How To Use HAProxy to Set Up MySQL Load Balancing

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: http://faq.asphosthelpdesk.com/article.php?id=189

Prelude

HAProxy is an open source software which can load balance HTTP and TCP servers. In the previous article on HAProxy we configured load balancing for HTTP and in this one we'll do the same for MySQL. All your MySQL servers have to be configured to perform Master-Master replication as load balancing involves both reading and writing to all the backends.

The following three panels will be used in this article:

panel 1 - Load Balancer
Hostname: haproxy
OS: Ubuntu
Private IP: 10.0.0.100

panel 2 - Node 1
Hostname: mysql-1
OS: Debian 7
Private IP: 10.0.0.1

panel 3 - Node 2
Hostname: mysql-2
OS: Debian 7
Private IP: 10.0.0.2

Before proceeding, make sure all MySQL servers are up, running and are properly replicating database writes.

Prepare MySQL Servers

We need to prepare the MySQL servers by creating two additional users for HAProxy. The first user will be used by HAProxy to check the status of a server.

```
root@mysql-1# mysql -u root -p -e "INSERT INTO mysql.user (Host,User) values
('10.0.0.100','haproxy_check'); FLUSH PRIVILEGES;"
```

A MySQL user is needed with root privileges when accessing the MySQL cluster from HAProxy. The default root user on all the servers are allowed to login only locally. While this can be fixed by granting additional privileges to the root user, it is better to have a separate user with root privileges.

```
root@mysql-1# mysql -u root -p -e "GRANT ALL PRIVILEGES ON *.* TO
'haproxy_root'@'10.0.0.100' IDENTIFIED BY 'password' WITH GRANT OPTION; FLUSH
PRIVILEGES"
```

Replace haproxy_root and password with your own secure values. It is enough to execute these queries on one MySQL master as changes will replicate to others.

Install MySQL Client

MySQL client has to be installed on the HAProxy to test connectivity.

```
root@haproxy# apt-get install mysql-client
```

Now try executing a query on one of the masters as the haproxy_root user.

```
root@haproxy# mysql -h 10.0.0.1 -u haproxy_root -p -e "SHOW DATABASES"
```

This should display a list of MySQL databases.

Installing HAProxy

On the HAProxy server install the package.

```
root@haproxy# apt-get install haproxy
```

Enable HAProxy to be started by the init script.

```
root@haproxy# sed -i "s/ENABLED=0/ENABLED=1/" /etc/default/haproxy
```

To check if this change is done properly execute the init script of HAProxy without any parameters.

```
root@haproxy:~# service haproxy
Usage: /etc/init.d/haproxy {start|stop|reload|restart|status}
```

Configuring HAProxy

Rename the original configuration file

```
mv /etc/haproxy/haproxy.cfg{,.original}
```

Create and edit a new one

```
nano /etc/haproxy/haproxy.cfg
```

The first block is the global and defaults configuration block.

```
global
    log 127.0.0.1 local0 notice
    user haproxy
    group haproxy
defaults
    log global
    retries 2
    timeout connect 3000
    timeout server 5000
    timeout client 5000
```

More information about each of these options are covered in this article. Since we've told HAProxy to send log messages to 127.0.0.1 we have to configure rsyslog to listen on it. This has too been covered in the same article under Configure Logging for HAProxy.

Moving to the main configuration part.

```
listen mysql-cluster
    bind 127.0.0.1:3306
    mode tcp
    option mysql-check user haproxy_check
    balance roundrobin
    server mysql-1 10.0.0.1:3306 check
    server mysql-2 10.0.0.2:3306 check
```

Unlike HTTP load balancing HAProxy doesn't have a specific "mode" for MySQL so we use tcp. We've set HAProxy to listen only on the loopback address (assuming that application is on the same server) however if your application resides on a different panel make it listen on 0.0.0.0 or the private IP address.

We need one more configuration block to see the statistics of load balancing. This is completely optional and can be omitted if you don't want stats.

```
listen 0.0.0.0:8080
    mode http
    stats enable
    stats uri /
    stats realm Strictly\ Private
    stats auth A_Username:YourPassword
    stats auth Another_User:passwd
```

Replace the usernames and passwords in "stats auth". This will make HAProxy listen on port 8080 for HTTP requests and the statistics will be protected with HTTP Basic Authentication. So you can access stats at

```
http://<Public IP of Load Balancer>:8080/
```

Once you're done configuring start the HAProxy service.

```
service haproxy start
```

Use the mysql client to query HAProxy.

```
root@haproxy# mysql -h 127.0.0.1 -u haproxy_root -p -e "SHOW DATABASES"
```

The "-h" option has to be present with the loopback IP address. Omitting it or using localhost will make the MySQL client connect to the mysql.sock file which will fail.

Testing Load Balancing and Failover

To check if load balancing is working query the **server_id** variable twice or more.

```
root@haproxy# mysql -h 127.0.0.1 -u haproxy_root -p -e "show variables like
```

```
'server_id'"
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 1     |
+---------------+-------+
root@haproxy# mysql -h 127.0.0.1 -u haproxy_root -p -e "show variables like
'server_id'"
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 2     |
+---------------+-------+
```

This demonstrates roundrobin load balancing with equal weights, we'll now change the weight for mysql-2 and see the results.

```
nano /etc/haproxy/haproxy.cfg

server mysql-2 10.0.0.2:3306 check weight 2
```

Reload to apply this change.

```
service haproxy reload
```

Query for the server_id multiple times.

```
root@haproxy:~# for i in `seq 1 6`
do
mysql -h 127.0.0.1 -u haproxy_root -ppassword -e "show variables like
'server_id'"
done
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 1     |
+---------------+-------+
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 2     |
+---------------+-------+
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 2     |
+---------------+-------+
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 1     |
+---------------+-------+
+---------------+-------+
```

```
| Variable_name | Value |
+---------------+-------+
| server_id     | 2     |
+---------------+-------+
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 2     |
+---------------+-------+
```

Now load balancing works in the ratio of 1:2 with one-thirds of the requests going to mysql-1 and two-thirds going to mysql-2.

Fail a MySQL server either by stopping the service

```
root@mysql-1# service mysql stop
```

or bringing the interface down.

```
root@mysql-1# ifconfig eth1 down
```

Try the "show variables" query now to see the result. The following log entries will indicate when and how HAProxy detected the failure.

```
tail /var/log/haproxy/haproxy.log

Nov 15 00:08:51 localhost haproxy[1671]: Server mysql-cluster/mysql-1 is DOWN,
reason: Layer4 timeout, check duration: 2002ms. 1 active and 0 backup servers
left. 0 sessions active, 0 requeued, 0 remaining in queue.
```

Reducing Failover Interval

When a MySQL server goes down HAProxy takes some time to detect this failure and remove it from the cluster. In this section we'll see how to control this time. First we'll see how to measure this value. One way is to block the MySQL port using iptables for a certain amount of time, then remove the rule and check the log.

```
root@mysql-1:~# ifconfig eth1 down &&
date &&
sleep 20 &&
ifconfig eth1 up &&
date
Fri Nov 15 00:37:09 IST 2013
Fri Nov 15 00:37:29 IST 2013
```

The port 3306 was blocked for 20 seconds, we'll look at the log file now.

```
root@haproxy:~# tail /var/log/haproxy.log
Nov 15 16:49:38 localhost haproxy[1275]: Server mysql-cluster/mysql-1 is DOWN,
reason: Layer4 connection problem, info: "Connection refused", check duration:
0ms. 1 active and 0 backup servers left. 0 sessions active, 0 requeued, 0
```

```
remaining in queue.
Nov 15 16:49:56 localhost haproxy[1275]: Server mysql-cluster/mysql-1 is UP,
reason: Layer7 check passed, code: 0, info: "5.5.31-0+wheezy1-log", check
duration: 1ms. 2 active and 0 backup servers online. 0 sessions requeued, 0
total in queue.
```

It took 6 seconds to detect a failure (difference between 16:49:38 and 16:49:32) and 4 seconds_to detect that the server can be reached (difference between 16:49:56 and 16:49:52). This is determined by the server parameters rise, fall and inter.

The rise parameter sets the number of checks a server must pass to be declared operational. Default is 2.

The fall parameter sets the number of checks a server must pass to be declared dead. Default is 3.

The inter parameter sets the interval between these checks. Default is 2000 milliseconds.

Putting this info together a server must fail 3 continuous checks which are performed at an interval of 2 seconds to be considered dead. So in our example above the following would've happened.

```
16:49:32 - Port 3306 on mysql-1 was blocked
16:49:34 - Check - Failed - Failure No. 1
16:49:36 - Check - Failed - Failure No. 2
16:49:38 - Check - Failed - Failure No. 3 (server removed and event logged)
```

And when the firewall rule was removed.

```
16:49:52 - Firewall rule removed port 3306 accessible
16:49:54 - Check - Passed - Success No. 1
16:49:56 - Check - Passed - Success No. 2 (server added to cluster and event
logged)
```

The following settings will reduce the test interval to 1 second and also reduce the number of fall tests.

```
nano /etc/haproxy/haproxy.cfg

server mysql-1 10.0.0.1:3306 check fall 2 inter 1000
server mysql-2 10.0.0.2:3306 check fall 2 inter 1000
```

Sometimes you may not want to flood the private network with too many "test" packets especialy if you have a large amount of MySQL servers. In such cases the fastinter and downinter parameters will come handy.

The fastinter parameter sets the interval between checks while a server is transitioning UP or DOWN.

The downinter parameter sets the test interval when a server is DOWN.

That explanation might be confusing so we'll see it with an example.

```
nano /etc/haproxy/haproxy.cfg

server mysql-1 10.0.0.1:3306 check fastinter 1000
server mysql-2 10.0.0.2:3306 check fastinter 1000
```

Since we haven't specified the "inter" parameter it defaults to 2000ms. With this configuration we'll restart HAProxy and do the test again.

```
root@mysql-1:~# iptables -A INPUT -p tcp --dport 3306 -j REJECT &&
date &&
sleep 20 &&
iptables -D INPUT -p tcp --dport 3306 -j REJECT &&
date
Fri Nov 15 17:18:48 IST 2013
Fri Nov 15 17:19:08 IST 2013
```

Check the HAProxy log file.

```
root@haproxy:~# tail /var/log/haproxy.log
Nov 15 17:18:52 localhost haproxy[1353]: Server mysql-cluster/mysql-1 is DOWN,
reason: Layer4 connection problem, info: "Connection refused", check duration:
0ms. 1 active and 0 backup servers left. 0 sessions active, 0 requeued, 0
remaining in queue.
Nov 15 17:19:11 localhost haproxy[1353]: Server mysql-cluster/mysql-1 is UP,
reason: Layer7 check passed, code: 0, info: "5.5.31-0+wheezy1-log", check
duration: 1ms. 2 active and 0 backup servers online. 0 sessions requeued, 0
total in queue.
```

Now it took only 4 seconds (compared to 6 earlier) to detect a failure and 3 seconds (compared to 4) to detect that the server was up. Behind the scenes this is what happened.

```
17:18:48 - Port 3306 blocked
17:18:50 - Check - Failed - Failure No. 1
17:18:51 - Check - Failed - Failure No. 2
17:18:52 - Check - Failed - Failure No. 3 (server removed and event logged)
```

And when the port was unblocked.

```
17:19:08 - Firewall rule removed
17:19:10 - Check - Passed - Success No. 1
17:19:11 - Check - Passed - Success No. 2 (server added to cluster and event
logged)
```

First notice the interval between the port block event (17:18:48) and the first check (17:18:50), it is 2 seconds (the "inter" interval). Then notice the interval between Test 1 <-> Test 2 and Test 2 <-> Test 3, it is only 1 second (the "fastinter" interval). The same intervals can be noticed when the server moved from DOWN to UP. So "fastinter" controls the interval between these checks.

So what is downinter? When a server has been declared DOWN HAProxy continues checking it every 2 seconds (or the interval mentioned in inter). If you feel you're using up unnecessary network resources setting the downinter to say 5000 will make HAProxy check a DOWN server only once in 5 seconds.

Important

The tests we did previously REJECTed the packets which means when HAProxy initiated a connection by sending a SYN packet to **mysql-1** it received a RST packet (instead of SYN + ACK). This is why the log

entry mentioned "Connection refused". In this case only the fall, inter and fastinter values come into the scene.

Instead if HAProxy didn't receive anything after sending SYN the connection times out. In this case in addition to the above mentioned parameters the "timeout" duration comes into the scene. This situation can happen if

- the **iptables** is set to **DROP** the incoming
- there is a problem with the private networking infrastructure