Load balancing MySQL with HaProxy

Peter Boros
Consultant @ Percona
4/23/13
Santa Clara, CA
Agenda

- What is HaProxy
- HaProxy configuration
- Load balancing topologies
- Checks
- Load balancing Percona XtraDB Cluster
- Load balancing master-slave Cluster managed by PRM
- Load balancing MySQL Cluster
- Writing custom checks, large scale issues
What is HaProxy

- http://haproxy.1wt.eu
- General purpose load balancer
  - We are using it at the TCP level
  - It doesn't understand the mysql wire protocol
  - It can check backend state on HTTP
  - Current stable version is 1.4
- Single process, event-driven
HaProxy configuration

- Global
  - Logging, user, group, ...
- Defaults
  - Mode, maxconn, timeouts, ...
- Frontend definitions
  - Port, default backed ...
- Backend definitions
  - Backend servers, check method, check interval ...
- Listen
  - Frontend + backend
HaProxy configuration II.

- **Global**
  - global
  - log 127.0.0.1 local0
  - maxconn 4096
  - chroot /usr/share/haproxy
  - user haproxy
  - group haproxy
  - daemon

- **Defaults**
  - option redispatch
  - maxconn 2000
  - contimeout 5000
  - clitimeout 50000
  - srvtimeout 50000
HaProxy configuration III.

- **Frontend**
  
  ```
  frontend stats-front
  bind *:80
  mode http
  default_backend stats-back
  ```

- **Backend**
  
  ```
  mode http
  balance roundrobin
  stats uri /haproxy/stats
  stats auth pxcstats:secret
  ```
Deployment scenarios
Separate layer

- 2xRTT in application response time
- Scaling: new load balancer pairs for new application servers
- On MySQL, it will look like every connection is coming from the load balancers
On the application

- No additional latency
- Load balancing layer scales with application layer
- A lot of checks will happen on the databases (each loadbalancer does its own checks)
- Connections from application servers
On the database

- No additional latency
- The load balancer consumer resources on the database server
- VIP management is still needed
- The databases will see that the connections are coming from the local IP
HaProxy performance

• Session rate
  – Determines if the load balancer can distribute all requests it receives

• Session concurrency
  – Related to the session rate, the slower the server, the higher the concurrency

• Data rate
  – Measured in MB/s, highest throughput comes with large sessions
### Status page sample

<table>
<thead>
<tr>
<th>Queue</th>
<th>Session rate</th>
<th>Sessions</th>
<th>Bytes</th>
<th>Denied</th>
<th>Errors</th>
<th>Warnings</th>
<th>Status</th>
<th>LastChk</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>0 0 - 0 9</td>
<td>8 8 - 9 9</td>
<td>112</td>
<td>1615</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18m1s UP</td>
<td>L70K/200 in 109ms</td>
</tr>
<tr>
<td>c2</td>
<td>0 0 - 0 0</td>
<td>0 0 - 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18m1s UP</td>
<td>L70K/200 in 69ms</td>
</tr>
<tr>
<td>c3</td>
<td>0 0 - 0 0</td>
<td>0 0 - 0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18m1s UP</td>
<td>L70K/200 in 15ms</td>
</tr>
<tr>
<td>Backend</td>
<td>0 0 - 0 9</td>
<td>8 8 - 9 9</td>
<td>112</td>
<td>1615</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18m1s UP</td>
<td></td>
</tr>
</tbody>
</table>

### Diagram

- **pxc-onenode-back**
- **Queue**
- **Session rate**
- **Sessions**
- **Bytes**
- **Denied**
- **Errors**
- **Warnings**
- **Status**
- **LastChk**
- **Server**
- **Wght Act Bck Chk**
- **Dwn Dvmtme**
- **Thrtle**
Checks

server c1 10.116.39.76:3306 check port 9200 inter 1500 rise 3 fall 3

• TCP port 9200 is checked on HTTP
  - Response code 200 -> backend up
  - Response code 500 -> backend down
  - We need something which turns database state into HTTP response codes
Checks II.

- Clustercheck supplied with PXC
  - Shell script ran through xinetd
  - Simple and maintainable, 1 check can mean several forks
- With many application servers
  - Daemon, which “caches” results for the check period
  - Using non-blocking IO
Load balancing PXC

- **PXC architecture**
  - N (typically 3) active nodes
- **Writes and reads can go to all nodes**
- **Checks dependent on wsrep variables**

server c1 10.116.39.76:3306 check port 9200 inter 1500 rise 3 fall 3
server c2 10.195.206.117:3306 check port 9200 inter 1500 rise 3 fall 3 backup
server c3 10.202.23.92:3306 check port 9200 inter 1500 rise 3 fall 3 backup
Load balancing PRM

- PRM architecture
  - One master, n slave
- Separate backend definition for the master and slave
- Check for read_only=0 flag for master
- Check if the node is managed by pacemaker
Load balancing MySQL Cluster

- MySQL Cluster architecture
  - All mysql servers are active
- One backend definition containing all the servers
- Check: if mysql is up and ndbcluster is registered as a storage engine, server is up
Performance vs Parallelism

From: mysqlperformanceblog.com
Additional Features

- **Queueing**
  - `maxqueue >> maxconn`
  - Only `maxconn` TCP connections can be there
  - The rest of the connections will be queued on TCP/IP level

- **Ramp-up (slowstart Xs)**
  - A server is not getting the full traffic when it comes online
  - The server's weight is adjusted dynamically
Persistent connections

- HaProxy decides where the connection will go at TCP handshake
- Once the TCP session is established, sessions will stay where they are
- Be cautious with persistent connections
  - Configuring connection pool properly
  - Important parameters are minimum, maximum connections and connection lifetime
High traffic issues

• Show themselves later if haproxy is deployed on each application server.

• Not specific to haproxy
  – Limitations in MySQL client-server protocol
  – Linux TCP/IP implementation

http://blog.exceliance.fr/2012/12/12/haproxy-high-mysql-request-rate-and-tcp-source-port-exhaustion/
MySQL client-server communication

- This means that the connection at the TCP level can be in TIME_WAIT state in minutes.
- Leads to source ip:port paris exhaustion.
Possible solutions

- Using more than one IP address
- Tune max local port range
  - /proc/sys/net/ipv4/ip_local_port_range
- Tune allow tw_recycle, tw_reuse
  - /proc/sys/net/ipv4/tcp_tw_recycle
  - /proc/sys/net/ipv4/tcp_twReuse
- Allow the kernel to kill the connections
  - /proc/sys/net/ipv4/tcp_max_tw_buckets
  - This is very high by default
Possible solutions II.

- Haproxy's nolinger option
  - Needs 1.4
  - Connections in \texttt{TIME\_WAIT} and \texttt{FIN\_WAIT1}
- “Nolinger” patch for glb by Frederic Descamps
  - Using \texttt{SO\_LINGER} in \texttt{setsockopt()} helps if this happens at the proxy level
  - But not if a real client is connecting and disconnecting to MySQL too fast
Server closing TCP connection

- If the server closed the connection, the issue would not be there.
Thanks for attention.