Key Indicators in HAProxy
Willy Tarreau
(willy@haproxy.org)

FRnOG 31
Disclaimer

A high level presentation was given based on some of these slides at Dotscale 2018. This presentation will instead focus on deep-diving into the technical stuff.
What does the LB see?

- global failures (aborts, timeouts)
- abnormal delays caused by network retransmits
- connection failures and retries caused by bad tuning (eg: conntrack)
- connection slowdowns caused by inefficient firewall policies (#rules)

...
What does the LB see (…) ?

- **client-side** issues (*BW limitations*)
- **per-URL** processing time (*application issues, svc partners*)
- **per-node vs per-cluster** variations
  => *narrow down to individual node or shared resource*
- **deployment** issues: new occasional error on a specific page, can be addressed before going full-scale
Accessing metrics in HAProxy

- **Logs** :
  - Halog, ELK, Prometheus, ...
  - Provides unique-id for tracing/event correlation

- **Stats** :
  - Stats page, CLI, hatop

- **Stick-tables** (per arbitrary key like IP, URL, cookie) :
  - Byte count, cumulated/concurrent conns, errors, ...
Sequence of events on HAProxy
Sequence of events on HAProxy

Client → front network → HAProxy → back network → Server

TCP: Operating system only
Sequence of events on HAProxy
Sequence of events on HAProxy

- **Th** affected by:
  - key size (CPU time)
  - lack of entropy (/dev/random instead of /dev/urandom)
  - network losses with the client
Sequence of events on HAProxy

Client -> HAProxy -> Server

Front network

TCP: Operating system only

Th
Ti
Sequence of events on HAProxy

- TCP: Operating system only
- Th
- Ti

**Ti affected by:**
- client pauses (preconnect, connection reuse)
- network losses with the client
- if POST: MTU issues with the client (VPN)
Sequence of events on HAProxy
Sequence of events on HAProxy

TCP: Operating system only

Tq affected by:
- network losses with the client
- large requests: client’s bandwidth
- if POST: MTU issues with the client (VPN)
Sequence of events on HAProxy
Sequence of events on HAProxy

Tw affected by:
- server's maxconn value
- # requests already in queue
- server's response time on previous requests
Sequence of events on HAProxy
Sequence of events on HAProxy

Tc affected by:
- network losses to the server
- TLS handshake on the server (CPU time)
- lack of entropy on the server (if TLS used)
Sequence of events on HAProxy
Sequence of events on HAProxy

Tr affected by:
- network losses to the server
- server’s processing time
- shared component behind server (typ: database)
Sequence of events on HAProxy
Sequence of events on HAProxy

Td affected by:
- network losses to the server
- server’s bandwidth
- haproxy’s bandwidth
- client’s bandwidth (for large objects)
More timers to come in HAProxy 1.9

- HAProxy now supports heavier per-request workloads (Lua, device identification, …)
- Processing times over 200 µs can become noticeable

Actions:
- log per-request total CPU time spent in analysers
- log per-request total CPU time spent in TLS handshake
- log per-request total latency added by other tasks
- log per-process total stolen time by other processes
- Ability to kill offending tasks
- Ability to alert on high latencies
Event timing reports

- Timers are averaged in the stats
- Each timer appears in the logs
- **Halog** `-rt/-RT/-pct` for quick analysis
- Each timer crossing a limit triggers a **timeout**
- Each abort at a specific step causes a **hard error**

=> **termination codes**
Termination codes

- Distinguish between **timeout** and **abort**
- Indicate **whom** *(client, server, haproxy, kill, ...)*
- Indicate **when** *(req, queue, connect, response...)*
- Completed by persistence cookie indications
- Filtered and sorted by halog:

```bash
# halog -tcn|-TCN ... # for filtering
# halog -tc           # for sorting
```
Other relevant metrics: HTTP status distribution

- Stats page: distribution per frontend/backend/server
- Filter by ranges: `halog -hs/-HS`
- Sorted output: `halog -st`

=> graph the distribution and watch for variations between application deployments
Other relevant metrics: queue length

- Uses server `maxconn`
- Grows **exponentially** with slowdowns: easy to detect!
- Tells you **how many** extra servers you need
- Reported by `halog -Q/-QS`
- Shown in real time on the stats page per backend/srv

=> *If you watch only one metric, watch this one!*
Other relevant metrics: LB fairness

LB algorithm implies **fairness** between servers:
- Equal request count with **roundrobin**
  => Higher than average concurrency indicates abnormally slow server
- Equal load with **leastconn**
  => Low req count indicates abnormally slow server

=> *graph relevant values within the farm*

<table>
<thead>
<tr>
<th>Queue</th>
<th>Session rate</th>
<th>Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cur</td>
<td>Max</td>
</tr>
<tr>
<td>web01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>web02</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>web03</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Backend</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Other relevant metrics: error rate

- Global: `halog -e`
- Per server: `halog -srv`
- Per client IP: `halog -e -ic` *(detect bad CDN nodes)*
- Per URL: `halog -ue`
- Stats page: `per frontend/backend/server`
- Stick-tables: `per arbitrary key using http_err_rate()`

=> no threshold, watch for variations
Useful entries in log-format

- Default httplog format is quite rich
- Can be improved using the `log-format` directive
- Hint: log stick-table stats for similar keys

```
haproxy[14389]: 10.0.1.2:33317 [06/Feb/2018:12:14:14.655] http-in static/srv1 10/0/30/69/109 200 2750 -- SDNN 1/1/1/1/0 0/0 {haproxy.org} {} "GET /index.html HTTP/1.1"
```
Tips: sampling : why / when

"I can't enable logs, I have too much traffic!"

- an average syslog server can store 20k events/s without sweating
- that's 1.7B events/day or 350GB of uncompressed haproxy logs/day
- compresses to 1TB/month
- for $100 you can store 4 months with no loss
- have more traffic / not interested in this level of detail?

```bash
# log only 5% of requests
http-request set-log-level silent unless { rand(100) -lt 5 }
```
Tips: selective logging: why / when

- you only want to catch \texttt{suspicious} events
- disable logging unless \(T_c/T_q/T_r/T_w/\ldots\) is above a certain threshold
- on the fly for \texttt{selected keys} from the CLI + \texttt{stick-table}
- also see "\texttt{option dontlognormal}"
- **WARNING**: you'll lose any valid \texttt{reference}
Tips: other halog goodies

- Poorly documented, use `halog --help`
- response time per url: `halog -uat`
- errors per server: `halog -srv`
- Percentiles on req/queue/conn/resp times: `halog -pct`
- detect stolen CPU / swap: `halog -ac ... -ad ...`
- very fast (1-2 GB per second)

=> Use it in production to figure the relevant metrics
Success stories

Customer spotting a broken fiber between two core switches

- Tc from HA1 to srv 1, 2, 3, 5 always low, srv 4, 6 high at 99 pct
- Tc from HA2 to srv 1, 2, 4, 6 always low, srv 3, 5 high at 99 pct
  => both haproxy and servers out of cause
- Issue rate stable at various traffic levels => not congestion
- Inter-switch link apparently at cause but not for all flows
- Inter-switch link made of two fibers balanced on MAC tuple
- Thanks to long-term logs, origin could even be identified
Success stories

Customer figuring a wrong web server configuration using /dev/random

- Tc abnormally high with lots of random values to several seconds, and only for TLS
- timer also covers TLS handshake

=> not a network, hardware or performance issue, only server config.

=> system was regularly running out of entropy due to mistakenly using /dev/random as a random source for SSL
Conclusion

- exploit your **stats**
- **enable logs on LBs**, no excuse for not doing it!
- process them automatically, **manually once in a while**
- **compare numbers** between similar objects
- detect anomalies
- fix problems before they are witnessed
- profit :-)