"SERVERLOAD BALANCING USING HAPROXY"

Shreyas Lakhe, Ashutosh Shinde, Nandan Sukthankar, Chaitanya Reddy

Department Of Computer Science, Govt.College Of Engg. Pune

Email - shreyaslakhe@gmail.com.

Abstract:

The use of computers and related software packages such as ERP, SAP in manufacturing companies and also the net operations particularly websites requires proper load management as due to the increase in data size, number of users and the speed of operation, the servers are overloaded which reduces it's effectiveness, this results in slowing down of the work and necessitates balancing of load on multi server operations. This paper uses HAProxy software for load balancing of websites servers.Network Load Balancing clusters enable to manage a group of independent servers as a single system for greater scalability, increased availability, and easier manageability. This paper shows implementation of Round Robin, Weighted Round Robin, Source IP hash, URL hash and analyzed the differences. The algorithms are among the most commonly used in load balancing.

Key Words: server load, computer hardware

INTRODUCTION

Load balancing is the most commonly used in server farms where multiple physical boxes are coordinated to fulfill the requests of many end users. Load balancing can be implemented through not only software but can also be implemented using dedicated hardware appliance. One of the most important reasons for load balancing is failover. Failover is the ability of the system to remain operational while one or more of the components have failed or gone down. Many open source load balancing softwares like nginx, HAProxy, BalanceNG and paid load balancing softwares like NetScaler and F5 are available. Out of these load balancing softwares we choose HAProxy because it is easy to install, simple to configure and setup, works fast and suitable for basic purposes. It is faster and less bug prone. It is actually a TCP load balancer with features for HTTP. HAProxy offers the most options of a true load balancer and scales extremely well. It is one of the most popular open source load balancing softwares. Node.js can be used for setting up the backend servers. As an asynchronous event driven framework, Node.js is designed to build fast, scalable network applications capable of handling a huge number of simultaneous connections with high throughput. In Node.js, in addition to normal events here the events are also raised by network connections. When someone connects to your server event is raised and you act accordingly to the event. So it is event driven. Now events are async. Events can come anytime in any sequence. There is no fixed time when an event (a connection request for example) will be raised. Besides this it allows to program it in JavaScript and provides lightning fast JavaScript execution as it uses the V8 engine used by Google which compiles JavaScript into native machine code. This paper paper presents the utility of HAProxy software and its algorithms more particularly round robin, weighted round robin and source ip hash.

LOAD BALANCING

Load balancing is dividing the amount of work that a computer has to do between two or more computers so that more work gets done in the same amount of time and, in general, all users get served faster. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource.

Load balancing uses a software program that is listening on the port where external clients connect to access services. The load balancer forwards requests to one of the "backend" servers, which usually replies to the load balancer. This allows the load balancer to reply to the client without the client ever knowing about the internal separation of functions. It also prevents clients from contacting back-end servers directly, which may have security benefits by hiding the structure of the internal network and preventing attacks on the kernel's network stack or unrelated services running on other ports.

HAPROXY

HAproxy stands for High Availability Proxy, is a popular open source software TCP/HTTP Load Balancer and proxying solution. It is used in many high-profile environments, including GitHub, StackOverflow, Imgur, Instagram, and Twitter. It supports many options which helped us analyze multiple load balancing algorithms. It supports out of band health checks and TCP/HTTP load balancing HAproxy provides security for backend stack HAproxy allows you to manipulate traffic based on incoming URLs or Cookies. HAproxy queries the server status constantly and checks the server before redirecting traffic to it. It has an interface to view statistics.

LOAD BALANCING ALGORITHMS

HaProxy supports approximately seven-eight load balancing algorithms. It supports more or less all the major load balancing algorithms. The following are the load balancing algorithms supported by HaProxy.

ROUNDROBIN

Each server is used in turns, according to their weights. This is the smoothest and fairest algorithm when the server's processing time remains equally distributed. This algorithm is dynamic, which means that server weights may be adjusted on the fly for slow starts for instance. It is limited by design to 4095 active servers per backend.

SOURCE

The source IP address is hashed and divided by the total weight of the running servers to designate which server will receive the request. This ensures that the same client IP address will always reach the same server as long as no server goes down or up. If the hash result changes due to the number of running servers changing, many clients will be directed to a different server.

URI

This algorithm hashes either the left part of the URI (before the question mark) or the whole URI (if the "whole" parameter is present) and divides the hash value by the total weight of the running servers. The result designates which server will receive the request. This ensures that the same URI will always be directed to the same server as long as no server goes up or down. This is used with proxy caches and anti-virus proxies in order to maximize the cache hit rate.

STATIC-RR

Each server is used in turns, according to their weights. This algorithm is as similar to roundrobin except that it is static, which means that changing a server's weight on the fly will have no effect. On the other hand, it has no design limitation on the number of servers.

LEASTCONN

The server with the lowest number of connections receives the connection. Round-robin is performed within groups of servers of the same load to ensure that all servers will be used. Use of this algorithm is recommended where very long sessions are expected, such as LDAP, SQL, TSE, etc... This algorithm is dynamic, which means that server weights may be adjusted on the fly for slow starts for instance.

FIRST

The first server with available connection slots receives the connection. The servers are chosen from the lowest numeric identifier to the highest which defaults to the server's position in the farm. Once a server reaches its maxconn value, the next server is used. The purpose of this algorithm is to always use the smallest number of servers so that extra servers can be powered off during non-intensive hours.

hdr(<name>)

The HTTP header <name> will be looked up in each HTTP request. Just as with the equivalent ACL 'hdr()' function, the header name in parenthesis is not case sensitive. If the header is absent or if it does not contain any value,the roundrobin algorithm is applied inst This algorithm is static by default, which means that changing a server's weight on the fly will have no effect, but this can be changed using "hash-type".

Load Balancing Implementation in HAproxy

The installation procedure for HAProxy 1.5.4 software on Ubuntu 14.04 LTS operating system is as follows.

fig1. Installing haproxy ubuntu

```
sudo add-apt-repository ppa:vbernat/haproxy-1.5
sudo apt-get update
sudo apt-get install haproxy
```

The first line of fig1. adds a dedicated ppa repository vbernet that is needed to install haproxy. The second line of the fig1. updates the package lists to newest versions of packages and their dependencies. The third line will install haproxy.

The procedure for installation of nodejs v0.10.25 software on Ubuntu 14.04 LTS operating system is as follows.This line in the will install nodejs from the default ubuntu package repositories. It is not the latest version but is quite stable and provides consistent experience across multiple servers.

For implementing load balancing a network was created with three servers with different address and connected haproxy to the three servers. The client knows the IP address of only the HAProxy server. The client sends HTTP requests to the HAProxy server that are directed by(using load balancing) the HAProxy server to the connected servers. The connected servers reply with a message consisting of IP address/port of the server, along with the request headers received in the HTTP request to these listeners.

To setup the three Node.js servers create a file called server.js in /srv/. Copy the following code to /srv/server.js. To start the servers run ‘nodejs server.js’ in the /srv/ directory.

fig3. server.js code for setting up three servers

```
//        File /srv/server.js
var http = require('http');

function serve(ip, port)
{
    http.createServer(function (req, res) {
      res.writeHead(200, {'Content-Type': 'text/plain'});
      res.write(JSON.stringify(req.headers));
      res.end("\nThere's no place like "+ip+":"+port+"\n");
    }).listen(port, ip);
    console.log('Server running at http://'+ip+':'+port+'/');
}

// Create three servers for
// the load balancer, listening on any
// network on the following three ports
serve('0.0.0.0', 9000);
serve('0.0.0.0', 9001);
serve('0.0.0.0', 9002);
```

In fig3. the serve function takes the ip address and the port number as input and creates a server that listens passively at that socket address. It responds to HTTP requests with valid response(code 200) and includes the request header(req.headers) and the ip address and port number(res.end("\n There's no place like " + ip + ":" + port + "\n")) of the responding server. To configure HAProxy to load balance the client requests open the /etc/haproxy/haproxy.cfg file and copy the following code into it:

fig4. System configuration for HaProxy

```
global
    log /dev/log    local0
    log /dev/log    local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin
    stats timeout 30s
    user haproxy
    group haproxy
    daemon

    # Default SSL material locations
    ca-base /etc/ssl/certs
    crt-base /etc/ssl/private

    # Default ciphers to use on SSL-enabled listening sockets.
    # For more information, see ciphers(1SSL).
    ssl-default-bind-ciphers kEECDH+aRSA+AES:kRSA+AES:+AES256:RC4-
SHA:!kEDH:!LOW:!EXP:!MD5:!aNULL:!eNULL
```

The code present in fig4 is explained below:**GLOBAL:** Within the global section we see that HAproxy is run as the user/group/haproxy. Running as a separate system user/group provides some extra avenues for increasing security through user/group permissions. It sets the default string describing the list of cipher algorithms ("cipher suite") that are negotiated during the SSL/TLS handshake for all "bind" lines which do not explicitly define theirs.

fig5. Timeout and Error reporting options

```
defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        timeout connect 5000
        timeout client  50000
        timeout server  50000
        errorfile 400 /etc/haproxy/errors/400.http
        errorfile 403 /etc/haproxy/errors/403.http
        errorfile 408 /etc/haproxy/errors/408.http
        errorfile 500 /etc/haproxy/errors/500.http
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http
```

The code contained in fig5 is explained below:

DEFAULTS: Within the default section we have some logging and timeout options and locations of files that contains errors response.

HAproxy gives you the the option to turn off access logs in each web node, or conversely, turning logs off at the load balancer while having them on within each web server.

fig6. Load balancing configuration of HaProxy

```
frontend localnodes
    bind *:80
    mode http
    default_backend nodes

backend nodes
    mode http
    balance roundrobin
    option forwardfor
    http-request set-header X-Forwarded-Port %[dst_port]
    http-request add-header X-Forwarded-Proto https if { ssl_fc }
    option httpchk HEAD / HTTP/1.1\r\nHost:localhost
    server web01 127.0.0.1:9000 check
    server web02 127.0.0.1:9001 check
    server web03 127.0.0.1:9002 check

listen stats *:1936
    stats enable
    stats uri /
    stats hide-version
    stats auth someuser:password
```

The code contained in fig 6 is explained below:

To balance the traffic between our three HTTP listeners we set some options within HAproxy.

1. **frontend** - where HAProxy listens to connections
2. **backend** - Where HAProxy sends incoming connections
3. **stats** - Optionally, setup HAProxy web tool for monitoring the load balancer and its nodes

1. Frontend Configuration

The **frontend** has been named ‘localnodes’.

bind *:80 - We’ve bound this frontend to all network interfaces on port 80. HAProxy will listen on port 80 on each available network for new HTTP connections.

mode http - This is listening for HTTP connections. HAProxy can handle lower-level TCP connections as well, which is useful for load balancing things like MySQL read databases, if you setup database replication.

default_backend nodes - This frontend should use default backend named nodes.

. Backend Configuration

mode http - This will pass the HTTP requests to the servers listed

balance roundrobin - Use the roundrobin strategy for distributing load amongst the servers. The balance option allows you to specify the load balancing algorithm that you want to implement. You can specify multiple options in it like uri, source etc.

option forwardfor - Adds the X-Forwarded-For header so our applications can get the clients actually IP address. Without this, our application would instead see every incoming request as coming from the load balancer's IP address.

server web01 127.0.0.1:9000 check - This specifies the server to which HAProxy connects and does the load balancing on. The check option makes HAProxy perform out of band health checks on the server and check if the server is available to process requests. Default health check is to try to establish a TCP connection with the server.

Stats Configuration

HAProxy comes with a web interface for monitoring the load balancer and the servers it is setup to use.

listen stats *: 1936 - Use the listen directive, name it stats and have it listen on port 1936.

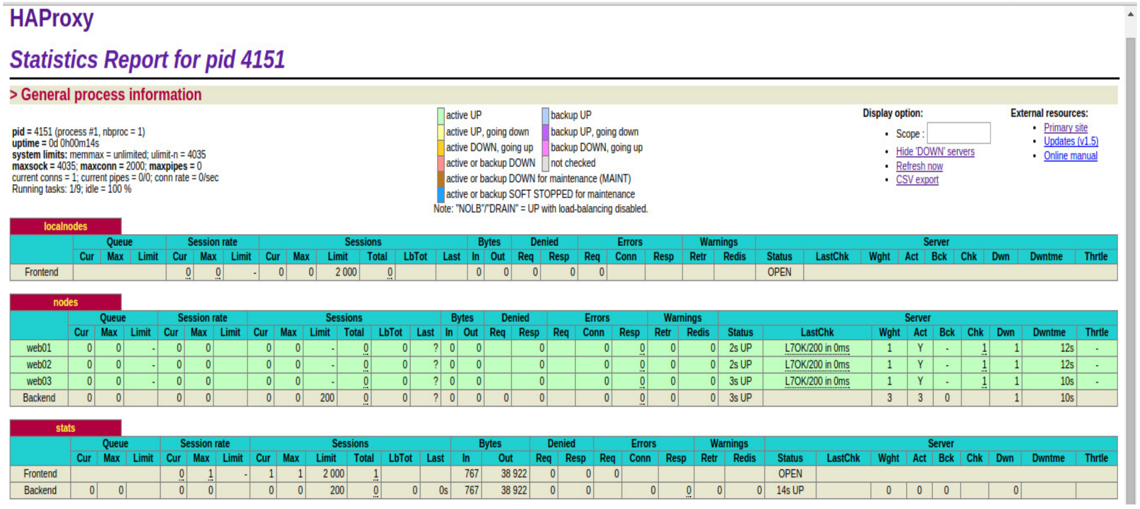
stats enable - Enable the stats monitoring dashboard

stats uri / - The URI to reach it is just / (on port 1936)

stats hide-version - Hide the version of HAProxy used

stats auth someuser:password - Use HTTP basic authentication, with the set username and password. In this example, the username is someuser and the password is just password. Don't use that in production - in fact, make sure your firewall blocks external HTTP access to that port.

Fig7 contains a snapshot of the haproxy stats page taken with three servers up and round robin algorithm implemented.



HAProxy

Statistics Report for pid 4151

> General process information

pid = 4151 (process #1, nbproc = 1)
uptime = 0d 0h00m14s
system limits: memmax = unlimited; ulimit-n = 4035
maxsock = 4035; **maxconn =** 2000; **maxpipes =** 0
current conns = 1; current pipes = 0/0; conn rate = 0/sec
Running tasks: 1/9; idle = 100 %

localnodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	2 000	0			0	0	0	0	0					OPEN								

nodes

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
web01	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	2s UP	L7OK/200 in 0ms	1	Y	-	1	1	12s	-
web02	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	2s UP	L7OK/200 in 0ms	1	Y	-	1	1	12s	-
web03	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	3s UP	L7OK/200 in 0ms	1	Y	-	1	1	10s	-
Backend	0	0		0	0		0	0	200	0	0	?	0	0	0	0		0	0	0	0	3s UP		3	3	0		1	10s	

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server							
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	1	1	2 000	1			767	38 922	0	0	0					OPEN								
Backend	0	0		0	0		0	0	200	0	0	0s	767	38 922	0	0		0	0	0	0	14s UP		0	0	0		0		

fig7.

The stats has three tables localnodes(frontend), nodes(backend) and the stats table. The tables show information about number of bytes that arrived at the in port and number of bytes that were forwarded for each server. They also show the number of bytes that were denied and the number of errors in request, connection and response. web01, web02, web03 are the names of the three servers on which haproxy is doing load balancing. Session rate is the number of new sessions per second. Under sessions, cur is the current number of sessions, max is the maximum concurrent sessions, total is the total number of sessions since haproxy was restarted. Under server, status indicates the current status of the server and for how long it has been there. LastChk indicates the result on the last check by haproxy on the server(Here Layer 7 check has been performed with (200)positive response). Chk indicates the number of failed checks, dwntime indicates the amount of time server has been down. Under queue, cur is the current size of the queue, max is the maximum capacity of the queue.

Conclusion

It can be seen that load balancing using round robin(optionally weighted round robin) is simple and resilient and the load distribution remains fair in all the situations.It can also be seen that least connections can be useful when there is a collection of servers with similar performance, it is effective in smoothing distribution when a server becomes bogged down. Source Ip hash is useful if it’s important that a client should connect to a session that is still active after a disconnection and reconnection. Uri hash allows the same server to handle the same server endpoints, so can be used by proxy caches and anti-virus proxies to maximize the cache hit rate. In general load balancing can be considered as one of the most simple and elegant methods for distributing the load on servers.

Future Work

Additional comparison of algorithms could be carried out on a simulated network representing real time traffic. The algorithms could be studied to examine what features in the algorithms are mainly responsible for balancing the traffic. Additional analysis could be done on how features of various algorithms could be combined to build more advanced load balancing algorithms. Some work could be done on the IP Source algorithm, on how that algorithm could be modified to implement anycast routing.

REFERENCES

[1] M. Mannan and P. C. Van Oorschot, “Secure Public Instant Messaging: A Survey”, in Privacy, Security and Trust, 2004.

[2] Declan McCullagh, “How safe is instant messaging? A security and privacy survey”, CNET News website. http://news.cnet.com/8301-13578_3-9962106-38.html

[3] H. Kikuchi, M. Tada, and S. Nakanishi, “Secure instant messaging protocol preserving confidentiality against administrator,” in 18th International Conference on Advanced Information Networking and Applications, 2004 (AINA 2004), vol. 2, pp. 27- 30.

[4] D. Boneh and R. J. Lipton, “A revocable backup system,” in Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6, San Jose, California, 1996, pp. 91–96.

[5] R. Perlman, “The Ephemerizer: Making Data Disappear”, Journal of Information System Security 1(1), 51–68 (2005)

[6] R. Perlman, “File system design with assured delete”, In SISW 2005 Proceedings of the Third IEEE International Security in Storage Workshop, pp. 83–88. IEEE Computer Society, Los Alamitos (2005)