Anycast as a Load Balancing feature

Fernanda Weiden <nanda@google.com>, Google Switzerland GmbH
Peter Frost <pfrost@google.com>, Google Switzerland GmbH

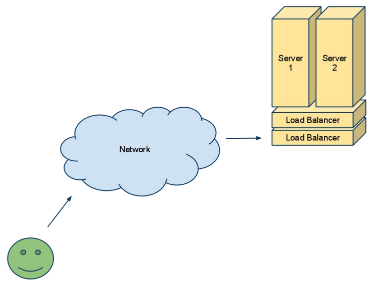
Tags: Load balancing, anycast, high availability, case study.

Abstract

Our IT organization is made up of many sub-teams, each providing a service such as DNS, LDAP, HTTP proxy, and so on. Each one is deployed globally, using their own replication mechanisms. Our team provides Load Balancing and failover services in a way that other teams can use without having to manage the underlying technology. We recently added Anycast as a service we offer to other teams that need to be able to failover between Load Balancers. While Anycast is complex and mysterious to many systems administrators, our architecture provides the service in a way that the other teams do not need to worry about the details. They simply provide the service behind Load Balancers they currently use, with an additional virtual IP address. This paper describes how Anycast works, it's benefits, and the architecture we used to provide Anycast failover as a service.

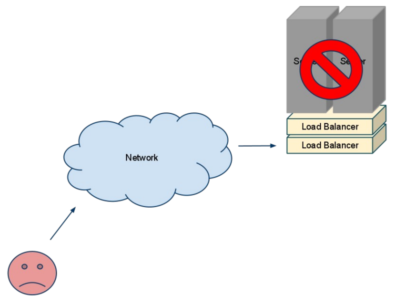
Introduction

The most natural way to think about Load Balancing, is to put as many service replicas as required in your server room, and have a Load Balancer distribute the load amongst them. To increase reliability, Load Balancers are usually deployed in High Availability pairs, and we assume this to be the case throughout this paper. A regular Load Balancing scenario would look like *Drawing 1*.



Drawing 1: Regular Load Balancing Scenario

The above is already an improvement in reliability, but it can go further. Imagine a disaster scenario where the users are still active and requesting the service and so is the Load Balancer, but all the backends for a specific service are not. This solution in itself wouldn't solve the problem (*see: Drawing 2*).

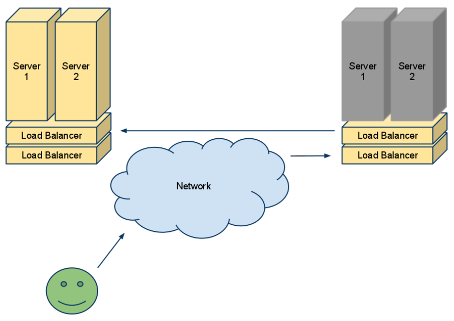


Drawing 2: Failure mode

A better design would automatically redirect all those clients to another location (or server room), making the process as transparent as possible. A way to accomplish this is to identify the nearest secondary location, and configure the Load Balancer to proxy or redirect all the user requests there, until the local service is re-established. Most load balancing products offer automatic redirection as depicted in the right (*see: Drawing 3*).

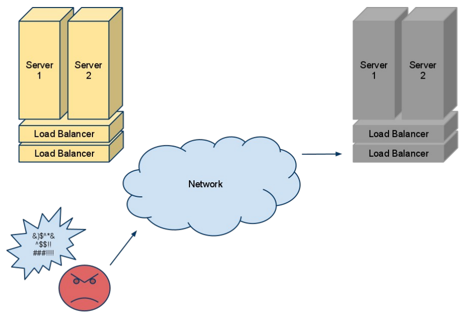
But what if the Load Balancers are also not available? (*see: Drawing 4*) How can we guarantee that the users will be able to reach another instance of the service? How can that be accomplished in the least intrusive way?

There are many ways to solve this problem, depending on the specifics of each implementation. One possibility would be to update DNS records for the services, so users can now reach the service in a different location.



Drawing 3: Remote failover

Potentially, this DNS update can be automated but there has to be a mechanism to check service status in other locations and keep track of their state so the system knows where to send users in case of a failure. Considering that services are sometimes deployed in hundreds of locations, it would not be effective to have a central place collecting all the information about services, so the DNS update mechanism would need to be distributed to as many locations as the service is deployed. We consider this a non optimal solution, since there is the possibility to integrate the monitoring and automatic failover in the existing Load Balancing infrastructure.



Drawing 4: Remote failover - failure mode

DNS TTL can also be a burden. Sometimes it is not possible to use very small TTLs and the time it takes to propagate DNS changes would still be downtime from the users' perspective. Once your system is back, there is again the need to update DNS entries to point users back to the original location.

Basics of Anycast

Anycast is a network routing technique where many hosts have the exact same IP address. Clients trying to reach that IP address are routed to the nearest host. If these duplicate hosts all provide the same service, the clients simply receive the service from the host topologically nearest.

Anycast per se doesn't have information on service specific health status, which might result in requests being sent to locations which do not have a healthy instance of the service running. It is then necessary to think about service specific healthchecks. If a given service has about 200 different instances, managing healthchecks and the Border Gateway Protocol 4 (BGP) configuration for each of those instances can be very complicated.

Our implementation

We use Anycast for failover between Load Balancing clusters, providing the benefits of Anycast to any service behind our Load Balancers.

This reduces a lot the network environment complexity, given the reduced number of machines advertising routes. Our solution uses BGP because it allows creation of a hierarchy for the route advertising, but other protocols work as well. Using Anycast, there is no longer need for remote failover using proxies, providing a cleaner solution since the client connects directly to the failover location, whilst proxying usually makes you lose the client identifying information. It also saves the proxy overhead between servers and users.

Our team provides Load Balancing as a service, making it completely separate from the specific service setup. Multiple services can profit from the same Load Balancing infrastructure, and growing the number of replicas of a service does not increase the complexity of the network design, since there are a controlled number of route advertisers.

Another advantage of having Load Balancers as Anycast peers is the reduced number of routing changes, because the Load Balancer combines multiple instances of a service into one VIP. That has been one of the concerns regarding Anycast deployments. Having the Load Balancer deal with service specific state healthchecks makes it possible to deploy Anycast not only for UDP based services, but also for TCP based services.

We configured the network environment so there is one subnet reserved for all Anycast virtual IPs (VIPs). The routers are configured to accept /32 route advertisements in that subnet from the Load Balancers. This allows the implementation of protection against misconfiguration by using ACLs which only allow routes from the specified subnet, preventing accidental takeover of IP space.

Anycast VIPs can be configured in addition to the normal VIPs on the same Load Balancers.

Software used for this implementation

All our Load Balancers are deployed in high availability pairs to protect against single machine failure. For this we used Heartbeat, from Linux-HA project[1], which is a cluster resource management software. Heartbeat brings network interfaces and backend management software up and down. These are all configured as heartbeat resources.

For backend monitoring and failover, we use ldirectord[4]. Ldirectord runs healthchecks against the backends for each of our VIPs, adding or removing from the Load Balancing pool service instances that changes health state. It can also redirect all the connections to a different location in case of failure in all backends, using the fallback option.

We added a feature to ldirectord, implementing a fallback command configuration: when the last of the local backends goes down, it triggers this command. We use that to bring the Anycast IP address up and down based on backend status.

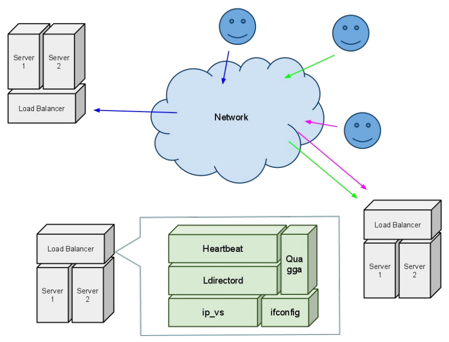
Ldirectord communicates directly with ifconfig (to bring IPs up and down) and ip_vs (via ipvsadm) to add and remove service backends from the Load Balancing pool.

ip_vs is the Linux Kernel module for Load Balancing[0]. It currently supports tunnelling, half NAT and Direct Routing (DR) modes. In our setup, all VIPs are configured using DR.

Quagga[2] is a network routing software package, allowing a GNU/Linux system to participate in network routing protocols. In our solution, we use the BGP implementation to enable our Load Balancers to advertise BGP routes to the network devices.

We allocate an IP for each service, and create the standard configuration for LVS to bring the VIPs up, and use the feature we added to ldirectord to bring the Anycast IP network interface up or down, depending on the state of the backends. If all the backends are down, ldirectord will bring the IP from that VIP down, and Quagga will immediately let the routers know so they withdraw the route to that VIP.

The picture on the right illustrates the network architecture, and details on the software deployment.



Drawing 5: Load Balancing with Anycast - software architecture

Adding new services to the setup

Services can be added to Anycast by simply configuring their backends in a VIP on an Anycast enabled Load Balancer. In our case, that means Heartbeat and ldirectord configuration.

The network configuration will already be in place, significantly reducing the barrier of entry for new services. Expanding a service to new locations follows exactly the same process, the Anycast routing will take care of sending user traffic to the new, nearer load balancer setup.

Services currently using this setup

- DNS
- HTTP proxy
- Radius
- Web SSO
- NTP
- LDAP (in test now).

Failure modes and recovery times

All of the mentioned recovery times take into consideration our specific Anycast deployment. Environments with different timeouts and configuration parameters can have different response and recovery times. The route propagation time is less than 1 second, and we have set a 30 second “dead timer” for the routers to consider the BGP peer dead.

Cleanly stopping the BGP peering service creates a outage of less than 1 second for the services as the routes update. In the case of all service backends becoming unavailable, it will take the service specific healthcheck time plus the <1s route propagation delay for recovery.

In a sudden network or power failure at the location, it will take the time for the “dead timer” to expire plus the small route propagation delay for recovery.

Conclusion

Moving Anycast routing configuration to a managed load balancer service minimizes the work and complexity required for service configuration while providing them with fast, automatic, distance aware failover. It also helps reduce load and complexity of network infrastructure by aggregating service advertisements into one peering point per site and reducing the rate of routing changes to complete site failures only.

References

[0] The Linux Virtual Server Project, http://www.linuxvirtualserver.org
[1] High Availability, http://www.linux-ha.org
[2] Quagga, a software routing suite, http://www.quagga.net
[3] RFC1771 - A Border Gateway Protocol 4, http://www.faqs.org/rfcs/rfc1771.html
[4] Ldirectord, http://www.vergenet.net/linux/ldirectord/