EPJ Web of Conferences **214**, 08028 (2019) https://doi.org/10.1051/epjconf/201921408028
CHEP 2018

Concurrent Adaptive Load Balancing at CERN

Paulo Canilho[1,*], *Ignacio* Reguero[1], and *Pablo* Saiz[1]

[1]CERN IT Department, CH-1211 Geneva 23, Switzerland

Abstract. CERN is using an increasing number of DNS based load balanced aliases (currently over 700). This article explains the Go based concurrent implementation of the Load Balancing Service, both the client (lbclient) and the server (lbd). The article describes how it is being progressively deployed using Puppet and how concurrency greatly improves scalability, ultimately allowing a single master-slave couple of Openstack virtual machines to server all the aliases. It explains the new implementation of the lbclient, which, among other things, allows to incorporate Collectd metrics to determine the status of the node and takes advantage of the Go language concurrency features to reduce the real time needed for checking the status of the node. The article explains that the LBD server acts as an arbiter getting feedback on load and health from the backend nodes using snmp (Simple Network Management Protocol) to decide which IP addresses the LB alias will present. While this architecture has been used since long at CERN for DNS based aliases, the LBD code is generic enough to drive other load balancers. A proof of concept using HAProxy to provide adaptive responses to load and health monitoring has been implemented.

1 Introduction

This paper describes the main components of the Load Balancing Service created at CERN. The service has been running in production for the last twelve years, and, recently, it has undergone a major reorganization to improve its performance, scalability and monitoring.

First, the paper gives an overview of the system and introduces the different components. After that, it describes in more detail the client, the server and the way they communicate with each other. The following chapter describes how the service is being handled, and a prototype that was evaluated to use HAProxy instead of DNS as a backend. Finally, a summary concludes the article.

2 System architecture - overview

The overall architecture of the load balance service is depicted in Figure 1. There is a central machine running the Load Balanced Daemon (LBD). This daemon periodically probes the clients and queries their status. The different clients run the lbclient, which, according its configuration, decides if the node is healthy and its current load. After gathering the results from all the nodes that could potentially sit behind an alias, the LBD will select the best candidates, and it will update the DNS server accordingly.

The communication between the LBD and the nodes running lbclient is done through SNMP.

*e-mail: lb-experts-public@cern.ch

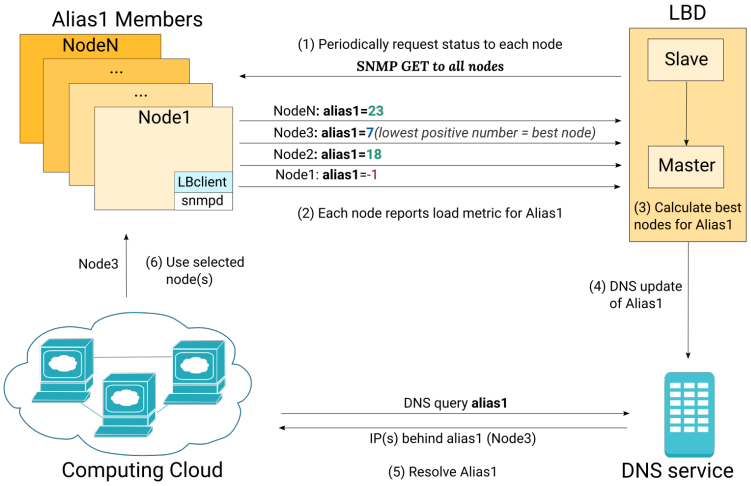

Figure 1: Components of the Load Balancing Service.

3 Load-balanced client

3.1 Implementation

The load-balanced client (lbclient) runs on the hosts that could be behind a DNS alias. The configuration of the lbclient describes a set of metrics, called *checks*, that will be used to evaluate if the node is healthy, and, a second set of metrics, called *loads* to evaluate how busy the node is. A subset of the checks and loads can be found in Table 1. Most of these metrics are generic, like checking the disk space, or if there are services listening behind particular ports. Others have been created to take advantage of other CERN infrastructure components, like lemon [1], roger [2] and collectd [3].

The latest version of the lbclient extended the *c-style* arithmetic/string expression and *JSON* support in several metrics. Using this, the user is able to perform complex checks, similar to the ones described in Table 1.

3.2 Deployment

The deployment of the lbclient requires the following steps:

- The first step is the installation of the lbclient software. This is currently done with an rpm (Redhat Package Manager).
- The second one is the configuration of the lbclient. Currently, this is done with a Puppet resource.
- The last one is to configure a SNMP daemon to allow incoming calls from the LBD. The same Puppet resource mentioned above takes care also of setting the SNMP daemon, opening the firewall and configuring the rules of SElinux.

The simplest configuration consists of a single configuration file per node, with all the metrics needed for the configuration. This layout can be extended to allow a single node to sit behind different aliases, and even use different metrics for the evaluation of each alias. As an example, a machine that runs both a web server and a database server could have two

Type	Tool	Parameters	Description
check	nologin		verifies that the files /etc/iss.nologin and /etc/nologin do not exist
check	webdaemon		verifies that a process listens on port 22
check	daemon	{"port":[80,8080], "protocol":["tcp", "udp"]}	verifies that a process listens on both ports 80 and 8080, using both tcp and udp
check	tmpfull		verifies that the location /tmp/ is not full
check	collectd	[X:1]>2 && [Y]=0	using the collectd service, verifies that the 2nd index of the X metric slide is greater than 2 and that the value of the metric Y is equal to 0
check	command	<command_name>	verifies that <command_name> returns 0 when executed
load	constant	4	the load of the node will be 4
load	lemon	[12163:1] + [13423]	using the lemon service, the load of the node will be the sum of the 2nd index of the metric number 12163 and the value of the metric 13423
load	collectd	[Z] * [M:key]	the load of the node will be the value of the collectd entry [Z] multiplied by the value that has is identified by 'key' of the entry [M]

Table 1: Some metrics available when using lbclient.

aliases configured. The first alias would check if the web server is running properly, whereas the second alias would check if the database is running properly.

4 Load-Balanced Daemon

4.1 Implementation

The Load Balancing Daemon (LBD) has been running at CERN in production for the last twelve years. It is a stable service, which has been adjusted and kept up to date. At the same time, it was reaching its limit on the number of aliases that it could handle. This was an opportunity to take a step back, and evaluate the assumptions and decisions taken during the first implementation. The first lesson taken was that it would be better to parallelize the evaluation of the aliases. A sequential evaluation on a single node does not scale. This lead to two improvements in the new generation of the Load Balancing Daemon:

- Since the aliases are independent, it should be possible to parallelize their evaluation. This will take advantage from the multiple core architecture. Due to the possible timeouts an retries on each of the SNMP calls, the sequential approach was already reaching five minutes of running time. Thanks to parallelization, the time was brought down to less than one minute.

- Split all the aliases into different partitions, and run a different LBD per partition. This allowed a progressive deployment of the new version, as it will be explained on the next chapter.

The workflow diagram of the LBD is depicted in Figure 2. The aliases might have different refresh policies, so, first, the LBD figures out which aliases have to be evaluated. Once it has done that, there are three main tasks:

- First, identify all the nodes that could potentially sit behind all the aliases being evaluated. Call the lbclient on all these nodes. Note that, if a node could be behind multiple aliases, it should be called only once.
- Then, for each alias, compare the response of the nodes, and select the best nodes.
- Finally, compare the selected nodes with the current nodes that appear on the DNS. If they are different, update the DNS accordingly.

The first two items can be executed concurrently, as long as all the nodes have been contacted before starting the evaluation of the aliases. The last part, the update of the DNS, has been left sequential to reduce the amount of simultaneous calls done to the DNS service. Since most of the execution time of the LBD is spent on the evaluation of the nodes, this sequential part was acceptable.

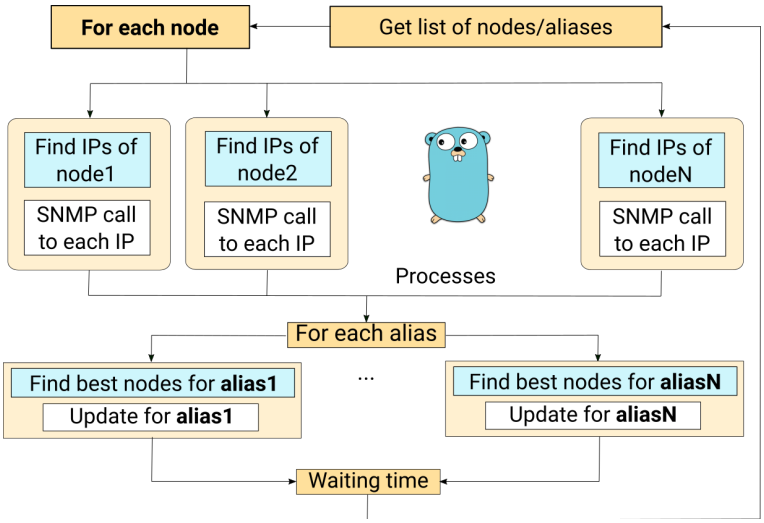

Figure 2: Workflow of the Load Balanced Daemon.

4.2 Deployment

The Load-Balance Daemon needs two sources of information to be able to process the aliases:

- The list of aliases that it has to process.
- The list of nodes that might be behind each of the alias.

The former information is stored in a central database, called ermis [4]. The current implementation of ermis is a Django interface on top of a MySQL database. The list of nodes is currently fetched from Puppetdb, querying all the nodes that have configured the lbclient Puppet resource.

Taking advantage of the Puppet infrastructure at CERN [2], every time that Puppet runs on the LBD, it gathers all the information from both sources, and creates the configuration file that the LBD uses. Note that there is no strong interconnection between the LBD and Puppet. As long as the configuration file gets created and contains a list of aliases, the LBD can run. This enables to run the LBD and lbclient setup on sites that do not have Puppet.

5 Communication

The Load Balanced Daemon uses SNMP as the communication protocol between the LBD and the lbclient running on the nodes. This was identified as a simple, secure and small footprint solution to get information from the node. The SNMP daemon has been configured in such a way that the request arriving asking for a particular identifier will execute the lbclient command and it will return its output. The output could be either an integer, in the cases where the node has a single configuration file, or string containing a list key-value pairs, where the keys are the names of the aliases configured on that node, and the value the load of each of them.

The port that the SNMP daemon uses is behind a firewall, and only the traffic coming from the LBD is allowed. SElinux had to be configured as well to allow the execution of the lbclient.

6 Operations

6.1 Partitions

As mentioned before, the concept of partitions was introduced to split all the aliases into several groups. Each partition can be evaluated by a different set of LBD. Thanks to this, the transition from the old version of the LBD to the new GoLBD has been done progressively. The initial situation was that all the aliases were on the partition of the old LBD. Then, a new partition was created, where the aliases were evaluated by the Go implementation. Then, the aliases used for development were moved into this partition. The rest of the aliases followed, each time migrating a small bunch of aliases. The full operation was done over several months, thus minimizing the risk of sudden transitions and gaining expertise on the service. The migration was transparent for the end users.

6.2 Achieving High Availability

Each partition of load balanced aliases is deployed on two nodes running the LBD. The first node, the primary, does the evaluation of the aliases and then it checks if it has to update the DNS. The secondary node does also the evaluation of the aliases. Then, it checks if the primary is alive. In that case, it just goes to sleep until the next iteration. If the secondary finds that the primary is not alive, it will also update the DNS. It was decided that both servers should check the status of the nodes to have some comparison, and to be able to identify network issues. In the scenario where the communication between the primary and the secondary has been cut, both servers would check if they have to update the DNS. As long as they see a coherent state of the nodes, the end result will still be reasonable. If, on the other hand, the primary and the secondary see different behaviour of the nodes, they will both update the DNS with different values.

6.3 Monitoring

The monitoring of the Load Balanced Daemon has been done with the UMA infrastructure [5]. In particular, it uses Logstash to send the data, and then a Kibana dashboard on top of Elasticsearch shows the status of the service. Three different dashboards have been created:

- One dashboard for the creator of the alias. This dashboard presents the nodes that are behind the alias, and how they evolve over time.
- The second one presents more detailed information about the decisions that the LBD took for each alias, and the type of policy that it applied.
- The last dashboard is for the service managers of the LBD, with the status of the LBD machines.

An example of the first dashboard can be seen in Figure 3. This particular alias has had ninety different nodes behind. The plot on the left presents the average load of each of the nodes, evaluated every five minutes over the last thirty day. Then, the fifteen least loaded nodes were selected, and presented behind the alias. The plot on the right presents the percentage of time that each node was behind the alias. The status and load of each node was evaluated every five minutes, and the fifteen least loaded healthy nodes were presented by the alias.

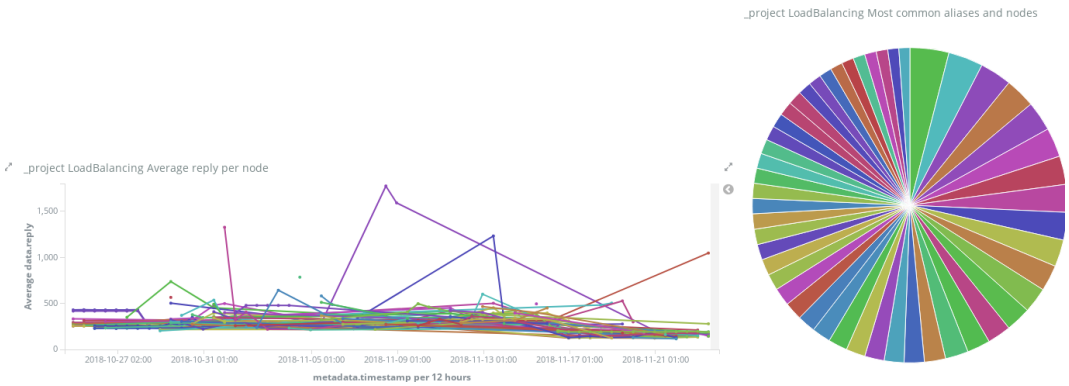

Figure 3: Monitoring of the nodes behind a particular alias.

7 DNS & HAProxy

Most of the work done by the the LBD is in fact not DNS specific. The part of evaluating a list of nodes, getting their health status and choosing the healthiest nodes can be applied to other situations. Following this idea, and as a proof of concept, a version of Load Balancing with an HAProxy [6] backend was implemented. The workflow has been depicted in Figure 4. The LBD uses the same logic to evaluate the nodes, and then it sets the weights of HAProxy accordingly.

8 Future work

The proof of concept of the LBD with HAProxy backend should be taken to the next level, and configure it on real use cases. The scenarios that are being considered at the moment include a standard DNS Load Balancing for the HAProxy service itself, and then the Load Balancing with HAProxy backend for the service nodes.

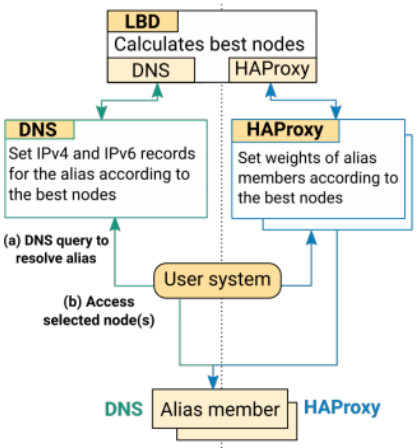

Figure 4: Component-reuse for the implementation of the DNS & HAProxy services.

9 Summary

This paper has presented the Load Balanced system developed and maintained at CERN. It is based on three components:

- A thin and configurable client, running on the nodes.
- A set of servers that can evaluate if the nodes are healthy and their load and update the DNS accordingly.
- A transport mechanism that has been configured so that the server can retrieve the information from the client.

A new implementation has been done this year to improve the service scalability. This implementation has been done in Go, and uses the concurrent features offered by the language to reduce the execution time.

References

[1] LEMON - LHC Era Monitoring. 2018. LEMON - LHC Era Monitoring. [ONLINE] Available at: http://lemon-monitoring.web.cern.ch/. [Accessed 23 October 2018].

[2] B. Jones, G. McCance, S. Traylen and N. Arias (2015). Scaling Agile Infrastructure to People. Journal of Physics: Conference Series, 664(2), p.022026.

[3] Documentation – collectd – The system statistics collection daemon. 2018. Documentation – collectd – The system statistics collection daemon. [ONLINE] Available at: https://collectd.org/documentation.shtml. [Accessed 23 October 2018].

[4] I. Reguero and L. Lobato (2017). DNS load balancing in the CERN cloud. Journal of Physics: Conference Series, 898, p.062007.

[5] A. Aimar et al. (2017). Unified Monitoring Architecture for IT and Grid Services. Journal of Physics: Conference Series, 898, p.092033.

[6] HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer. 2018. HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer. [ONLINE] Available at: https://haproxy.org. [Accessed 26 November 2018].

[7] GitHub. 2018. GitHub - cernops/golbd: Go implementation of CERN lbd DNS Load balancing daemon. [ONLINE] Available at: https://github.com/cernops/golbd. [Accessed 28 November 2018].

[8] GitHub. 2018. GitHub - cernops/golbclient. [ONLINE] Available at: https://github.com/cernops/golbclient. [Accessed 28 November 2018].