MySQL Load Balancers - MaxScale, ProxySQL, HAProx, MySQL Router & nginx - a close up look

October 05, 2016

Krzysztof Książek

Severalnines

krzysztof@severalnines.com
Agenda

- Why load balancers?
- Types of load balancers
  - Reverse proxies
  - SQL-aware proxies
  - Application connectors
- Health checks used with proxies
Why load balancers?

- High availability
  - Detect node health
  - Route traffic away from failed nodes

- Performance
  - Scale out by routing traffic to multiple nodes
  - Minimize number of connections to databases

- Keep complexity away from the application
  - Monitor node state
  - Understand replication topology
  - Failover client connections

- Additional features
  - Collect performance data from cluster
  - Rewrite queries as they pass the proxy
  - Route queries depending on some rules (e.g. Read/write split)
Types of load balancers
Types of load balancers

- Reverse proxies
  - HAProxy
  - Keepalived
  - NGINX

- SQL-aware proxies
  - MaxScale
  - MySQL Router
  - ProxySQL

- Application connectors
  - mysqlnd_ms
  - MySQL Connector/J
Types of load balancers - reverse proxies

- Designed to route packets - nothing more, nothing less
  - Work (with MySQL) on transport layer (L4)
  - Fast and efficient in moving data around
- Health checks options are limited
  - Typically no MySQL support
  - Checks based on port being available
- Do not understand MySQL states
  - master, slave
  - SYNCED, DONOR/DESYNC, JOINING, JOINED
- Primary, Non-Primary
- Not really suitable for work with MySQL unless you extend them with external checks
Types of load balancers - SQL-aware load balancers

- Dedicated for MySQL
  - Understand MySQL protocol
  - Understand MySQL states
- Improved high availability thanks to:
  - Monitoring the health of backend MySQL nodes
  - Monitoring metrics of backend MySQL nodes
  - Monitoring of replication topology

- Advanced features like:
  - Read/write split
  - Query rewriting
  - Advanced query routing
  - Support for sharding
  - Binary log servers
  - Query caching
Types of load balancers - Application connectors

- Keep database layer complexity away from the application (to some extent)
  - Implement read/write split transparently to the application
  - More advanced features will require interaction with an application
- Improve availability of the database tier
  - Automated failover for slave nodes
  - Automated failover in master/standby pair
- May be more compelling to developers and DevOps staff
  - Configuration happens on web hosts
  - No need for additional layer of proxies
Reverse proxies
Reverse proxies - HAProxy

- Industry standard - thousands of installations across the world
- 1.6 released in October, 2015
- 1.6.8 released on August, 14th 2016
- LUA integration, improved support for server connection multiplexing, much faster and cheaper HTTP compression via libels

- Reverse proxy
  - Load balancing, scaling out, failover
  - Lack of advanced internal support for MySQL backends
Reverse proxies - HAProxy

- Lack of ability to perform read/write split on a single port
- Two backends on two different ports are required
- Could be a blocker if an application can’t split reads from writes

- No understanding of MySQL state
  - Only simple check on port 3306: handshake or authentication
  - Workaround exists: clustercheck
  - HTTP check, xinetd, external script to monitor MySQL/Galera states
Reverse proxies - HAProxy

```
listen haproxy_172.30.4.215_3307
  bind *:3307
  mode tcp
  timeout client 10000s
  timeout server 10000s
  balance leastconn
  option httpchk
  option allbackups
  default-server port 9200 inter 2s downinter 5s rise 3 fall 2 slowstart 60s maxconn 64 maxqueue 128 weight 100
  server 172.30.4.238 172.30.4.238:3306 check
  server 172.30.4.184 172.30.4.184:3306 check
  server 172.30.4.67 172.30.4.67:3306 check

listen haproxy_172.30.4.215_3308
  bind *:3308
  mode tcp
  timeout client 10000s
  timeout server 10000s
  balance leastconn
  option httpchk
  option allbackups
  default-server port 9200 inter 2s downinter 5s rise 3 fall 2 slowstart 60s maxconn 64 maxqueue 128 weight 100
  server 172.30.4.238 172.30.4.238:3306 check
  server 172.30.4.184 172.30.4.184:3306 check backup
  server 172.30.4.67 172.30.4.67:3306 check backup
```
## Reverse proxies - HAProxy

**HAProxy version 1.5.14, released 2015/07/02**

### Statistics Report for pid 27954

#### General process information

<table>
<thead>
<tr>
<th>pid</th>
<th>27954 (process #1, type=1)</th>
<th>uptime</th>
<th>35,654.359s</th>
<th>system load averages</th>
<th>0.04, 0.18, 0.27</th>
<th>maxprocs</th>
<th>15423</th>
<th>maxconn</th>
<th>6132</th>
<th>maxpipes</th>
<th>0</th>
<th>current_qps</th>
<th>7</th>
<th>current_pipes</th>
<th>20</th>
<th>conn_rate</th>
<th>1960</th>
</tr>
</thead>
</table>

#### Reverse proxies

<table>
<thead>
<tr>
<th>Queue</th>
<th>Current</th>
<th>Max</th>
<th>Limit</th>
<th>Cur</th>
<th>Max</th>
<th>Limit</th>
<th>Total</th>
<th>Li/Tot</th>
<th>Last</th>
<th>In</th>
<th>Out</th>
<th>Req</th>
<th>Resp</th>
<th>Status</th>
<th>LastChk</th>
<th>Wght</th>
<th>Act</th>
<th>Bck</th>
<th>Chk</th>
<th>Dwn</th>
<th>Den.html</th>
<th>Thr.html</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontend</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>192</td>
<td>15</td>
<td>5</td>
<td>916</td>
<td>25</td>
<td>916</td>
<td>25</td>
<td>OPEN</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5m34a UP</td>
<td>0</td>
</tr>
<tr>
<td>Backend</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>830</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4m34a UP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5m34a UP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Queue</th>
<th>Current</th>
<th>Max</th>
<th>Limit</th>
<th>Cur</th>
<th>Max</th>
<th>Limit</th>
<th>Total</th>
<th>Li/Tot</th>
<th>Last</th>
<th>In</th>
<th>Out</th>
<th>Req</th>
<th>Resp</th>
<th>Errors</th>
<th>Conn</th>
<th>Status</th>
<th>LastChk</th>
<th>Wght</th>
<th>Act</th>
<th>Bck</th>
<th>Chk</th>
<th>Dwn</th>
<th>Den.html</th>
<th>Thr.html</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontend</td>
<td>0</td>
<td>0</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8192</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3m54a UP</td>
<td>L7CK:200 in 21ms</td>
<td>100</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>44a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>172.30.4.208</td>
<td>0</td>
<td>0</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8192</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3m54a UP</td>
<td>L7CK:200 in 21ms</td>
<td>100</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>44a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>172.30.4.184</td>
<td>0</td>
<td>0</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8192</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3m54a UP</td>
<td>L7CK:200 in 21ms</td>
<td>100</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>44a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>172.30.4.67</td>
<td>0</td>
<td>0</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8192</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4m34a UP</td>
<td>L7CK:200 in 21ms</td>
<td>100</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backend</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>820</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4m34a UP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5m34a UP</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Queue</th>
<th>Current</th>
<th>Max</th>
<th>Limit</th>
<th>Cur</th>
<th>Max</th>
<th>Limit</th>
<th>Total</th>
<th>Li/Tot</th>
<th>Last</th>
<th>In</th>
<th>Out</th>
<th>Req</th>
<th>Resp</th>
<th>Errors</th>
<th>Conn</th>
<th>Status</th>
<th>LastChk</th>
<th>Wght</th>
<th>Act</th>
<th>Bck</th>
<th>Chk</th>
<th>Dwn</th>
<th>Den.html</th>
<th>Thr.html</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frontend</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>12</td>
<td>12</td>
<td>1m34a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>6</td>
<td>3m34a UP</td>
<td>L7CK:200 in 30ms</td>
<td>100</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>35a</td>
<td></td>
</tr>
<tr>
<td>172.30.4.208</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>12</td>
<td>12</td>
<td>1m34a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>6</td>
<td>3m34a UP</td>
<td>L7CK:200 in 30ms</td>
<td>100</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>35a</td>
<td></td>
</tr>
<tr>
<td>172.30.4.184</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>12</td>
<td>12</td>
<td>1m34a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>6</td>
<td>3m34a UP</td>
<td>L7CK:200 in 26ms</td>
<td>100</td>
<td>Y</td>
<td>1</td>
<td>1</td>
<td>50s</td>
<td></td>
</tr>
<tr>
<td>172.30.4.67</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>12</td>
<td>12</td>
<td>1m34a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>6</td>
<td>3m34a UP</td>
<td>L7CK:200 in 24ms</td>
<td>100</td>
<td>Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Backend</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>12</td>
<td>12</td>
<td>18</td>
<td>1m34a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>6</td>
<td>3m34a UP</td>
<td>100</td>
<td>Y</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Display option:**
- Status
- Error (404)
- Upstream
- Downstream
- Total

**External resources:**
- Primary site
- Updates (v1.5)
- Online manual
Reverse proxies - Keepalived

- Designed to detect failed nodes and move VIP around
- One single VIP for connections, no load balancing is possible
  - Might be useful to build high availability if you connect only to a single node
- Potential use cases:
  - Galera cluster (for HA only)
  - Master - standby replication
  - pair of hosts connected through DRBD
Reverse proxies - Keepalived

- Example config for Galera cluster of three nodes
- Keepalived should be installed on all hosts
- All hosts should be using this configuration
- Leverage mysqlchk script deployed by ClusterControl
- Percona’s clustercheck can also be used
- Can also be configured to use multicast

```bash
vrrp_script chk_galera {
    script "/usr/local/sbin/mysqlchk"
    interval 1
}
vrrp_instance PXC {
    state MASTER
    interface eth0
    virtual_router_id 51
    priority 100
    unicast_src_ip 172.30.4.238
    unicast_peer {
        172.30.4.67
        172.30.4.184 # The other lb
    }
    nopreempt
    virtual_ipaddress {
        172.30.4.100
    }
    track_script {
        chk_galera
    }
    notify_master "/bin/echo 'now master' > /tmp/keepalived.state"
    notify_backup "/bin/echo 'now backup' > /tmp/keepalived.state"
    notify_fault "/bin/echo 'now fault' > /tmp/keepalived.state"
}
Reverse proxies - NGINX

- A HTTP server and a HTTP reverse proxy
- Since 1.9 can be used as a TCP reverse proxy
- Limited features in free (NGINX) version
  - No configurable port for health checks
  - Not possible to integrate with clustercheck scripts
- More features in paid (NGINX Plus) version
  - Advanced health checks
  - Custom port for checking the health of the backend
Reverse proxies - NGINX

- Functionality similar to HAProxy (with regards to MySQL)
- Read/write split could be done using two ports
- Backup servers work differently though - all at once
- No understanding of MySQL and Galera states
- Has to be used with clustercheck-iptables

```yaml
stream {
  upstream stream_reads {
    zone tcp_servers 64k;
    server 172.30.4.67:3306;
    server 172.30.4.184:3306;
    server 172.30.4.238:3306;
  }
  server {
    listen 3310;
policy proxy_pass stream_reads;
    proxy_connect_timeout 1s;
  }
}
upstream stream_writes {
  zone tcp_servers 64k;
  server 172.30.4.67:3306;
  server 172.30.4.184:3306 backup;
}
server {
  listen 3311;
policy proxy_pass stream_writes;
    proxy_connect_timeout 1s;
  }
```
SQL-aware proxies
SQL-aware proxies - MaxScale

- Created by MariaDB Corporation, currently in version 2.0
- Tool designed to solve load balancing needs of DBA’s
  - Handles read/write split
  - Automatically tracks health of backend nodes
- Understands MySQL and Galera states
  - Master or slave? Synced or donor?
SQL-aware proxies - MaxScale

- Support for SSL
- Support for replication, Galera and MySQL Cluster
- Statement hinting
- Where to route
- Maximum replication lag
- Nagios integration

```
[RW]
type=service
router=readwritesplit
servers=svr_172304238,svr_172304184,svr_17230467
user=user
passwd=D469E414F1C095B40B7F8C8776807C83
max_slave_connections=100%
router_options=slave_selection_criteria=LEAST_CURRENT_OPERATIONS

[RR]
type=service
router=readcomnnroute
router_options=synced
servers=svr_172304238,svr_172304184,svr_17230467
user=user
passwd=D469E414F1C095B40B7F8C8776807C83
```
SQL-aware proxies - MaxScale

- Several types of routers are available:
  - readwritesplit - performs read/write split
  - readconnroute - round-robin access
  - dbfwfilter - filters queries based on regex
  - schemarouter - implements schema-based sharding
  - binlogrouter - MaxScale as a binlog server
  - avrorouter - saves binlogs in Avro file

- Limitations
  - Read queries are routed to master if executed within a transaction
  - Stored procedures, UDF’s, prepared statements are sent to the master only
  - Session variables and settings are sent to all hosts
  - No connection multiplexing
  - No seamless failover
SQL-aware proxies - MaxScale

- Lack of real-time traffic statistics
- No remote admin interface (available in 1.4.x, removed in 2.0)
- Most of configuration settings require restart
- Lack of more advanced routing configuration
  - Route to master or slave, using hints
  - No regex-based routing
- Serious scalability issues with more than 100 concurrent threads running
  - Based on Vadim Tkachenko’s blog post from May 12, 2016
  - Tested on MaxScale 1.4.1
- Performance regression when using more than 16 threads for MaxScale (from the blog post as stated above)
- Non-GPL license starting from 2.0
  - Up to two nodes in production free (more - paid)
SQL-aware proxies - MaxScale
## SQL-aware proxies - MaxScale

- **[600s] threads: 100, tps: 7724.81, reads: 108143.72, writes: 0.00, response time: 16.65ms (95%), errors: 0.00, reconnects: 0.00**

<table>
<thead>
<tr>
<th>General statistics:</th>
</tr>
</thead>
<tbody>
<tr>
<td>total time: 600.0089s</td>
</tr>
<tr>
<td>total number of events: 4400776</td>
</tr>
<tr>
<td>total time taken by event execution: 59987.5850s</td>
</tr>
<tr>
<td>response time:</td>
</tr>
<tr>
<td>min: 4.97ms</td>
</tr>
<tr>
<td>avg: 13.39ms</td>
</tr>
<tr>
<td>max: 502.37ms</td>
</tr>
<tr>
<td>approx. 95 percentile: 17.75ms</td>
</tr>
</tbody>
</table>

- **[600s] threads: 250, tps: 8208.70, reads: 114915.94, writes: 0.00, response time: 34.93ms (95%), errors: 0.00, reconnects: 0.00**

<table>
<thead>
<tr>
<th>General statistics:</th>
</tr>
</thead>
<tbody>
<tr>
<td>total time: 600.0181s</td>
</tr>
<tr>
<td>total number of events: 4998034</td>
</tr>
<tr>
<td>total time taken by event execution: 149984.2452s</td>
</tr>
<tr>
<td>response time:</td>
</tr>
<tr>
<td>min: 6.45ms</td>
</tr>
<tr>
<td>avg: 30.01ms</td>
</tr>
<tr>
<td>max: 11250.81ms</td>
</tr>
<tr>
<td>approx. 95 percentile: 35.59ms</td>
</tr>
</tbody>
</table>
SQL-aware proxies - ProxySQL
SQL-aware proxies - ProxySQL

- Created by René Cannaò, currently in version 1.2.4
- Designed to handle complex topologies
  - Implements read write split
  - Support for multiple hostgroups
  - Tracks health of backends on per-query basis
  - Advanced routing based on query rules
SQL-aware proxies - ProxySQL

- Admin interface uses SQL
  - INSERT INTO mysql_servers ... 
  - SET mysql-query_retries_on_failure=...;
- Cisco-like configuration approach
  - LOAD MYSQL QUERY RULES TO RUNTIME;
  - SAVE MYSQL USERS TO DISK;
- Virtually every setting can be changed dynamically

- Query rules mechanism for great flexibility
  - Match queries using regex
  - Match query types using digest hash
  - Route queries based on user, schema, regex
  - Query rewrite and query mirroring
  - Query caching
SQL-aware proxies - ProxySQL

- Connection multiplexing - minimizes number of connections to backends
- Support for MySQL replication
- Support for Galera Cluster using external scripts
- Example monitoring script is provided with the ProxySQL
- Non-standard approach for backed health monitoring
- Health check happens when traffic is sent
- Monitor module extends functionality
- Detailed statistics about traffic handled by the ProxySQL

```sql
MySQL [(none)]> SELECT * FROM stats_mysql_query_digest LIMIT 10;
```

<table>
<thead>
<tr>
<th>hostgroup</th>
<th>schema_name</th>
<th>user_name</th>
<th>digest</th>
<th>digest_text</th>
<th>count_star</th>
<th>first_seen</th>
<th>last_seen</th>
<th>sun_time</th>
<th>min_time</th>
<th>max_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>sibttest</td>
<td>sibttest</td>
<td>0x6cd03d667e0f2e29</td>
<td>UPDATE sibttest16 SET cn WHERE id=?</td>
<td>2206141</td>
<td>147283834</td>
<td>147356665</td>
<td>9552608152</td>
<td>14</td>
<td>6379798</td>
</tr>
<tr>
<td>1</td>
<td>sibttest</td>
<td>sibttest</td>
<td>0x23f94a6b36cb67a0</td>
<td>SELECT c FROM sibttest16 WHERE id BETWEEN ? AND ? ORDER BY c</td>
<td>2206151</td>
<td>147283834</td>
<td>147356665</td>
<td>3815554402</td>
<td>17</td>
<td>498544</td>
</tr>
<tr>
<td>1</td>
<td>sibttest</td>
<td>sibttest</td>
<td>0x580eef79e4b1093f</td>
<td>SELECT c FROM sibttest16 WHERE id BETWEEN ? AND ?</td>
<td>2206151</td>
<td>147283834</td>
<td>147356665</td>
<td>3186564149</td>
<td>18</td>
<td>3805576</td>
</tr>
<tr>
<td>1</td>
<td>sibttest</td>
<td>sibttest</td>
<td>0x665f74a6b36cb67a0</td>
<td>SELECT c FROM sibttest16 WHERE id BETWEEN ? AND ?</td>
<td>2206151</td>
<td>147283834</td>
<td>147356665</td>
<td>3512787372</td>
<td>18</td>
<td>1447613</td>
</tr>
<tr>
<td>1</td>
<td>sibttest</td>
<td>sibttest</td>
<td>0x6a9c63c38c9b900b</td>
<td>SELECT c FROM sibttest16 WHERE id=?</td>
<td>2206151</td>
<td>147283834</td>
<td>147356665</td>
<td>2722665062</td>
<td>16</td>
<td>537308</td>
</tr>
<tr>
<td>0</td>
<td>sibttest</td>
<td>sibttest</td>
<td>0x8a6a69a48b15e953e</td>
<td>INSERT INTO sibttest16 (id, c) VALUES (?, ?, ?)</td>
<td>2206151</td>
<td>147283834</td>
<td>147356665</td>
<td>9882468066</td>
<td>18</td>
<td>2152086</td>
</tr>
<tr>
<td>0</td>
<td>sibttest</td>
<td>sibttest</td>
<td>0x213f94a6b36cb67a0</td>
<td>UPDATE sibttest9 SET cn=? WHERE id=?</td>
<td>2206151</td>
<td>147283834</td>
<td>147356664</td>
<td>9549954028</td>
<td>18</td>
<td>1796095</td>
</tr>
<tr>
<td>0</td>
<td>sibttest</td>
<td>sibttest</td>
<td>0x69f5fd714a8b601</td>
<td>UPDATE sibttest9 SET cn=? WHERE id=?</td>
<td>2206151</td>
<td>147283834</td>
<td>147356664</td>
<td>1287740227</td>
<td>16</td>
<td>611474</td>
</tr>
<tr>
<td>0</td>
<td>sibttest</td>
<td>sibttest</td>
<td>0x62c3b672c3c59ca</td>
<td>SELECT c FROM sibttest9 WHERE id BETWEEN ? AND ? ORDER BY c</td>
<td>2206151</td>
<td>147283834</td>
<td>147356664</td>
<td>3805281842</td>
<td>13</td>
<td>363537</td>
</tr>
<tr>
<td>0</td>
<td>sibttest</td>
<td>sibttest</td>
<td>0x615626a0202d01</td>
<td>UPDATE sibttest13 SET cn=? WHERE id=?</td>
<td>2206151</td>
<td>147283834</td>
<td>147356664</td>
<td>9595698119</td>
<td>17</td>
<td>6379383</td>
</tr>
<tr>
<td>0</td>
<td>sibttest</td>
<td>sibttest</td>
<td>0x6c63c667e0f2e29</td>
<td>UPDATE sibttest16 SET cn=? WHERE id=?</td>
<td>2206151</td>
<td>147283834</td>
<td>147356665</td>
<td>9552608152</td>
<td>14</td>
<td>6379798</td>
</tr>
</tbody>
</table>

10 rows in set (0.01 sec)
SQL-aware proxies - ProxySQL

- No client side SSL support
- Lack of out-of-the-box support for explicit transactions
  - You have to either use SET autocommit=0
  - Or enable persistent transactions for app user
- Session variables are tricky to support
  - You have to make sure they are routed to a single, writable node (i.e. replication master)
- No support for prepared statements in stable (1.2.x version)
  - Support added to 1.3, which is in alpha state now
- Lack of nagios integration
- Lack of internal support for Galera
- Although support via external script gives you more options for customization
SQL-aware proxies - MySQL Router

- Created by Oracle, currently in version 2.0.3
- Designed to provide high availability and scalability to your MySQL backends
- Supports external plugins using MySQL Harness
- Integrates well with MySQL Fabric for high availability setups
Two modes for routing

- **read-only** - read-only connections routed in round-robin fashion
- **read-write** - read-write connections routed to the first working node from the list

Support for MySQL Fabric:

- Can route traffic to given HA group
- Topology detection via Fabric
- Adds high availability through Fabric Cache
SQL-aware proxies - MySQL Router

- Work in progress to improve capabilities of MySQL Router
- New version, 2.1, available via MySQL Labs
- Support for MySQL InnoDB Cluster - a solution based on MySQL group replication
  - Provides a support for of InnoDB cluster Metadata Schema
  - Caches metadata of the InnoDB Cluster and improves high availability
Application connectors
Application connectors - mysqlind_ms

- Native driver for MySQL in PHP
  - Default since PHP 5.4
- mysqlind_ms adds option to implement read/write split and load balancing to PHP application
- Supports failover handling
- Supports query routing modifications through SQL hints
Application connectors - mysqlnd_ms

- Manual and automated failover support
  - Failover to master
  - For read-only request - loop through slaves before attempting to reach master
- Hints in SQL comments:
  - MYSQLND_MS_MASTER_SWITCH - run on the master
  - MYSQLND_MS_SLAVE_SWITCH - run on the slave
  - MYSQLND_MS_LAST_USED_SWITCH - run on last used node
- Application can use hints to route queries
  - Transactions sticked to master
  - Handle user-defined variables
  - Hints can also be used in conjunction with filters to route traffic to a given hostgroup - makes possible to implement sharding
  - Empowers developers - lots of control over DB handling in the application
  - Callback function to manually implement support for more complex scenarios
Application connectors - mysqlnd_ms

- Main focus is on master - slave replication
- Supports also multimaster environments
- No support for Galera states, unless you implement it in the application
- Empowering developers and moving database routing decisions to the application makes DBA’s live harder
- Unless he’s responsible for the database access code
- Limited support for topology change detection
Application connectors - MySQL Connector/J

- MySQL Connector/J, currently in version 5.1.39 (6.0.3 in beta)
- Adds great deal of flexibility to Java applications when it comes to database layer handling
- Failover handling
- Support for topology changes
- Advanced routing options
Application connectors - MySQL Connector/J

- Failover support made as close to transparent as possible
  - Differentiates between read/only and read/write connections
  - Transaction handling in failover (rollback if needed)
  - Supports scale-out by routing read/only traffic to slaves
  - Connection.setReadOnly(true/false) call

- Multimaster support
  - jdbc:mysql://address=(type=master) (host=master1host), address=(type=master)(host=master2host)
  - Live reconfiguration of replication topology
    - getMasterHosts(), getSlaveHosts()
    - addSlaveHost(), promoteSlaveToMaster()
    - removeSlaveHost(), removeMasterHost()
Health checks used with proxies
Health checks used with proxies - clustercheck

- HTTP check for HAProxy
  - Will work with any other proxy which supports http checks on custom port

- Script itself is a bash script which checks MySQL status
  - Designed for Galera but can be easily modified to use with replication
  - Checks wsrep_local_state for Galera states
  - Configurable Donor/Desync state handling

- Script returns HTTP codes
  - HTTP/1.1 200 OK - if node is available
  - HTTP/1.1 503 Service Unavailable - otherwise

- Clustercheck should be installed on all databases nodes
  - It’s executed by xinetd, listening on custom port (9200 by default)
  - Make sure to configure it in /etc/services
Health checks used with proxies - clustercheck

- By default, script is executed directly from xinetd
- Could become a problem if it’s executed too often
- Numerous forks are available out there
- One of nice modifications was to execute script via cron and store result in shared memory
- xinetd executes another script which reads the status - minimizing impact on MySQL

```bash
# default: on
# description: mysqlchk
service mysqlchk
{
# this is a config for xinetd, place it in /etc/xinetd.d/
  disable = no
  flags = REUSE
  socket_type = stream
  port = 9200
  wait = no
  user = nobody
  server = /usr/bin/clustercheck
  log_on_failure += USERID
  only_from = 0.0.0.0/0
# recommended to put the IPs that need
# to connect exclusively (security purposes)
  per_source = UNLIMITED
}
```
Health checks used with proxies - clustercheck-iptables

- Derived from Percona’s clustercheck
- Created by Severalnines’ Ashraf Sharif
- Designed to work with proxies which can do only simple check on port used for service
  - For example, NGINX
- Idea is simple - use iptables to open or close port, depending on node’s state

- Run script in a background, as a daemon
- Setup redirection from port 3308 to port 3306 in PREROUTING chain
- Every second checks the state of a node
  - If node is deemed unhealthy, remove redirection rule from PREROUTING chain
- Similar approach to health detection as in Percona's clustercheck
  - Checks wsrep_local_state for Galera states
Great deal of options to pick from

- Everyone can choose a suitable solution

Do you use NGINX in your setup? Why not to leverage your knowledge of it and use it for databases too?

- You’d need clustercheck-iptables or NGINX Plus, though

Do you want to build all database handling logic in your application?

- PHP and Java give you great opportunities for that

Or maybe you prefer to move complexity of database layer from the application?

- MaxScale or ProxySQL is there for you

Maybe you want to use advanced features like query rewriting or sharding?

- Again, ProxySQL or MaxScale will work great for you

Do you work with MySQL Fabric and want some integration?

- MySQL Router could be a great choice
Thank You!

Q&A

Contact: krzysztof@severalnines.com