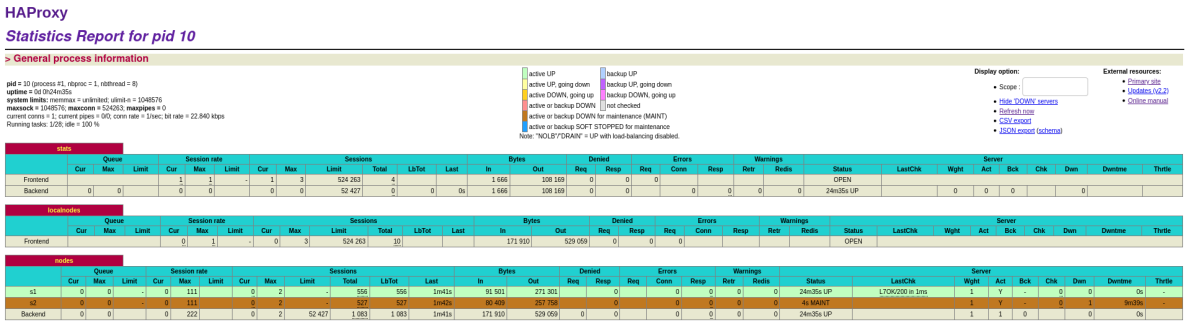
Answer

We now get responses from both nodes in a round-robin fashion, as before.



7. Finally, set the node in MAINT mode. Redo the three same steps and explain what is happening. Provide a screenshot of HAProxy's stats page.

Answer



As for step 4, we only get responses from s1 with variable `id`.

Tâche 4

Remark: Make sure you have the cookies are kept between two requests.

1. Be sure the delay is of 0 milliseconds is set on `s1`. Do a run to have base data to compare with the next experiments.

Note: j'utilise 2 threads et 1000 loops et j'ai set le delay de S2 a 0

Label ↓	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
S2 reached	1000	0	0	5	0.37	0.00%	94.4/sec	0.00	0.00	.0
S1 reached	1000	0	0	2	0.31	0.00%	97.4/sec	0.00	0.00	.0
GET /	2000	8	1	52	15.36	0.00%	185.4/sec	71.51	42.52	395.1
TOTAL	4000	4	0	52	11.71	0.00%	370.7/sec	71.51	42.51	197.5

2. Set a delay of 250 milliseconds on `s1`. Relaunch a run with the JMeter script and explain what it is happening?

Ça prend beaucoup plus de temps a faire le test, du aux 2 users et qu'il faut attendre 250 ms sur S1, on peut le voir dans la colonne Throughput: S2 a une valeur très proche de celle de la base, alors que S1 atteint juste 3.3 par seconde.

Label ↓	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
S2 reached	1000	0	0	5	0.40	0.00%	86.5/sec	0.00	0.00	.0
S1 reached	1000	0	0	1	0.34	0.00%	3.3/sec	0.00	0.00	.0
GET /	2000	155	1	561	157.27	0.00%	6.6/sec	2.54	1.51	395.1
TOTAL	4000	77	0	561	135.68	0.00%	13.2/sec	2.54	1.51	197.5

3. Set a delay of 2500 milliseconds on `s1`. Same than previous step.

On peut voir que le S1 n'est même pas affiché, car jamais atteint.

Label ↓	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
S2 reached	2000	0	0	4	0.34	0.00%	167.6/sec	0.00	0.00	.0
GET /	2000	9	1	53	15.96	0.00%	167.4/sec	64.59	37.91	395.1
TOTAL	4000	4	0	53	12.23	0.00%	334.8/sec	64.58	37.91	197.5

4. In the two previous steps, are there any error? Why?

dans les logs docker compose on peut voir ce message dans la ligne ha: "... Server nodes/s1 is DOWN ... Layer7 timeout ...", cela est du au fait que le délais de 2500 millisecondes est plus grand que le délais de timeout, du coup au yeux du proxy, le serveur s1 est down

```
s2          | GET / 200 0.731 ms - 131
s2          | GET / 200 0.567 ms - 131
ha          | [WARNING] 335/162158 (10) : Server nodes/s1 is DOWN, reason: Layer7 timeout, check duration: 2000ms. 1 active and 0 backup servers left. 1 s
essions active, 0 requeued, 0 remaining in queue.
s2          | GET / 200 1.365 ms - 131
s2          | GET / 200 0.994 ms - 131
s2          | GET / 200 0.959 ms - 131
s2          | GET / 200 1.110 ms - 131
s2          | GET / 200 1.220 ms - 131
s2          | GET / 200 0.647 ms - 131
```

5. Update the HAProxy configuration to add a weight to your nodes. For that, add `weight [1-256]` where the value of weight is between the two values (inclusive). Set `s1` to 2 and `s2` to 1. Redo a run with 250ms delay.

10 utilisateurs avec 100 loops pour gagner du temps (et avoir des chiffres qui se lisent facilement en pour mille), d'abord sans les 250 ms de delay sur S1, On peut voir que dans la répartition est différente: au lieu de la répartitions 50/50 comme dans la partie 4.2 (1000-1000) on a soit : 300-700 ou 600-400 (avoir exactement 333-666 est impossible avec 10 utilisateurs (10 mod 3 n'est pas 0))

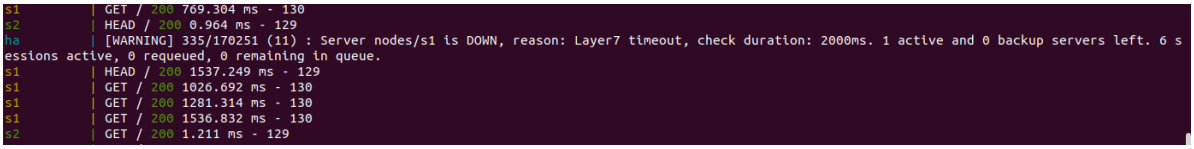
Label ↓	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
S2 reached	300	0	0	15	1.11	0.00%	134.0/sec	0.00	0.00	.0
S1 reached	700	0	0	7	0.42	0.00%	262.1/sec	0.00	0.00	.0
GET /	1000	15	1	55	18.22	0.00%	373.0/sec	144.15	84.45	395.7
TOTAL	2000	7	0	55	14.88	0.00%	745.7/sec	144.10	84.42	197.9

Label ↓	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
S2 reached	400	0	0	8	0.56	0.00%	169.7/sec	0.00	0.00	.0
S1 reached	600	0	0	4	0.42	0.00%	241.0/sec	0.00	0.00	.0
GET /	1000	14	1	63	18.17	0.00%	384.8/sec	148.70	87.34	395.7
TOTAL	2000	7	0	63	14.74	0.00%	768.9/sec	148.59	87.27	197.9

Avec le delay de 250 ms sur S1, l'exemple du dessous est la seule fois ou il a survécu. (je n'ai jamais pu voir l'autre resultat (300-700)).

Label ↓	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
S2 reached	400	0	0	5	0.41	0.00%	169.9/sec	0.00	0.00	.0
S1 reached	600	0	0	5	0.43	0.00%	3.6/sec	0.00	0.00	.0
GET /	1000	985	1	2066	806.43	0.00%	6.1/sec	2.34	1.37	395.7
TOTAL	2000	493	0	2066	753.75	0.00%	12.1/sec	2.34	1.37	197.9

Avec 250 ms de delay un poids de 2 sur S1 et 10, il a tendance a faire un timeout ce qui donne le résultat suivant. La conséquence est que tout est redirigé sur S2 ce qui donne des résultats qui ne s'approche pas du tout des poids 1 pour 2 (33-66) (logique comme S1 est considéré down)

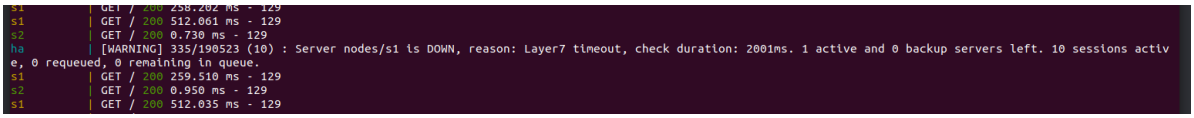


Label ↓	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
S2 reached	929	0	0	5	0.39	0.00%	44.3/sec	0.00	0.00	.0
S1 reached	71	0	0	1	0.37	0.00%	3.7/sec	0.00	0.00	.0
GET /	1000	124	1	1841	410.43	0.00%	47.6/sec	18.45	10.79	396.8
TOTAL	2000	62	0	1841	296.81	0.00%	95.2/sec	18.45	10.79	198.4

6. Now, what happened when the cookies are cleared between each requests and the delay is set to 250ms ? We expect just one or two sentence to summarize your observations of the behavior with/without cookies.

Malheureusement je ne peux pas faire cette partie avec 10 users comme j'ai systématiquement un timeout, même si au lieu de 100 itérations j'en fait seulement 10.

Label ↓	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
S2 reached	950	0	0	7	0.42	0.00%	63.3/sec	0.00	0.00	.0
S1 reached	50	0	0	2	0.40	0.00%	3.7/sec	0.00	0.00	.0
GET /	1000	135	1	3582	553.40	0.00%	66.6/sec	37.43	7.61	575.8
TOTAL	2000	67	0	3582	397.15	0.00%	133.1/sec	37.43	7.61	287.9



par contre avec 2 utilisateurs et 500 loops (pour avoir des pour mille) j'obtiens ça (333-667) (au lieu de 500-500) cela est du au fait que comme les cookies sont effacé à chaque tentative c'est comme si nos 2 users sont en fait 1000 users comme le cookie est leur mémoire et du coup oublie a chaque fois quel serveur leur avait été assigné

Label ↓	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
S2 reached	333	0	0	1	0.29	0.00%	1.8/sec	0.00	0.00	.0
S1 reached	667	0	0	12	0.58	0.00%	3.5/sec	0.00	0.00	.0
GET /	1000	363	1	1018	278.77	0.00%	5.2/sec	2.95	0.60	575.6
TOTAL	2000	181	0	1018	268.12	0.00%	10.5/sec	2.95	0.60	287.8

Tâche 5

In this part of the lab, you will be less guided and you will have more opportunity to play and discover HAProxy. The main goal of this part is to play with various strategies and compare them together.

We propose that you take the time to discover the different strategies in HAProxy documentation and then pick two of them (can be round-robin but will be better to chose two others). Once you have chosen your strategies, you have to play with them (change configuration, use Jmeter script, do some experiments).

Deliverables:

1. Briefly explain the strategies you have chosen and why you have chosen them.

Nous avons choisi les algorithm suivant pour afin de tester le Load-Balancer, cela nous permettra de comparer ces différentes stratégie avec Round-Robin

Stratégie choisi

- Source : Car il s'agit d'un stratégie statique sans cookie
- Leastconn : Car il s'agit d'un stratégie statique

2. Provide evidences that you have played with the two strategies (configuration done, screenshots, ...)

Stratégie Source

La stratégie (statique) de load balancing *Source* utilise l'address ip source et le poids des serveurs cible pour déterminer vers quels application un client devra être renvoyée.

Configuration de HAProxy

On configure la stratégie d'ordonnancement dans `backend` et on définie le nombres de connexions maximal.

```
balance source

server s1 ${WEBAPP_1_IP}:3000
server s2 ${WEBAPP_2_IP}:3000
```

Démonstration du fonctionnement avec Jmeter

Capture avec les poids de base

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	1000	3	2	40	1,99	0,00%	166,0/sec	64,03	35,64	395,0
S1 reached	1000	0	0	5	0,46	0,00%	167,6/sec	0,00	0,00	,0
TOTAL	2000	1	0	40	2,23	0,00%	331,8/sec	64,01	35,63	197,5

Capture avec s2 prioritaire

Modifions le serveur s2 avec un poids plus important

```
server s2 ${WEBAPP_2_IP}:3000 weight 30
```

Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	1000	3	2	32	1,67	0,00%	168,2/sec	64,88	36,12	395,0
S2 reached	1000	0	0	6	0,45	0,00%	170,0/sec	0,00	0,00	,0
TOTAL	2000	1	0	32	2,09	0,00%	336,3/sec	64,87	36,11	197,5

Stratégie leastconn

La stratégie d'ordonnancement *lestconn* prend comme paramètre le nombre de connections établis avec les serveur et va diriger les requêtes vers le serveur le moins chargé.

Configuration de HAProxy

On configure la stratégie d'ordonnancement dans `backend`.meme

```
balance leastconn
```

Démonstration du fonctionnement avec Jmeter

Pour tester la configuration nous allons introduire un délai de traitement entre les deux serveur ce qui forcera la stratégie à diriger les requêtes vers le serveurs avec le moins de délai (qui aura le moins de connexion en cours).

```
curl -H "Content-Type: application/json" -X POST -d '{"delay": 300}'
http://192.168.42.11:3000/delay

curl -H "Content-Type: application/json" -X POST -d '{"delay": 30}'
http://192.168.42.22:3000/delay
```

On observe ainsi que lors du test de charge on se retrouve avec le serveur s2 priorisé, car il possède un meilleur *throughput*.

Libellé	# Echantil... ↓	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	1000	19001	33	76077	27107,66	17,00%	9,6/sec	4,64	1,39	496,7
S2 reached	672	1	0	80	5,31	0,00%	8,0/sec	0,00	0,00	,0
S1 reached	198	0	0	9	1,39	0,00%	1,9/sec	0,00	0,00	,0
TOTAL	1870	10161	0	76077	21972,12	9,09%	17,9/sec	4,64	1,39	265,6

3. Compare the both strategies and conclude which is the best for this lab (not necessary the best at all).

Pour notre laboratoire il se trouve que chacune des stratégies n'est pas parfaitement adapté à notre situation, *Source* étant adapté pour des session sans cookie et *leastconn* n'est pas recommandé pour les session courtes comme HTTP, cependant comme il nous faut choisir nous allons préférer *leastconn* car il s'agit d'un algorithme dynamique (qui peut adapter les poids des nœuds)

Conclusion

Dans ce laboratoire nous avons pu découvrir et expérimenter le fonctionnement du LoadBalancer HAproxy, nous avons pu observer comment les session sont gérée au travers des cookies, et comment est répartie la charge utile en fonction des stratégie d'ordonnancement.