Lab 3: Load balancing

Besseau Léonard, Gamboni Fiona, Pellissier David  
18.11.2021

# Table of Contents

[Introduction 1](#_Toc89769198)

[Task 1: Install the tools 1](#_Toc89769199)

[Task 2: Sticky sessions 3](#_Toc89769200)

[Task 3: Drain mode 7](#_Toc89769201)

[Task 4: Round robin in degraded mode. 9](#_Toc89769202)

[Task 5: Balancing strategies 11](#_Toc89769203)

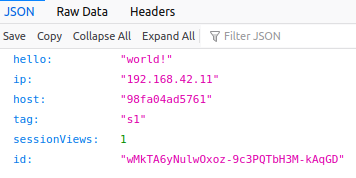
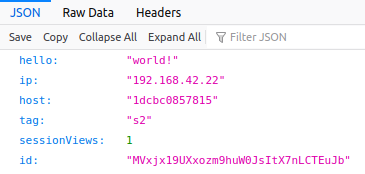
[Conclusion 14](#_Toc89769204)

# Introduction

The purpose of this lab is to experiment load balancing in a given environment.   
We started with an analysis of the load balancer default behaviour, especially on session management. Then we tested the configuration of sticky sessions with and without the Drain mode of HAProxy. We also tested the degraded mode of the round-robin strategy. Lastly, we tested and compared two different balancing strategies.

# Task 1: Install the tools

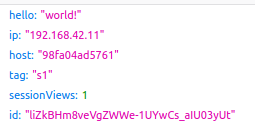
1. Explain how the load balancer behaves when you open and refresh the URL<http://192.168.42.42> in your browser. Add screenshots to complement your explanations. We expect that you take a deeper a look at session management

The load balancer redistributes each request in a round-robin way.

﻿ first request second request

The screens above show the response of two successive requests. We can see that we alternate between the two servers: they both have a different IP and ID.

If we don’t have a valid cookie (or not at all), a new session is created. However, the load balancer is misconfigured because when we receive a cookie from a server, we don’t keep the communication with this server as the round-robin policy continues. Thus, the server returns a new set cookie and we can never keep our session.

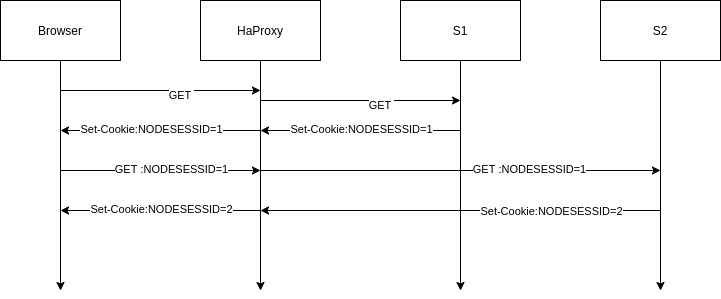
To test this, we tried sending a request with the same cookie 4 times and our session count for the server s2 increased up to 3 but we created 2 new sessions with the server s1.

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

The load balancer should redirect the request with a session cookie to the correct server to keep the session alive. To fix that, sticky-session should be implemented.

1. 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.

In the first request, the first server sets a session cookie. In the second, the request (with the cookie) is sent to the second server. As it doesn’t recognize the session, the second server sets a new cookie with the response.

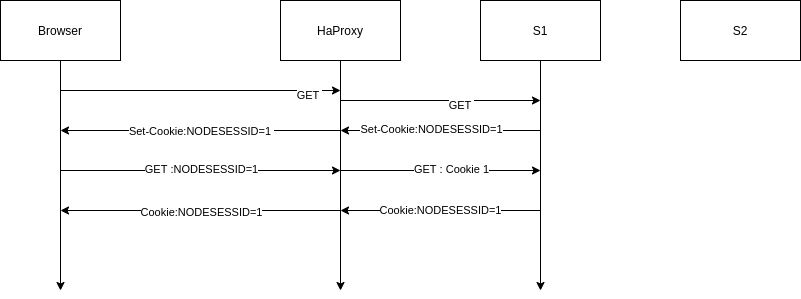
﻿

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

Image

1. Explain what is happening when only one node remains active. Provide another sequence diagram using the same model as the previous one.

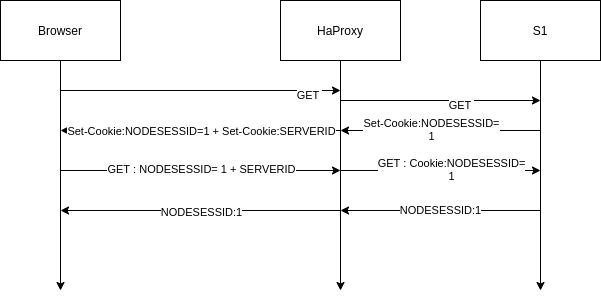
The load-balancer sends the requests exclusively to the active node, so the session is working as intended.

﻿

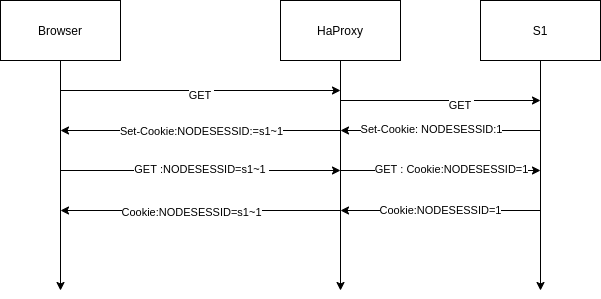
# Task 2: Sticky sessions

1. There are different ways 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).

The SERVERID method works by adding a new cookie when the response from the server traverses the reverse-proxy. When the client makes a new request, it will send the new cookie along to the reverse proxy. The proxy can then use the cookie to choose which server to send the request to.



The NODESESSID method works by modifying the session cookie to insert the id of the origin server to the cookie. The reverse-proxy will then read the cookie in the next request, extract the server information and send the original cookie along the request to the server specified in the cookie.



* 1. Choose one of the both stickiness approach for the next tasks

SERVERID

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

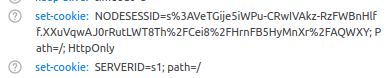
cookie SERVERID insert indirect nocache  
server s1 ${WEBAPP\_1\_IP}:3000 check cookie s1  
server s2 ${WEBAPP\_2\_IP}:3000 check cookie s2

The first line indicates to haproxy to add a new cookie if the user did not already have it. It is also recommended to not cache the cookie as a session is supposed to be personal.

The two other lines provide the value of the cookie based on the server.

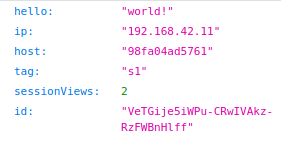
1. Explain what is the behavior when you open and refresh the URL<http://192.168.42.42> in your browser. Add screenshots to complement your explanations. We expect that you take a deeper look at session management.

First request: We get the two cookies. This is the first time this session is viewed:





Second request: We send the two cookies and we get the same server as before. The sessionViews increases to 2:

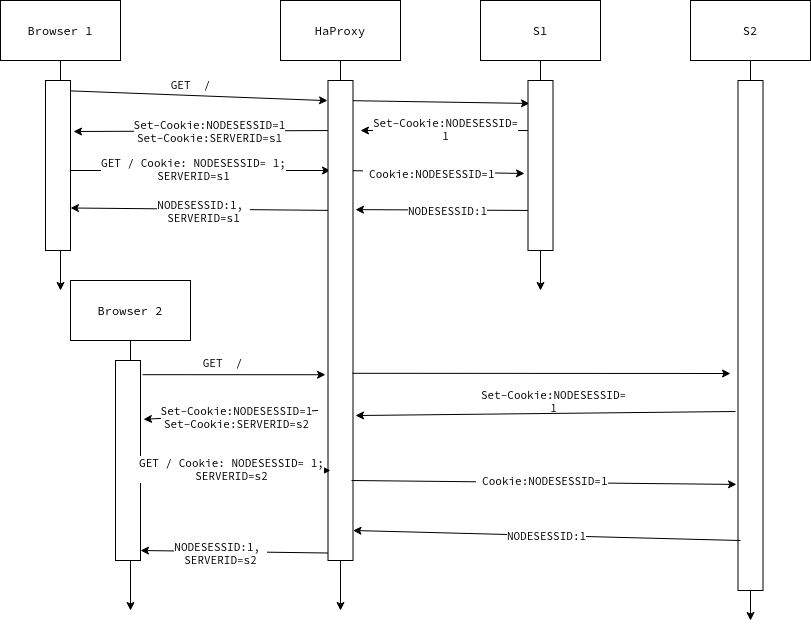


The two cookies were sent to the server:

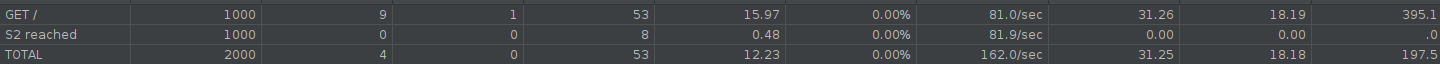


This changes from the previous part when we were always changing servers and thus setting a new cookie session each time. With the stickiness, we can keep the session for each request.

1. 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.

Browser 2 gets the second server attributed to him due to the proxy using round-robin.

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



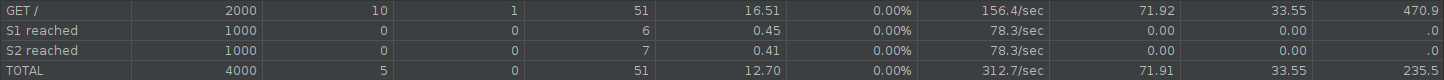
The difference is that only one server is reached due to the session stickiness.

* 1. Clear the results in JMeter.
  2. Now, update the JMeter script. Go in the HTTP Cookie Manager and verify that the box Clear cookies each iteration? is unchecked.

It is already unchecked.

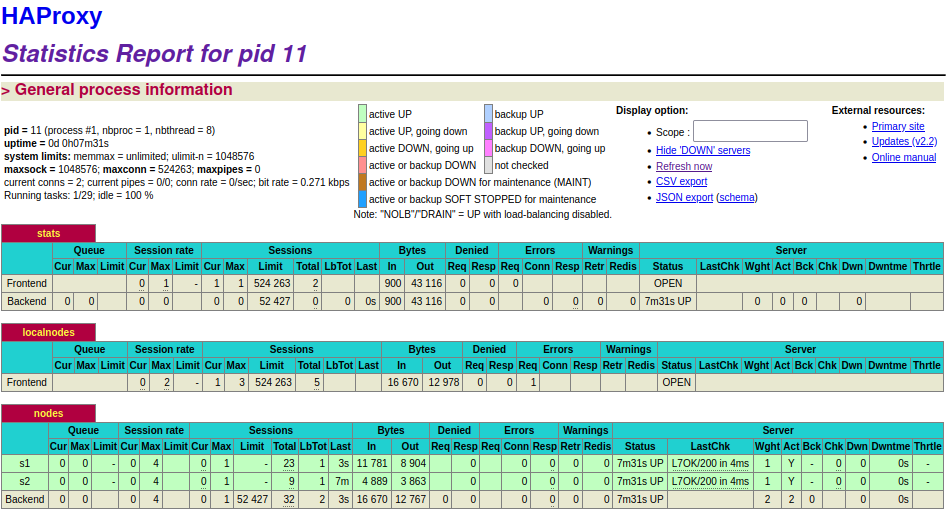
* 1. Go in Thread Group and update the Number of threads. Set the value to 2.

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

  
Each thread was assigned to a server and due to the session stickiness, kept that server for all subsequent requests.

# Task 3: Drain mode

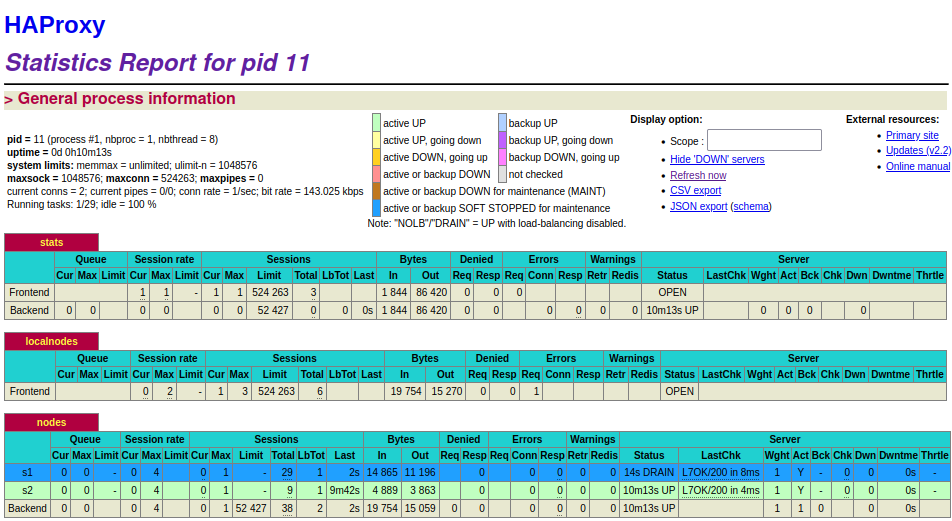
1. Take a screenshot of the Step 5 and tell us which node is answering.



The node answering is S1 as its last message was just sent. (3s)

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

set server nodes/s1 state drain



1. Refresh your browser and explain what is happening. Tell us if you stay on the same node or not. If yes, why? If not, why?

We stay on the same node. It’s the expected behaviour as there is an already established connection.

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

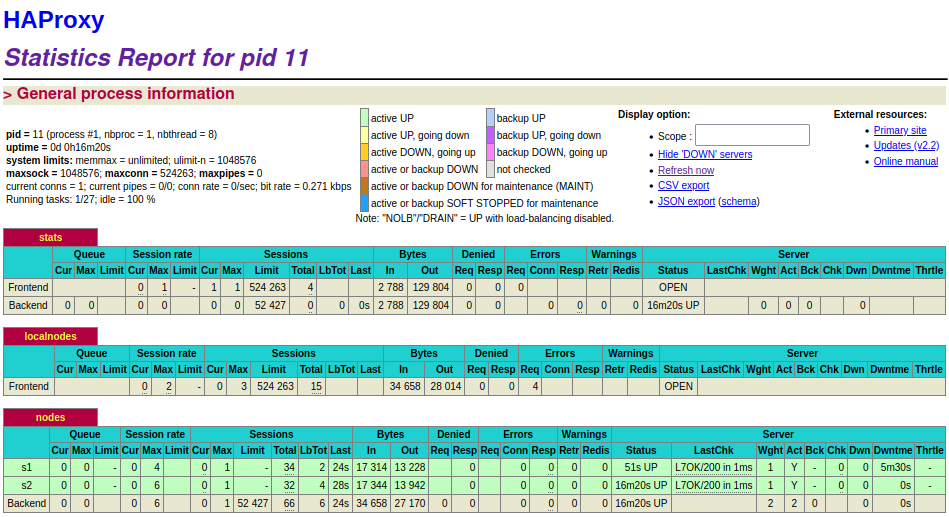
The new connections are redirected to the other server.

1. 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?

The new connections are still redirected to the other server. The reverse proxy directs all new traffic to s2. S1 can only be reached by an already existing connection.

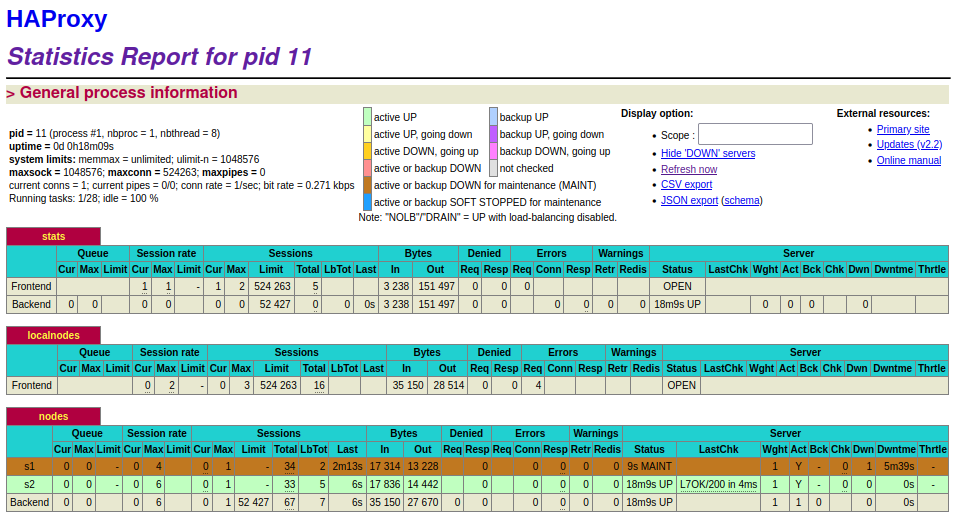
1. Reset the node in READY mode. Repeat the three previous steps and explain what is happening. Provide a screenshot of HAProxy's stats page.

set server nodes/s1 state ready



This time, the new connections are balanced between the two servers. Existing connections are still redirected to the same server.

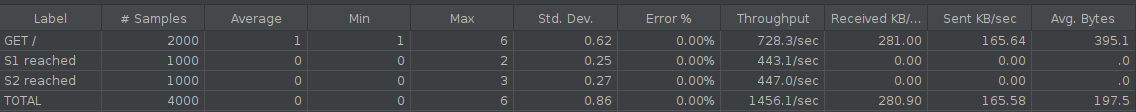
1. Finally, set the node in MAINT mode. Redo the three same steps and explain what is happening. Provide a screenshot of HAProxy's stats page.



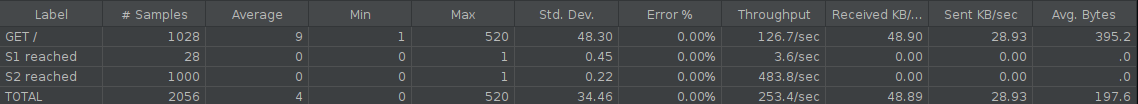
This time, we cannot reach s1 even with an existing connection. When we try, we are redirected to s2 and given a new cookie. For the new connections, all traffic goes to s2.

# Task 4: Round robin in degraded mode.

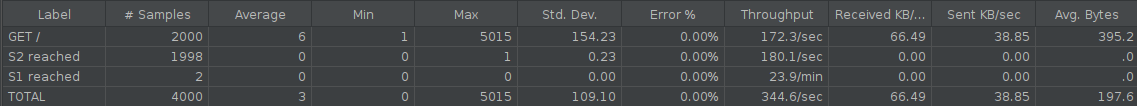
1. Make sure a delay of 0 milliseconds is set on s1. Do a run to have a baseline to compare with in the next experiments.



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

  
We can see that while S2’s throughput increases a little, S1’s throughput drops to 3.6 requests per seconds.

1. Set a delay of 2500 milliseconds on s1. Same as the previous step.

  
S1 can only be reached two times before the reverse proxy considers the server is down and redirects all requests to S2. The delay before declaring S1 down is the reason for the decreased throughput of S2.

1. In the two previous steps, are there any errors? Why?

There is no error for step 2 but there is one for step 3:



As we assumed previously, S1 was considered down by HAProxy due to the very long delay we’ve set up.

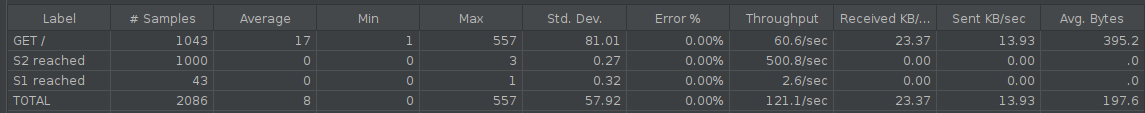
1. Update the HAProxy configuration to add a weight to your nodes. For that, add weight [1-256] where the value of weight is between the two values (inclusive). Set s1 to 2 and s2 to 1. Redo a run with a 250ms delay.

Config:

server s1 ${WEBAPP\_1\_IP}:3000 check cookie s1 weight 2  
server s2 ${WEBAPP\_2\_IP}:3000 check cookie s2 weight 1

We can see that the weights are rightly set:

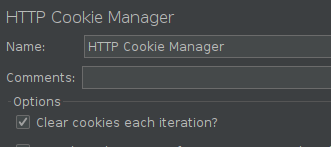
S1:   
S2:﻿



Even though the S1 has more weight, due to the session the weight has no effect as there are only two threads.

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

To clear or keep the cookies between requests:

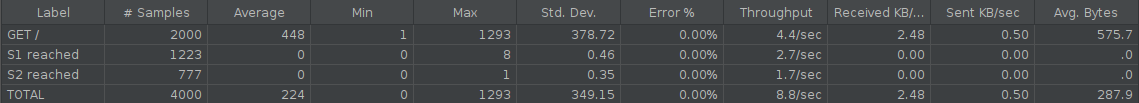


* With cookies

There are only two sessions and with the sticky cookies, the reverse-proxy always sends the same thread to the same server, so the weight has no effects.

* Without cookies

Due to the weight, S1 will receive twice the number of requests from S2. Due to this, S2 throughput will be reduced as fewer connections will be forwarded to him (due to the threads having to wait for S1 to send the request).



# Task 5: Balancing strategies

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

**first**: Each server in the list of servers should have a “maxcon” set, meaning it will only accept a certain number of connection slots. The first server with available connection slots receives the connection.

**leastconn**: the chosen server is the one having the less active connections at the moment of the first request.

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

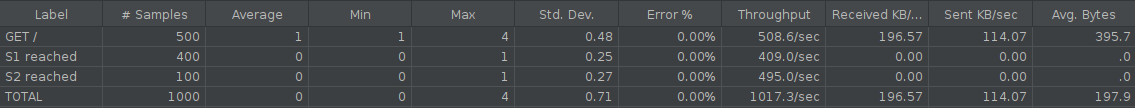
**first:**

Config:

balance first  
 server s1 ${WEBAPP\_1\_IP}:3000 check cookie s1 maxconn 3  
 server s2 ${WEBAPP\_2\_IP}:3000 check cookie s2 maxconn 3

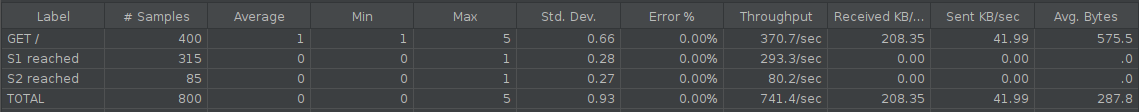
Testing:

Without clearing cookies

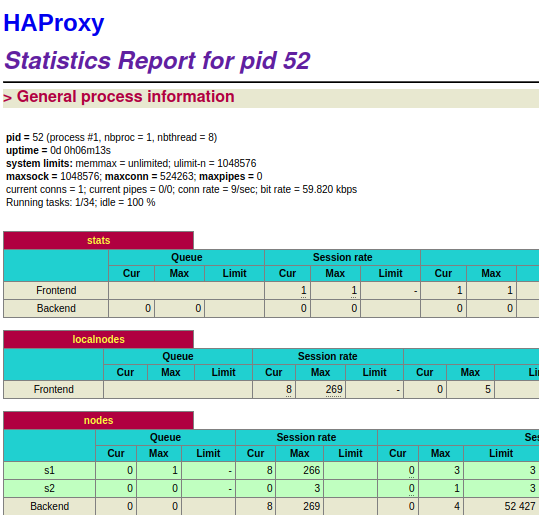


We needed to use 5 threads here to make sure to use that s1 had all connections used to be able to connect with s2.

With clearing cookies:



Here we can see that 10% of the time, we connect with s2 and otherwise stick with s1.



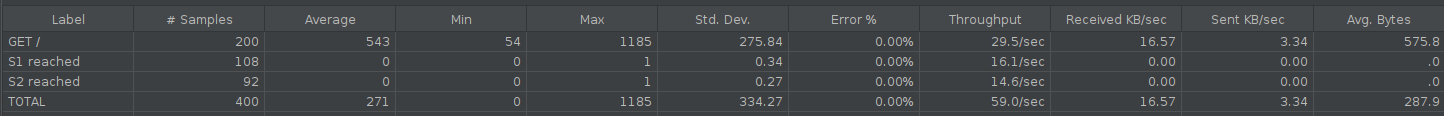
Config view to check that there are only 3 connections at maximum.

**leastconn:**

Config

#define the balancing policy  
 balance leastconn

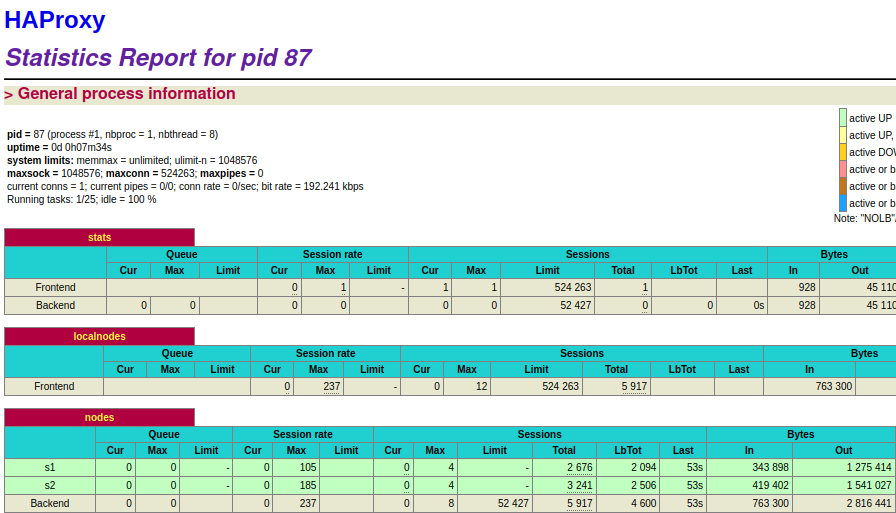
Testing:



Test with 20 threads with 50 ms of delay with session clearing.



Test with 20 threads with 50 ms of delay without session clearing. We can see that the round robin is enabled.



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

leastconn:

* good when working long sessions
* ensure that all servers will be used
* consider the number of queued connections in addition to the established ones in order to minimize queuing

first:

* good when working long sessions
* always use the smallest number of servers so that extra servers can be powered off during non-intensive hours

For this lab, leastconn is the better strategy, as we don’t plan on shutting down servers when they are not used. Leastconn assures a good distribution between the servers.

# Conclusion

This lab allowed us to get a better understanding of load-balancing and its strategies, and the impact on performance it can have. It also allowed us to get to know and experiment with concepts such as the sticky-session drain mode or round-robin degraded mode.