Task 1

1.

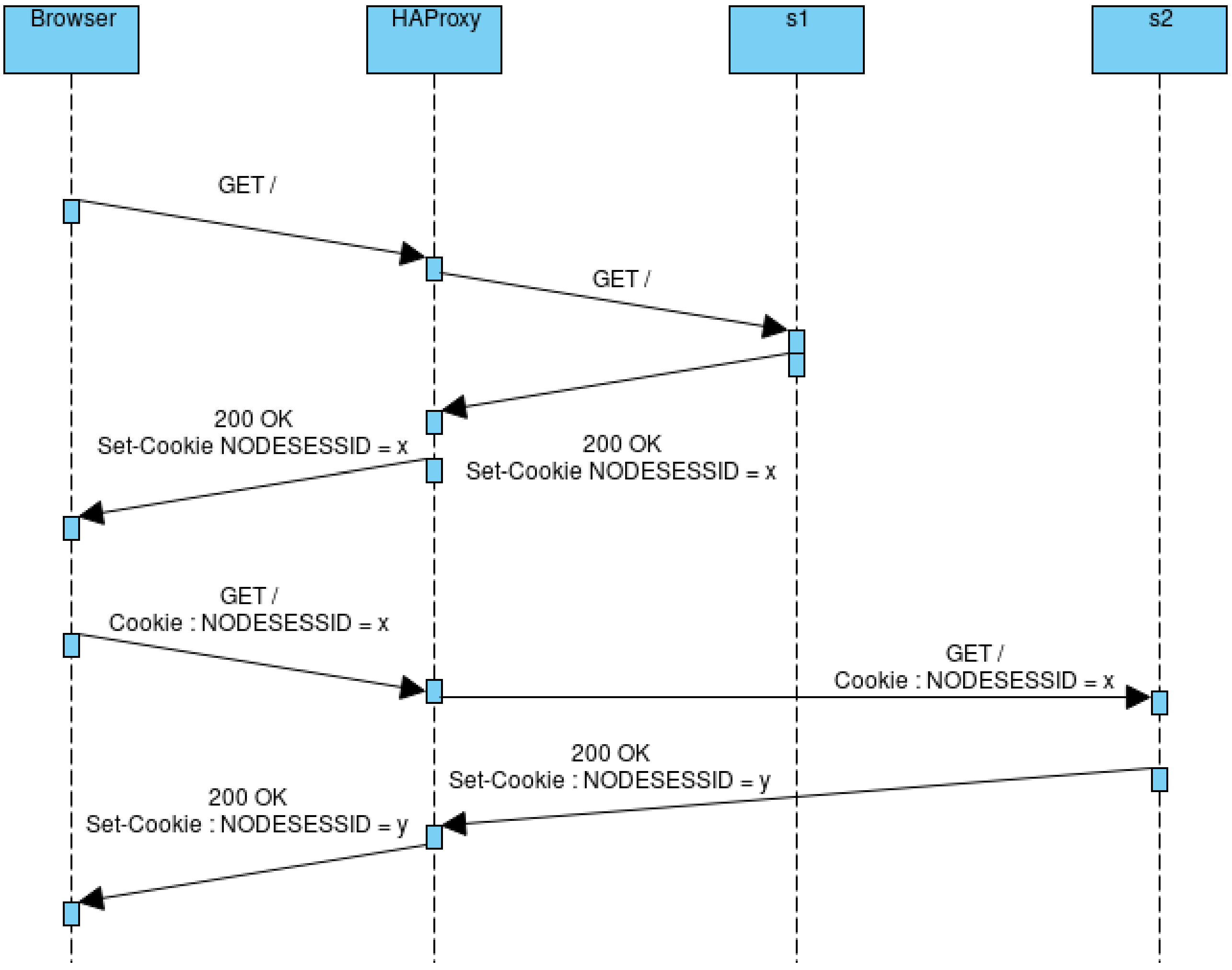
We can see that the proxy handles requests like this : It will address the requests on each server sequentially. It means that one time it will be sent to one server and the next request coming will go threw the other server. The NODESESSID is changing at each request which is not so surpirsing because each time we visit the site or actualize we have an ID which is invalid for the server requested (because the precendent response came from the other server).

2.

We expect the load balancer to address requests coming in with a session ID to the server who knows this session (so the server who created the session). If not, the session management can't be done.

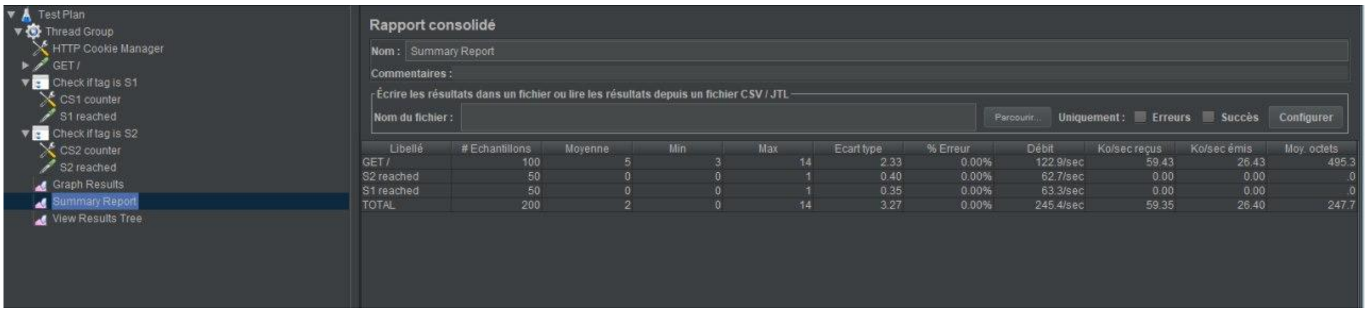
3.

Here is our sequence diagram for task 1 (roundrobin). This sequence diagram shows 2 requests by the same user, we see that the behaviour is not correct because both requests don't go on the same server as expected.



4.

Here is the test in JMeter, we can see that assertions are false because the counters doesn't increment, in fact th server don't know the users coming because of balancing strategy used here.



Rapport consolidé

Nom : Summary Report

Commentaires :

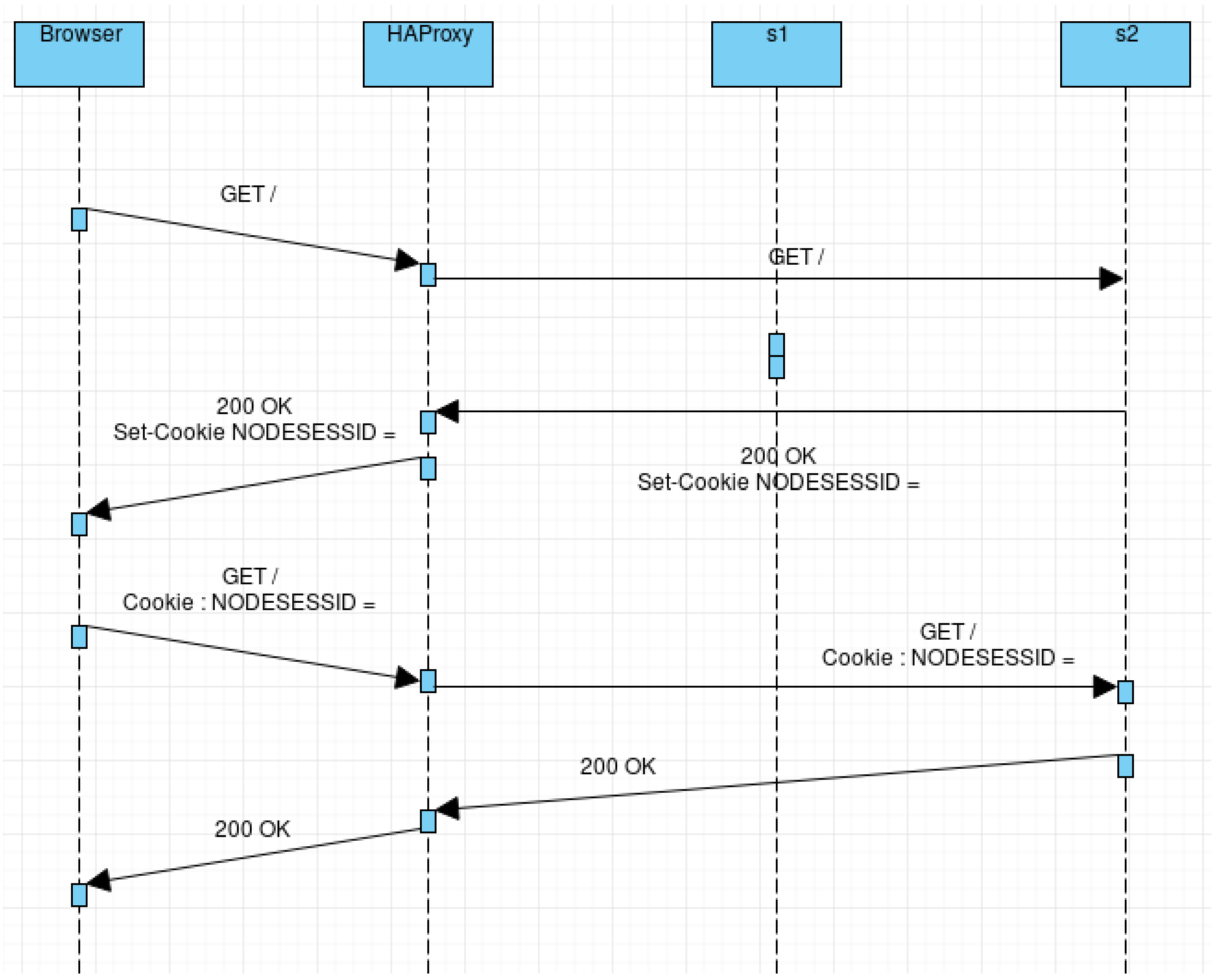
Écrire les résultats dans un fichier ou lire les résultats depuis un fichier CSV / JTL

Nom du fichier :

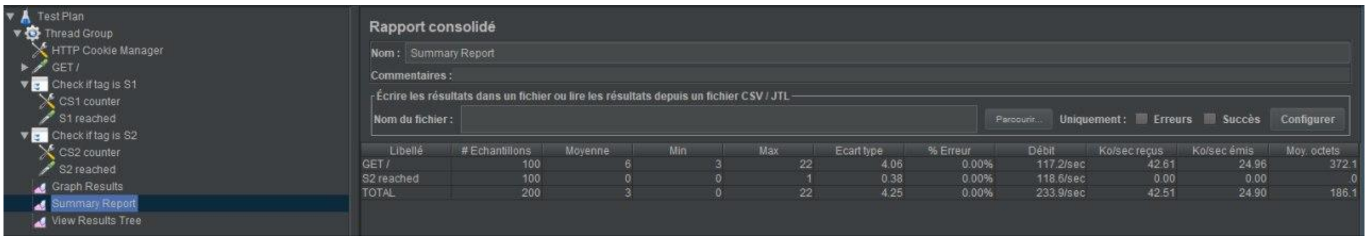
Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	100	5	3	14	2.33	0.00%	122.9/sec	59.43	26.43	495.3
S2 reached	50	0	0	1	0.40	0.00%	62.7/sec	0.00	0.00	.0
S1 reached	50	0	0	1	0.35	0.00%	63.3/sec	0.00	0.00	.0
TOTAL	200	2	0	14	3.27	0.00%	245.4/sec	59.35	26.40	247.7

5.

Here we see the sequence diagram for this task, we can see that because there's only one server running it responds and knows each user (because he serves all the requests), and so the counter increments correctly.



We can see the JMeter report too :



Libellé	# Echantillons	Moyenne	Min	Max	Ecart type	% Erreur	Débit	Ko/sec reçus	Ko/sec émis	Moy. octets
GET /	100	6	3	22	4.06	0.00%	117.2/sec	42.61	24.96	372.1
S2 reached	100	0	0	1	0.38	0.00%	118.6/sec	0.00	0.00	0
TOTAL	200	3	0	22	4.25	0.00%	233.9/sec	42.51	24.90	186.1

Task 2

1.

Sequence Diagram to show differences on stickiness We think that the SERVERID alternative is more effective because the balancer can just evaluate the SEVERID and redirect the request accordingly.

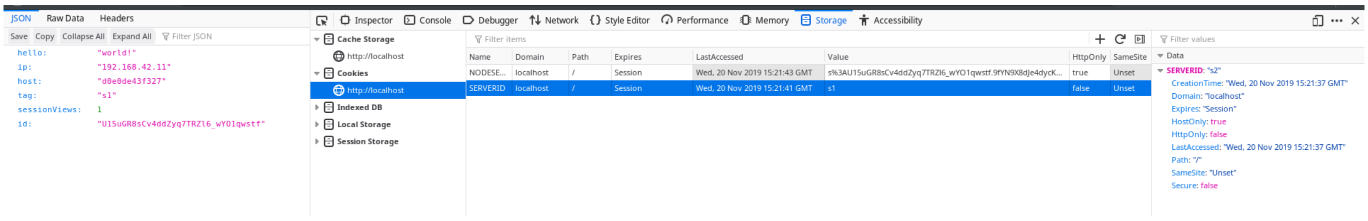
2.

We can see the modified configuration at the root of the git repo : `haproxy_sticky.cfg`. The modifications are juste on the `backend nodes` section. We've just added `cookie SERVERID insert indirect nocache` and the value we want to add on the cookie for each server so :

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

3.

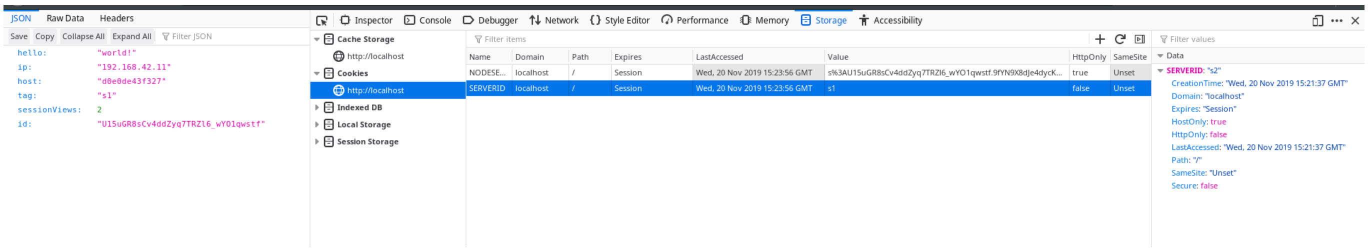
To see if we have achieved the right behaviour we can check like that : First we visit the site and we see that the load balancer crete the SERVERID cookie and that if we refresh the page we keep the same server :



```
hello:         "world!"
ip:            "192.168.42.11"
host:          "d0e0de43f327"
tag:           "s1"
sessionViews:  1
id:            "U15uGR8sCv4ddZyq7TRZl6_wYOlqwstf"
```

Name	Domain	Path	Expires	LastAccessed	Value	HttpOnly	SameSite
NODESE...	localhost	/	Session	Wed, 20 Nov 2019 15:21:43 GMT	s%3AU15uGR8sCv4ddZyq7TRZl6_wYO1qwstf.9fYN9X8dJe4dyck...	true	Unset
SERVERID	localhost	/	Session	Wed, 20 Nov 2019 15:21:41 GMT	s1	false	Unset

```
Data
SERVERID: "s2"
  CreationTime: "Wed, 20 Nov 2019 15:21:37 GMT"
  Domain: "localhost"
  Expires: "Session"
  HostOnly: true
  HttpOnly: false
  LastAccessed: "Wed, 20 Nov 2019 15:21:37 GMT"
  Path: "/"
  SameSite: "Unset"
  Secure: false
```

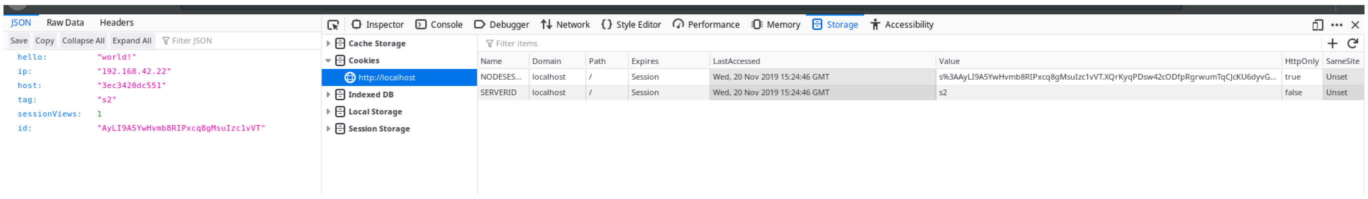


```
hello:         "world!"
ip:            "192.168.42.11"
host:          "d0e0de43f327"
tag:           "s1"
sessionViews:  2
id:            "U15uGR8sCv4ddZyq7TRZl6_wYOlqwstf"
```

Name	Domain	Path	Expires	LastAccessed	Value	HttpOnly	SameSite
NODESE...	localhost	/	Session	Wed, 20 Nov 2019 15:23:56 GMT	s%3AU15uGR8sCv4ddZyq7TRZl6_wYO1qwstf.9fYN9X8dJe4dyck...	true	Unset
SERVERID	localhost	/	Session	Wed, 20 Nov 2019 15:23:56 GMT	s1	false	Unset

```
Data
SERVERID: "s2"
  CreationTime: "Wed, 20 Nov 2019 15:21:37 GMT"
  Domain: "localhost"
  Expires: "Session"
  HostOnly: true
  HttpOnly: false
  LastAccessed: "Wed, 20 Nov 2019 15:21:37 GMT"
  Path: "/"
  SameSite: "Unset"
  Secure: false
```

But if we open an incognito (to remove all cookies) we can see that the roundrobin strategy apply and we have the other server attribued :



```
hello:         "world!"
ip:            "192.168.42.22"
host:          "3ec3420dc551"
tag:           "s2"
sessionViews:  1
id:            "AyLI9A5YwHvmb8RIPxcq8gMsuIzc1vVT"
```

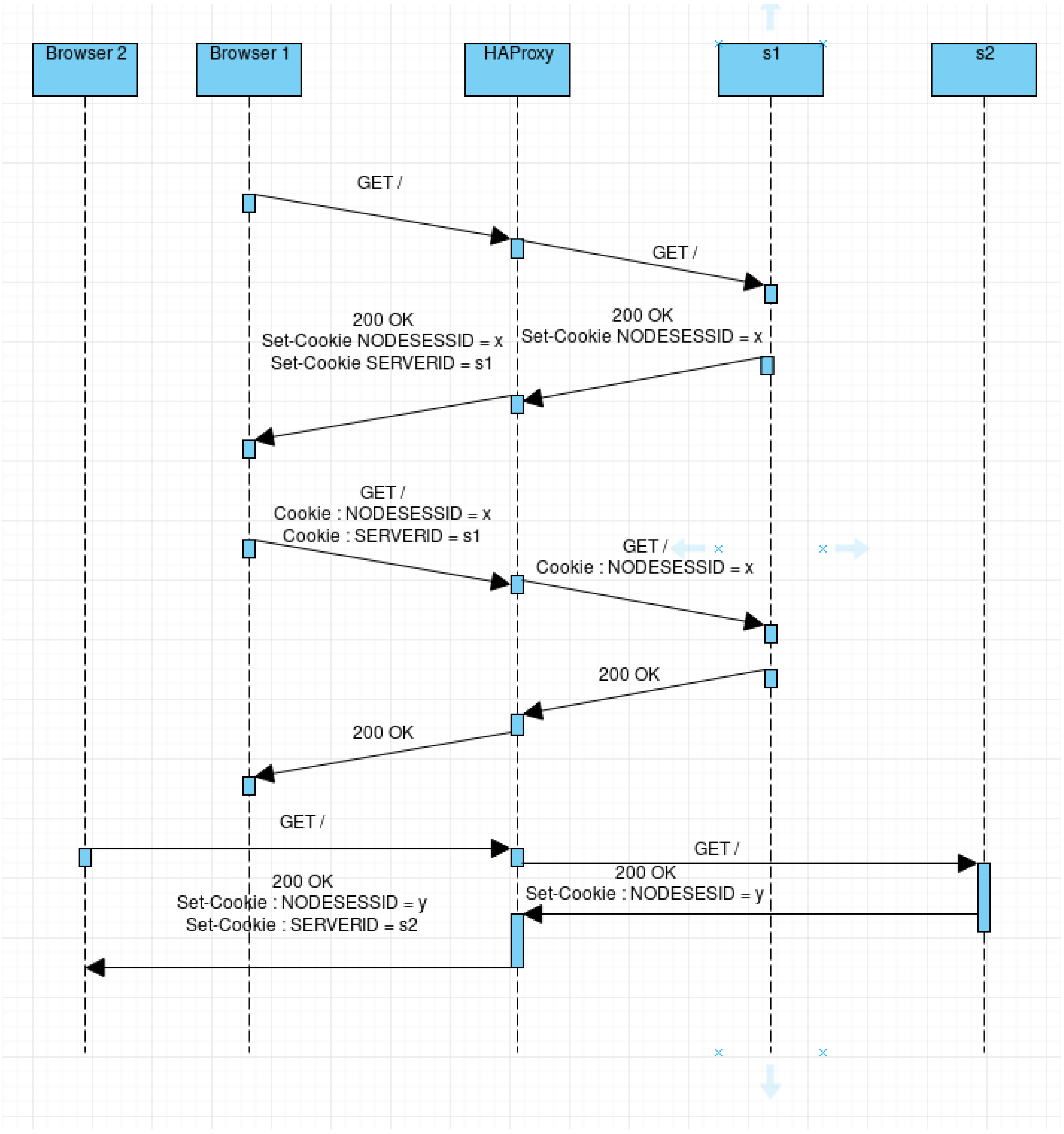
Name	Domain	Path	Expires	LastAccessed	Value	HttpOnly	SameSite
NODESES...	localhost	/	Session	Wed, 20 Nov 2019 15:24:46 GMT	s%3AAyLI9A5YwHvmb8RIPxcq8gMsuIzc1vVT.XQrKyqPDsw42cODfpRgrwumTqCJcKU6dyvG...	true	Unset
SERVERID	localhost	/	Session	Wed, 20 Nov 2019 15:24:46 GMT	s2	false	Unset

And if we refresh this page we keep the same server again because the SERVERID cookie is well set.

sticky session 4

4.

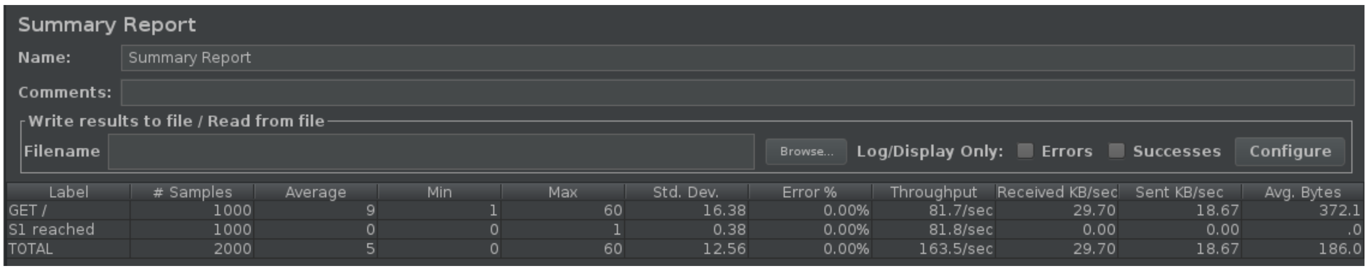
We can see here the diagram of the situation where 1 browser refresh the page and what happens if another browser opens a connection :



We can see that the SERVERID cookie is set at the HAProxy level, and it's removed at this level too. And because of the roundrobin if another browser connects to the infra it will be directed to s2 if s1 was the last to respond. This cookie is used by the HAProxy to redirect correctly to the right server.

5.

Here is the summary report of this task :



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	9	1	60	16.38	0.00%	81.7/sec	29.70	18.67	372.1
S1 reached	1000	0	0	1	0.38	0.00%	81.8/sec	0.00	0.00	.0
TOTAL	2000	5	0	60	12.56	0.00%	163.5/sec	29.70	18.67	186.0

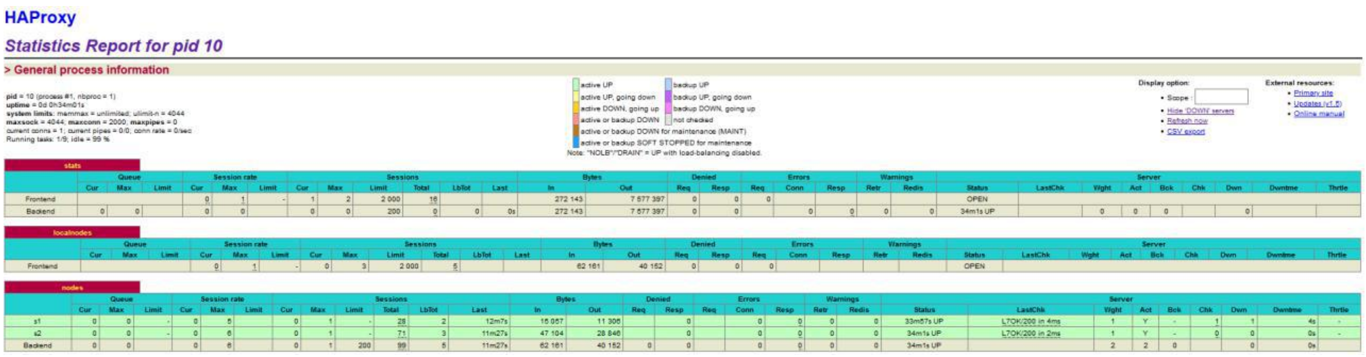
We can explain the behaviour because how JMeter handles the cookies, because there's only one thread group and so one user. So that's normal that we just have one server who responds (because of the SERVERID Cookie).

7.

We can see the result of the manipulations we have done, in fact we added one "user". So the second user when doing his request he's redirected to the other server and all of these future requests will too (because of the cookie).

Task 3

1.



HAProxy

Statistics Report for pid 10

> General process information

pid = 10 (process #1, nbproc = 1)
uptime = 0d 0h34m01s
system limits: memmax = unlimited; ulimit-n = 4044
maxsock = 4044; **maxconn** = 2000; **maxpipes** = 0
current conns = 1; current pipes = 0/0; conn rate = 0/sec
Running tasks: 1/9; idle = 99 %

- 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

External resources:
- Primary site
- Updates (v1.5)
- Online manual

stats

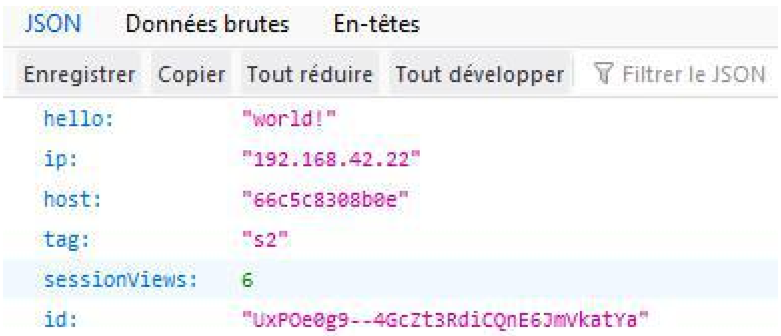
	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	1	2	2 000	16			272 143	7 577 397	0	0	0					OPEN								
Backend	0	0		0	0		0	0	200	0	0	0s	272 143	7 577 397	0	0		0	0	0	0	34m1s 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	3	2 000	5			62 161	40 152	0	0	0					OPEN								

nodes

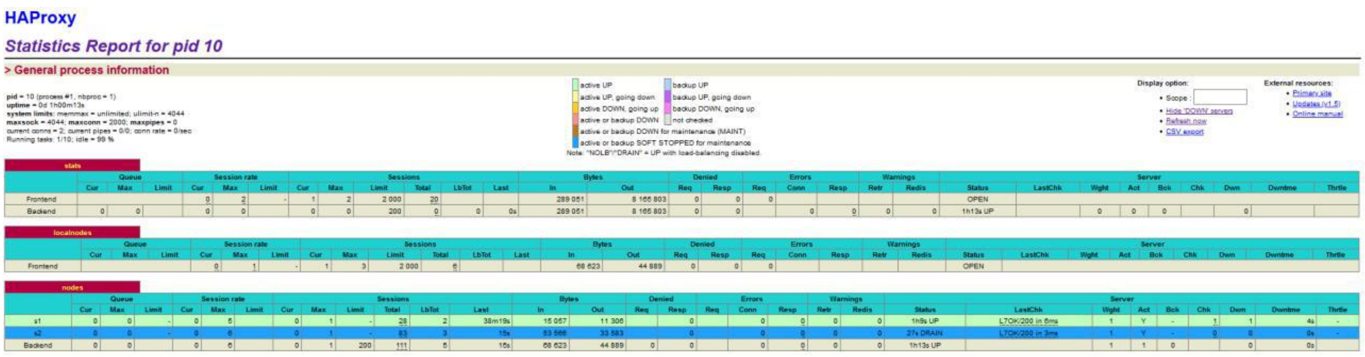
	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
s1	0	0	-	0	5		0	1	-	28	2	12m7s	16 067	11 306		0		0	0	0	0	33m57s UP	L7OK/200 in 4ms	1	Y	-	1	1	4s	-
s2	0	0	-	0	6		0	1	-	71	3	11m27s	47 104	28 846		0		0	0	0	0	34m1s UP	L7OK/200 in 2ms	1	Y	-	0	0	0s	-
Backend	0	0		0	6		0	1	200	99	5	11m27s	62 161	40 152	0	0		0	0	0	0	34m1s UP		2	2	0		0	0s	



```
hello:         "world!"
ip:            "192.168.42.22"
host:          "66c5c8308b0e"
tag:           "s2"
sessionViews:  6
id:            "UxPOe0g9--4GcZt3RdiCQnE6JmVkatYa"
```

We can see in the screenshot above that it is the node "s2" that answers.

2.



HAProxy

Statistics Report for pid 10

> General process information

pid = 10 (process #1, nbproc = 1)
uptime = 0d 1h00m13s
system limits: memmax = unlimited; ulimit-n = 4044
maxsock = 4044; **maxconn** = 2000; **maxpipes** = 0
current conns = 2; current pipes = 0/0; conn rate = 0/sec
Running tasks: 1/10; idle = 99 %

active UP	backup UP
active UP, going down	backup UP, going down
active DOWN, going up	backup DOWN, going up
active or backup DOWN	not checked
active or backup DOWN for maintenance (MAINT)	
active or backup SOFT STOPPED for maintenance	

Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Refresh now
- CSV export

External resources:
- Primary site
- Updates (v1.5)
- 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				0	2	-	1	2	2 000	20			289 051	8 166 803	0	0	0					OPEN								
Backend	0	0		0	0		0	0	200	0	0	0s	289 051	8 166 803	0	0		0	0	0	0	1h13s UP		0	0	0		0		

localnodes

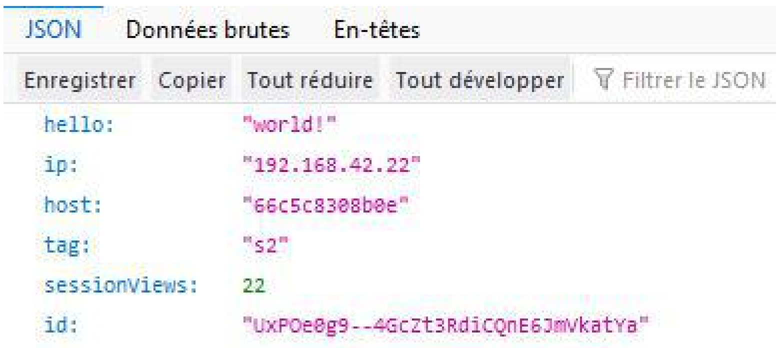
	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	1	3	2 000	6			68 623	44 889	0	0	0					OPEN								

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	5		0	1	-	28	2	38m19s	15 057	11 306		0		0	0	0	0	1h9s UP	L7OK/200 in 6ms	1	Y	-	1	1	4s	-
s2	0	0	-	0	6		0	1	-	83	3	15s	53 566	33 583		0		0	0	0	0	27s DRAIN	L7OK/200 in 3ms	1	Y	-	0	0	0s	-
Backend	0	0		0	6		0	1	200	111	5	16s	68 623	44 889	0	0		0	0	0	0	1h13s UP		1	1	0		0	0s	

We pass the node "s2" on DRAIN mode, it become blue "soft stopped" on the state page.

3.



```
hello:          "world!"
ip:             "192.168.42.22"
host:           "66c5c8308b0e"
tag:            "s2"
sessionViews:   22
id:             "UxPOe0g9--4GcZt3RdiCQnE6JmVkatYa"
```

It's the same node than answer our request. In DRAIN mode, all the new traffic will be redirected to the other nodes, but the current sesssion continue to make request to the node in DRAIN mode.

4.

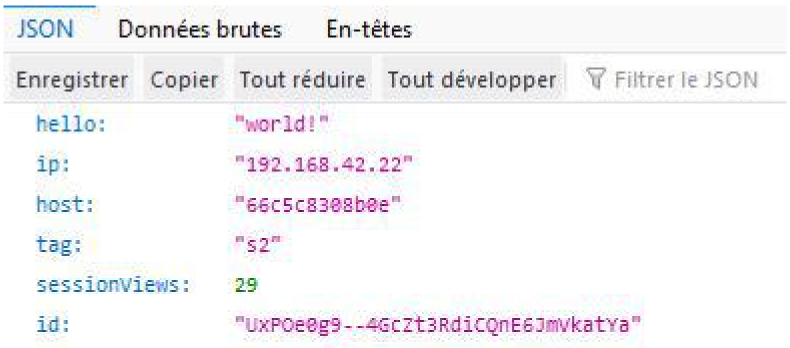
```
{"hello":"world!","ip":"192.168.42.11","host":"d2e8afe714f3","tag":"s1","sessionViews":8,"id":"q9KMi56T2unQk1lK45rSdKZWD186a07Y"}
```

In the new browser , we start a new session and we reach the node "s1".

5.

```
{"hello":"world!","ip":"192.168.42.11","host":"d2e8afe714f3","tag":"s1","sessionViews":1,"id":"L1rfRh7i2LJ4bSqXVbOxQaIrgeuOvFFs"}
```

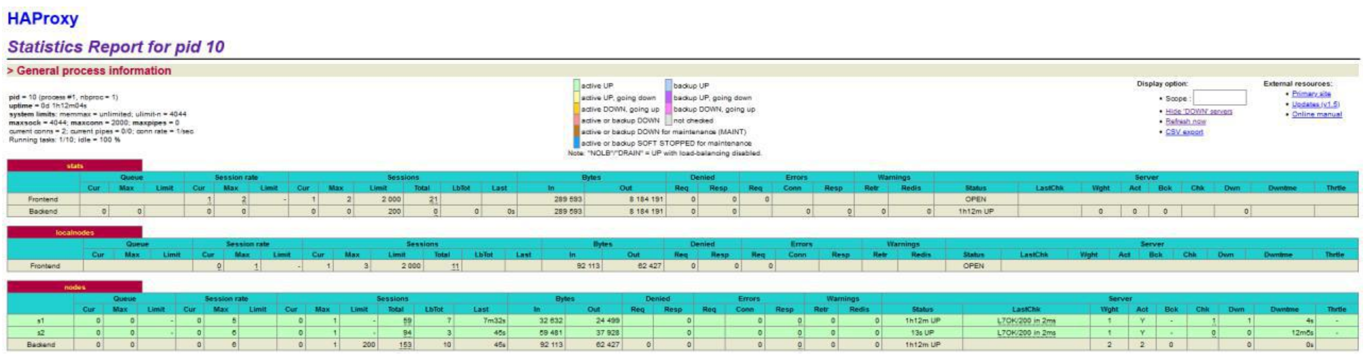
We clear the cookies on the new browser and we still reach the node "s1". Because "s2" is DRAIN mode, the new session can't reach the "s2". The new browser can only reach "s1".



```
hello:          "world!"
ip:             "192.168.42.22"
host:           "66c5c8308b0e"
tag:            "s2"
sessionViews:   29
id:             "UxPOe0g9--4GcZt3RdiCQnE6JmVkatYa"
```

We refresh the first browser and it is still on the node "s2". But if we clear the cookie in the first browser, it will not be able to reach "s2" too.

6.



HAProxy

Statistics Report for pid 10

> General process information

pid = 10 (process #1, nbproc = 1)
uptime = 0d 1h12m04s
system limits: memmax = unlimited; ulimit-n = 4044
maxsock = 4044; **maxconn** = 2000; **maxpipes** = 0
current conns = 2; current pipes = 0/0; conn rate = 1/sec
Running tasks: 1/10; idle = 100 %

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

Display option:
- Scope :
- Hide 'DOWN' servers
- Refresh now
- CSV export

External resources:
- Primary site
- Updates (v1.5)
- 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	2	-	1	2	2 000	21			289 593	8 184 191	0	0	0					OPEN								
Backend	0	0		0	0		0	0	200	0	0	0s	289 593	8 184 191	0	0		0	0	0	0	1h12m 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	-	1	3	2 000	11			92 113	62 427	0	0	0					OPEN								

nodes

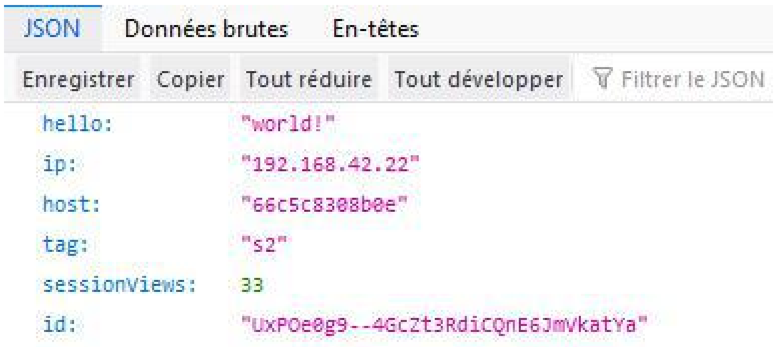
	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
s1	0	0	-	0	5		0	1	-	59	7	7m32s	32 632	24 499		0		0	0	0	0	1h12m UP	L7OK/200 in 2ms	1	Y	-	1	1	4s	-
s2	0	0	-	0	6		0	1	-	94	3	45s	59 481	37 928		0		0	0	0	0	13s UP	L7OK/200 in 2ms	1	Y	-	0	0	12m6s	-
Backend	0	0		0	6		0	1	200	153	10	45s	92 113	62 427	0	0		0	0	0	0	1h12m UP		2	2	0		0	0s	

We pass "s2" on READY mode. On the state page, it becomes like at beginning.

It balanced sequentially between the two nodes.

```
{"hello":"world!","ip":"192.168.42.22","host":"66c5c8308b0e","tag":"s2","sessionViews":4,"id":"UPtIM_ZOo3iSkHT7lso96WwMAn0AvYM0"}
```

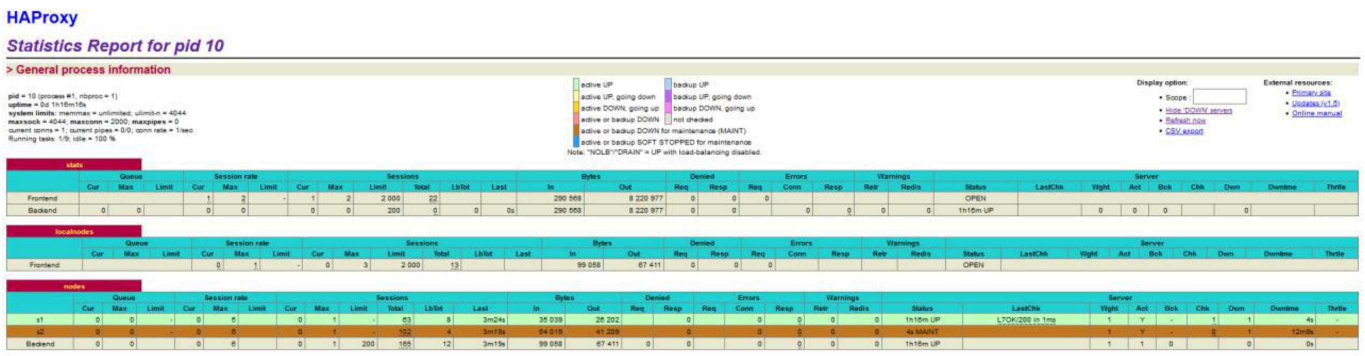
We clear the cookies in the new browser and it reach the node "s2". If we clear again, it will change sequentially.



```
hello:         "world!"
ip:            "192.168.42.22"
host:          "66c5c8308b0e"
tag:           "s2"
sessionViews:  33
id:            "UxPOe0g9--4GcZt3RdiCQnE6JmVkatYa"
```

On the first browser, it still on the node "s2".

7.



HAProxy

Statistics Report for pid 10

> General process information

pid = 10 (process #1, nbproc = 1)
uptime = 0d 1h16m16s
system limits: memmax = unlimited; ulimit-n = 4044
maxsock = 4044; **maxconn** = 2000; **maxpipes** = 0
current conns = 1; current pipes = 0/0; conn rate = 1/sec
Running tasks: 1/9; idle = 100 %

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

Display option:
- Scope :
- Hide 'DOWN' servers
- Refresh now
- CSV export

External resources:
- Primary site
- Updates (v1.5)
- 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	2	-	1	2	2 000	22			290 568	8 220 977	0	0	0					OPEN								
Backend	0	0		0	0		0	0	200	0	0	0s	290 568	8 220 977	0	0		0	0	0	0	1h16m 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	3	2 000	13			99 058	67 411	0	0	0					OPEN								

nodes

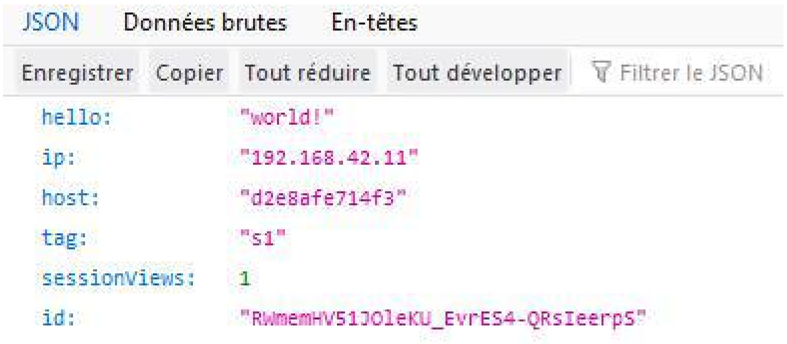
	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
s1	0	0	-	0	6		0	1	-	63	8	3m24s	35 039	26 202		0		0	0	0	0	1h16m UP	L7OK/200 in 1ms	1	Y	-	1	1	4s	-
s2	0	0	-	0	6		0	1	-	102	4	3m19s	64 019	41 209		0		0	0	0	0	4s MAINT		1	Y	-	0	1	12m9s	-
Backend	0	0		0	6		0	1	200	165	12	3m19s	99 058	67 411	0	0		0	0	0	0	1h16m UP		1	1	0		0	0s	

We pass the node "s2" on MAINT mode, it become brown "down" on the state page.

On MAINT mode, all the traffic is redirected to the other nodes, including existing sessions. The node "s2" can't be reached by request.

```
{"hello":"world!","ip":"192.168.42.11","host":"d2e8afe714f3","tag":"s1","sessionViews":1,"id":"yEsiGQ_Uz6Wgf-waB6wiGd7T6u_FmHWW"}
```

The new browser reach the node "s1". If we clear the cookies, we still only reach "s1".



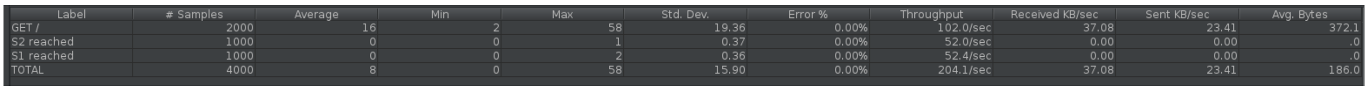
```
hello:          "world!"
ip:             "192.168.42.11"
host:           "d2e8afe714f3"
tag:            "s1"
sessionViews:   1
id:             "RWmemHV51JOleKU_EvrES4-QRsIeerpS"
```

The first browser is not anymore on the node "s2". And has a new session on the node "s1".

Task 4

1.

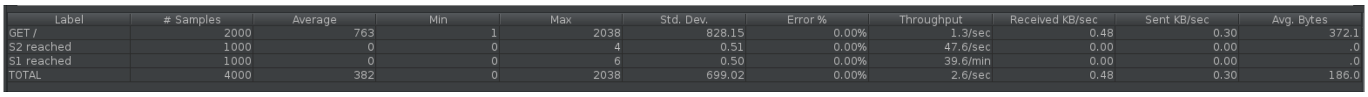
We have been asked to do a run for base data with the base config, so we set the delays to 0 for both servers (we took the JMeter conf. used for Task 2, 2 thread groups):



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	2000	16	2	58	19.36	0.00%	102.0/sec	37.08	23.41	372.1
S2 reached	1000	0	0	1	0.37	0.00%	52.0/sec	0.00	0.00	.0
S1 reached	1000	0	0	2	0.36	0.00%	52.4/sec	0.00	0.00	.0
TOTAL	4000	8	0	58	15.90	0.00%	204.1/sec	37.08	23.41	186.0

2.

We set a 250ms delay to s1 :

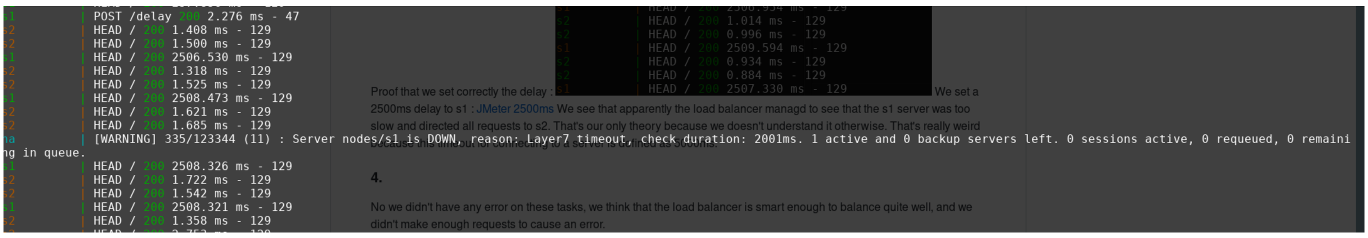


Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	2000	763	1	2038	828.15	0.00%	1.3/sec	0.48	0.30	372.1
S2 reached	1000	0	0	4	0.51	0.00%	47.6/sec	0.00	0.00	.0
S1 reached	1000	0	0	6	0.50	0.00%	39.6/min	0.00	0.00	.0
TOTAL	4000	382	0	2038	699.02	0.00%	2.6/sec	0.48	0.30	186.0

We can see with this run taht it took a really long time (25min) because 1000*0.25s, but it works well in fact. What happens it's that the first user takes the connection to s1 and it will be long because the cookie specify s1 for each request. And the second user go to s2 without problems and takes each time s2 because of the cookie.

3.

Proof that we set correctly the delay :



```
s1          | POST /delay 200 2.276 ms - 47
s2          | HEAD / 200 1.408 ms - 129
s2          | HEAD / 200 1.500 ms - 129
s1          | HEAD / 200 2506.530 ms - 129
s2          | HEAD / 200 1.318 ms - 129
s2          | HEAD / 200 1.525 ms - 129
s1          | HEAD / 200 2508.473 ms - 129
s2          | HEAD / 200 1.621 ms - 129
s2          | HEAD / 200 1.685 ms - 129
ha          | [WARNING] 335/123344 (11) : Server nodes/s1 is DOWN, reason: Layer7 timeout, check duration: 2001ms. 1 active and 0 backup servers left. 0 sessions active, 0 requeued, 0 remaini
ng in queue.
s1          | HEAD / 200 2508.326 ms - 129
s2          | HEAD / 200 1.722 ms - 129
s2          | HEAD / 200 1.542 ms - 129
s1          | HEAD / 200 2508.321 ms - 129
s2          | HEAD / 200 1.358 ms - 129
```

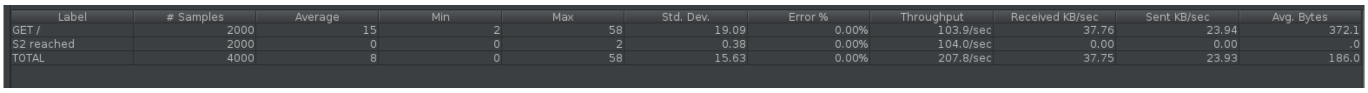
```
s2          | HEAD / 200 1.014 ms - 129
s2          | HEAD / 200 0.996 ms - 129
s1          | HEAD / 200 2509.594 ms - 129
s2          | HEAD / 200 0.934 ms - 129
s2          | HEAD / 200 0.884 ms - 129
s1          | HEAD / 200 2507.330 ms - 129
```

Proof that we set correctly the delay : We set a 2500ms delay to s1 : JMeter 2500ms We see that apparently the load balancer managd to see that the s1 server was too slow and directed all requests to s2. That's our only theory because we doesn't understand it otherwise. That's really weird because this timeout for connecting to a server is defined as 5000ms.

4.

No we didn't have any error on these tasks, we think that the load balancer is smart enough to balance quite well, and we didn't make enough requests to cause an error.

We can see on the proof that HA detects s1 as DOWN. We set a 2500ms delay to s1 :



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	2000	15	2	58	19.09	0.00%	103.9/sec	37.76	23.94	372.1
S2 reached	2000	0	0	2	0.38	0.00%	104.0/sec	0.00	0.00	.0
TOTAL	4000	8	0	58	15.63	0.00%	207.8/sec	37.75	23.93	186.0

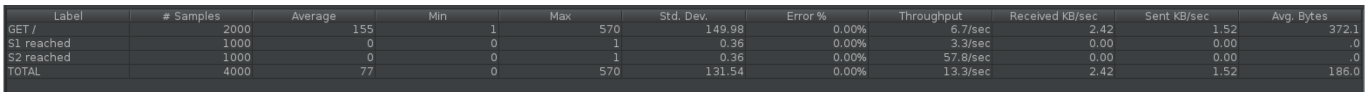
We see that apparently the load balancer managd to see that the s1 server was too slow and directed all requests to s2 (see proof above). But that's really weird because this timeout for connecting to a server is defined as 5000ms.

4.

No we didn't have any error on these tasks, we think that the load balancer is smart enough to balance quite well, and we didn't make enough requests to cause an error.

5.

After doing the weight config we have this behaviour in the JMeter tests :

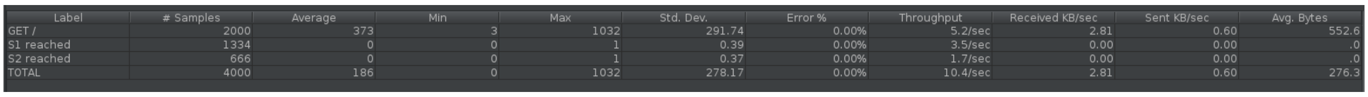


Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	2000	155	1	570	149.98	0.00%	6.7/sec	2.42	1.52	372.1
S1 reached	1000	0	0	1	0.36	0.00%	3.3/sec	0.00	0.00	.0
S2 reached	1000	0	0	1	0.36	0.00%	57.8/sec	0.00	0.00	.0
TOTAL	4000	77	0	570	131.54	0.00%	13.3/sec	2.42	1.52	186.0

Nothing change from the 2. apart of the time taken. That's because the weights doesn't change anything for 2 users.

6.

Now the weights takes effects because it's like we have another user each time, because we clean cookies for each iteration. So the weights influence the behaviour of the load balancer in this way :



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
GET /	2000	373	3	1032	291.74	0.00%	5.2/sec	2.81	0.60	552.6
S1 reached	1334	0	0	1	0.39	0.00%	3.5/sec	0.00	0.00	.0
S2 reached	666	0	0	1	0.37	0.00%	1.7/sec	0.00	0.00	.0
TOTAL	4000	186	0	1032	278.17	0.00%	10.4/sec	2.81	0.60	276.3