DNS LOAD BALANCING AND FAILOVER MECHANISM AT CERN

V. Bahyl, N. Garfield, CERN, Geneva, Switzerland

Abstract

Availability approaching 100% and response time converging to 0 are two factors that users expect of any system they interact with. Even if the real importance of these factors is a function of the size and nature of the project, todays users are rarely tolerant of performance issues with system of any size.

Commercial solutions for load balancing and failover are plentiful. Citrix NetScaler, Foundry ServerIron series, Coyote Point Systems Equalizer and Cisco Catalyst SLB switches, to name just a few, all offer industry standard approaches to these problems. Their solutions are optimized for standard protocol services such as HTTP, FTP or SSH but it remains difficult to extend them for other kinds of application. In addition to this, the granularity of their failover mechanisms are per node and not per application daemon, as is often required. Moreover, the pricing of these devices for small projects is uneconomical.

This paper describes the design and implementation of the DNS load balancing and failover mechanism currently used at CERN. Our system is based around SNMP, which is used as the transport layer for state information about the server nodes. A central decision making service collates this information and selects the best candidate(s) for the service. IP addresses of the chosen nodes are updated in DNS using the DynDNS mechanism.

The load balancing feature of our system is used for variety of standard protocols (including HTTP, SSH, (Grid)FTP, SRM) while the (easily extendable) failover mechanism adds support for applications like CVS and databases. The scale, in terms of the number of nodes, of the supported services ranges from a couple (2-4), up to around 100. The best known services using this mechanism at CERN are LXPLUS and CASTORGRID.

This paper also explains the advantages and disadvantages of our system, and advice is given about when it is appropriate to be used.

Last, but not least, given the fact that all components of our system are build around freely available open source products, our solution should be especially interesting in low resource locations.

INTRODUCTION

There are several approaches to improving system availability. Other than increasing mean time between failure (MTBF) at the hardware level, the careful setup of operating system parameters and a redundant network architecture, load-balancing and failover clustering are the two main techniques for improving availability. Load-balancing addresses the problem of resource consumption while failover simply solves the problem of whether or not a node exists from the client perspective.

CERN's network is large and complex. In order to aid the operation of the network *a database system* is employed to ensure the coherence of the network infrastructure. Typically a network management system strives to correlate information about how connected nodes are integrated into the network infrastructure by relating the device to sockets, cables, switch ports, IP subnets and finally IP addresses and host names. Network prefixes, interface names and IP addresses are extracted from the database and published in the DNS as forward and reverse zones. Updates are not required frequently, in general approximately every few hours or when needed. This model promotes DNS stability in the forward and reverse zones, but contains no knowledge of the use or purpose of the devices that are connected to the network. The database is unaware of the actual computer services running on the network. However, DNS is a ubiquitous, standardized and globally accessible database making it the ideal medium for the publication of services.

Reflecting the desire to promote a stable network infrastructure, a long Time to Live (TTL) value is configured in the CERN DNS namespace (1 hour). If the record changes on the server side the client will be unaware of the change until the TTL has passed.

The conditions of the CERN environment are such that while official services must be provided uninterrupted, it must also be possible to transparently migrate machines in and out of a particular service, perhaps for upgrades, hardware interventions or because they are simply required in another service (i.e. failover). Another reason could simply be because a machine might temporarily be under high load and it would be convenient to replace it with a less utilized machine (i.e. load-balancing).

For a long time DNS has provided a simple and zero-cost load-distribution system – DNS round-robin. DNS requests are distributed over several IP addresses by writing a list of host records with identical names into a DNS domain. Each time a request is made for a particular name the DNS server responds with the next address in the list until the last address is reached when the server resumes at the beginning of the list. At CERN the primary advantage of DNS round robin is that it is straightforward to integrate into the existing DNS architecture. An additional benefit of DNS round-robin load-distribution is that its operation is independent of the network architecture thereby operating with a low overhead.

The disadvantage of the DNS round robin system is that it is unaware of the state or reachability of the published nodes – it cannot perform intelligent failover.

In an ideal solution, providing failover, the DNS system would have to allow for the immediate update (withdrawal and addition) of DNS records. Clearly this is a problem to a DNS-based failover system as it would effectively take the TTL time to withdraw a published device from service.

At the network level there are primarily two different approaches to load-balancing. Windows 2003 Server Network Load Balancing (NLB) is a useful method where network load is the metric used to make packet forwarding decisions. The advantage of NLB is that it is a turnkey solution for building small clusters with network state awareness. The disadvantages of Windows NLB are that it is proprietary, has limited scalability beyond a single Ethernet segment and IP subnet and is not application aware.

The second method is layer 4 switching of which dedicated examples are Cisco's CSS and the open source Linux Virtual Server (LVS). Layer 4 switching functionality is also available on enterprise multi-layer switches (Cisco 6500 and Enterasys Xpedition). A layer 4 switch hides a cluster of application IP addresses behind a single IP address. The single IP address is then published in DNS and all clients send packets to that address. The router then applies administrator configured tests in order to choose the best destination to which it should forward packets. The disadvantages of this approach are: the tests are rather simplistic (e.g. a ping test or URL response test), the router itself becomes a single point of failure thereby passing the challenge of high-availability to the router manufacturer and all the devices of the cluster must belong to a directly connected subnet. For a large campus network layer 4 switches would be an expensive inflexible solution to the problem of load balancing. The actual throughput of such a system will not scale – ideally for a load-distribution system spreading the load over several switches has obvious advantages.

Having concluded that hardware based load-balancing products were neither adequate for CERN's needs nor appropriate for its network architecture it was decided to reconsider DNS as a medium for managing high-availability applications. The first part of this paper describes how instantaneous DNS updates can be used to operate high availability services. The second part describes an arbitration process within a large cluster of machines to select a small group of machines to be placed in service using Dynamic DNS updates (DDNS). The combination of the two techniques allows for a simple form of *application load balancing*.

PUBLICATION OF SERVICES IN DNS

For reasons explained above it appears impractical to publish services in DNS while it is required to be able to change records frequently. Efficient resource management (both host and network) has always been a major goal in DNS architecture. It is clear that frequent updates of large zones consume server resources while large zones also increase the server request rate. Both the distribution via delegation (see below) and the use of caching requests for the duration of a configured TTL are used to decrease the request rate to a DNS server.

Name space segmentation is one of the reasons for the success of DNS. The ability to segment the name space is called delegation – the administrator has the possibility to create sub-domains of domains. A sub-domain may be realized in two ways: remote delegation to a service manager's authoritative DNS server or delegation to local sub-domain. The first option was initially used at CERN. Operational complexity made the solution difficult to provide a reliable end-to-end service especially if support is required for internal and external network views. In practical terms, multiple views are unavoidable when the use of unique SOA and NS records in a particular view are required. The second option is desirable as it eliminates duplication of effort, simplifies the support process and scales easily to multiple views.

The operational model is simple. Services are mapped directly onto sub-domains. For example the interactive service LXPLUS is a sub-domain lxplus.cern.ch. This model has two very convenient points – the sub-domain has its own unique properties such as TTL and resource records (RR) while allowing the updates to be performed independently of the parent domain.

Basic load balancing is possible with DNS round-robin. It is enabled by writing multiple blank host records into a particular zone. Intelligent load-balancing and failover require an additional monitoring system (arbiter – see below) to add and withdraw host records when required. To provide simple failover the arbiter writes a single blank host record in the zone. On failure of the node it updates the host record with a reachable node. A load-balancing cluster writes multiple records to the zone monitoring the state of each node in the cluster and updating the records as necessary.

In order to centralize DNS updates of a sub-domain the service manager needs a way to notify the central name service of his changes. Again, there are (at least) two ways to approach this. The first way is to allow the existing system to access multiple data sources to construct the delegated zones. The problem with this method is that a common signalling system is required while needing to agree a standardization of the data source. When using views it is necessary to have all the views on a single master server otherwise a signal has to be sent for each view. CERN uses multiple views in the DNS to help in the logical partitioning of the network. A zone describes a sub-domain in a particular view. The architecture of today's DNS defines a master server as the source of the zone data whereas a slave is a name server which contains a copy of the zone data originated from the master (primary and secondary servers have no meaning). Slave servers may or may not be authoritative in a zone – it is simply the NS records and the SOA which denote the authoritative servers in the zone.

The database and signalling method has been used at CERN successfully with larger zones (a few thousand entries). However, the combined process was found to be too slow for frequently updated services and one may also note that additional points of failure are present in the system.

DYNAMIC DNS SERVICE MODEL

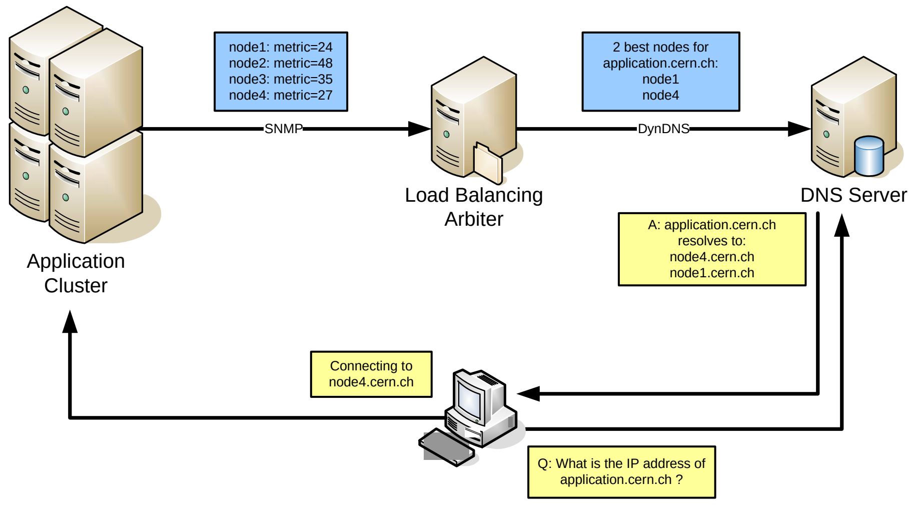
Continuing the discussion above the second method is to use DDNS update which is well suited to small frequently changed zones.

Fortunately DNS has been continuously evolving over the years and the DDNS standard recognises the need to make DNS more accessible by allowing RPC incremental updates of DNS records. To further reduce the consumption of resources the Incremental Zone transfer (IXFR) mechanism allows the incremental update of records between the master and its slaves.

APPLICATION LOAD BALANCING SYSTEM

In the event of node failure DNS round robin is not sufficient because unreachable nodes are not withdrawn automatically. Instead a more complex DNS record update system is required.

A diagram of the load balancing system is shown in Figure 1. It consists of 3 main components: application cluster nodes, arbiter and the DNS. The text boxes outline the most important processes happening in the cluster. The purpose of the arbiter is to collect metric values from all nodes in the application cluster, select the best candidates and update the DNS. Following an update when a client asks the DNS to resolve the hostname it will be referred to one of the best candidates.



node1: metric=24
node2: metric=48
node3: metric=35
node4: metric=27
SNMP
2 best nodes for application.cern.ch: node1 node4
DynDNS
Load Balancing Arbiter
DNS Server
Application Cluster
A: application.cern.ch resolves to: node4.cern.ch node1.cern.ch
Connecting to node4.cern.ch
Q: What is the IP address of application.cern.ch ?

Figure 1 – Overview of the load balancing system

Finally, the Transaction Signature standard (TSIG) can be used for authentication of DDNS update requests to the master. When running multiple views TSIG keys are essential to authenticate and access the various views available on the master server to which the updates are applied. The service managers have pairs of TSIG keys, one for each view, and those are used to authenticate him to any of his zones.

At CERN the TTL of all DDNS zones is set at 60 seconds, a value appropriate for the desired service failover time. The effective convergence failover time will be a maximum of TTL + arbiter (see below) poll interval (N.B. caching can be disabled by setting the TTL=0). Updates made to the master are propagated instantly to the slaves via IXFR. It was found that when running IXFR and DDNS the administrator must pay particular attention to the journal size of each zone – if the journal size reaches the file size limit of the server operating system the server will corrupt the journal and consume very large amounts of CPU – setting the journal-size at 10 MB has made a stable system. In situations where very high query rates are a concern it would be necessary to study the impact of journal-size on the reply saturation rate.

The following definitions are helpful to describe the arbitration process in detail:

N	number of all nodes in the application cluster
B	configurable number of best candidates that will resolve behind the DNS alias
R	number of nodes that returned positive non-zero metric value within time-out interval

THE ARBITER

The arbitration server process is the core of the whole system. It runs on a separate server machine and it periodically performs the following subtasks:

- Collects the metric values from the application cluster nodes
- Selects the best candidates following certain criteria
- Updates the central DNS records using DDNS

Data Collection and Selection of the Best Candidates

Each node taking part in the load balancing must report its state to the arbiter server. This reporting process is controlled by the server itself which periodically polls the information from the nodes over SNMP. The server issues

SNMP get calls and queries sequentially each node in the cluster. The interval between two scans of all nodes in the cluster is configurable (however, DNS updates are made at most once per minute). The value returned by the nodes is an integer number calculated individually by each node (see later section about metrics). Only positive values (> 0) that were returned by the hosts within a time-out interval (default 5 seconds) are considered, the nodes that reply create the R set.

Once the arbiter has queried all nodes in the application cluster it selects the best B candidates from the R set. Normally, B is a subset or equal to R, but if a serious problem affecting almost the whole cluster arises the following non-standard situations can occur:

Exceptional state	Description	Solution applied by the system
R < B	Number of nodes that replied is smaller than the number of best candidates that should be resolved by the DNS alias.	B = R
R = 0	There were no usable replies from any of the nodes in the application cluster.	System returns random B nodes from the group of N.

While the application load balancing configuration described above is most commonly used, the system also supports a simpler round robin mechanism where SNMP is replaced by ICMP ping. It then selects B nodes sequentially (if configured it will only consider those that replied to ping) from the group of N, rotating over N, allowing all nodes to participate in the load sharing.

DNS Updates

Once the best candidates have been selected, the arbiter updates the DNS for the given zone. This is done by Net::DNS Perl class using TSIG keys.

The actual sub process that updates the DNS could be easily externalized (i.e. provided by a separate binary) such that it can be performed outside of the arbitration process. This may be useful for allowing integration with site specific requirements (e.g. databases such as Oracle or even LDAP).

Active and Standby Arbiters

The arbitration server is protected against serious failures by a very simple active and standby design. There are 2 machines running the same arbiter daemon – active and standby and they both poll for the metric values from all nodes in all clusters via SNMP and both select the best candidates. Then, under normal operations, only the active server triggers the updates on the DNS. Since the machine where the active process runs is also a web server, it updates a local heartbeat file which is periodically fetched over HTTP and checked by the process on the standby server. If the time of the last update of the heartbeat file differs from the current time by more than 10 minutes or if the file is not accessible, the standby server starts to trigger DDNS updates. We are aware that this failover mechanism is quite simple and could be improved.

APPLICATION CLUSTER NODES

A cluster is defined as a group of nodes that offer the same functionality or run the same application. The application (i.e. service) is accessed not by connecting to each individual node but via client DNS resolution of the cluster DNS alias (a zone).

SNMP Daemon

The arbiter gathers information from the cluster nodes either by using SNMP or ICMP ping. If application load balancing is required each cluster node must run a local SNMP daemon configured to allow SNMP get calls to a private MIB OID. When the OID is called SNMP launches a standard binary (installed in bulk on many machines via an RPM package) which examines the operating state of its host and returns a metric value.

Metric

As described above, the handling of the mentioned MIB OID is passed to the external program which is in our system called the *load balancing metric*. This small C binary examines the actual running conditions of the system on the node, translates that into an integer number and it prints its value on the standard output which is then passed to the arbiter over SNMP. The purpose of this metric program is to reflect as much as possible the system load on the node such that users can be offered the set of least loaded nodes at any instant in time.

The list of items that the metric binary could include in the calculation is configurable. Depending on their return value, they can be divided into 2 groups:

- System checks that return a Boolean value
- System assessment indicators that (usually) return a positive number (e.g. system load from *uptime*)

System checks currently available to calculate the metric are:

- Check whether the following daemons are listening (uses /proc/net/tcp and /proc/net/tcp6) on their respective ports: FTP daemon (port 21), SSH daemon (port 22), WWW daemon (port 80), GridFTP daemon (port 2811)
- Check whether the /afs/cern.ch/user/ path is accessible (indicates AFS problems on a node)?
- Check whether /tmp is not full ?
- Check whether the file /etc/nologin is present (if yes, the node is closed for interactive user logins)?

As all these checks return a Boolean value, if one is not fulfilled, the program exits immediately with a negative value which means that the node is marked as a bad candidate.

System assessment indicators that are considered by the metric when evaluating the running conditions on the node are: system CPU load (the most important factor), number of users currently logged in, swapping activity, number of running X sessions (understands KDE and GNOME sessions only). It is the return values of these indicators that compose the final metric formula which quantifies the load of the system. The formula is CERN specific, it has been found empirically and its discussion is beyond the scope of this paper.

From the implementation point of view, in many cases described above, the /proc pseudo GNU/Linux filesystem serves as the source of the information. The disadvantage of this method is that not only it makes the metric binary GNU/Linux kernel specific but also the above described Boolean checks are often performed in parallel with the CERN central monitoring system (LEMON). This duplication is unnecessary and a new binary is being deployed which integrates with node LEMON sensors. However, this approach introduces latency into the detection of failures.

OPERATIONAL EXAMPLES

The above described load balancing system is used at CERN by many key services. These services differ not only in scale and purpose but also in the way how they benefit from our system. The following are the most important applications:

Interactive Login Service

The main client of the load balancing at CERN is the interactive service called LXPLUS (LinuX based Public Login UNIX Service). It allows users to connect via SSH to one of many (O(10)) lxplusXXX.cern.ch nodes and work interactively. In this usage pattern, users with their SSH clients resolve lxplus.cern.ch DNS alias once, connect to the offered IP address and stay connected to that node until they log out.

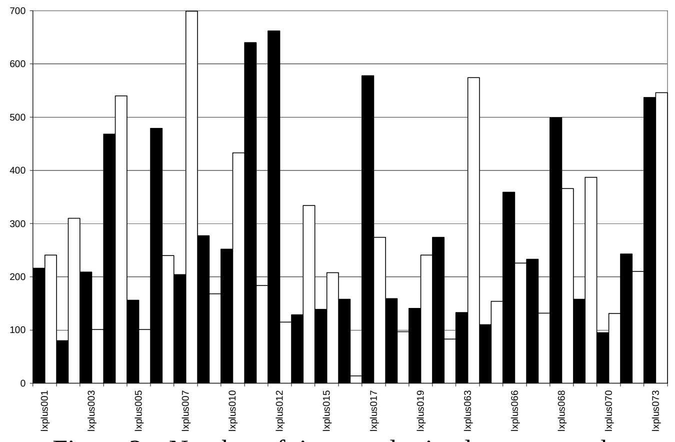


Figure 2 – Number of times nodes in cluster were chosen

The bar graph in the Figure 2 shows a statistical record of the selection of each node in the LXPLUS cluster for the period of 30 days. The abscissa indicates the hostname and the ordinate displays the number of times the given node was selected as part of the cluster DNS alias (lxplus.cern.ch). The bar graph displays two bars for each node. The dark bar is for the first 15 days, the light bar is for the second 15 days. It is clear from the variation in the height of the bars that the system is not distributing sessions fairly. However, over two 15 day periods a single machine was not preferred over another which indicates that there is not a systematic selection problem.

Stateless Applications

The simplest use of the application load balancing system is one where the state of the connection is not relevant in processing transactions on the server side because any subsequent connections will resolve the IP address of *any* of the nodes published under the DNS alias name (e.g. a web server providing static content or a (Grid)FTP server). The application load balancing system ensures that the subset of resolvable IP addresses always points to the least loaded nodes and then the DNS round robin will ensure load distribution from the client side.

State Aware Applications

The problem with DNS round robin is that it has no notion of the state of the application. Therefore the ideal server process of the state aware application should tell the client to make all following connections to a specific IP address or host name effectively substituting the registered cluster DNS alias for one of its member nodes.

Continuing with the example of a web service there are two ways of solving this problem: the first is to use the ServerName directive of the Apache daemon while the second method is to write the application to substitute the server name into all referenced URLs embedded in the returned HTML.

CONCLUSION

Using DDNS switching it is possible to automate intelligent load-balancing and failover systems ranging from a two node active/standby simple failover system using ICMP ping to a complex application cluster system containing many nodes. Common to both systems is the need for an arbiter, which uses metric values derived from operating system or application level parameters to evaluate whether a node should be included or excluded in a list of available nodes. The arbiter then updates the published list of nodes in DNS withdrawing or adding cluster nodes as required.

The system is built around Open Source tools and as such is sufficiently general to be easily adapted by institutions other than CERN.

ACKNOWLEDGEMENTS

For their direct or indirect contribution to the development of this load balancing system, we would like to thank (in time order of their contribution): Dan Pop, Martin Murth, Solene Quelard, Thorsten Kleinwort, Teemu Sidoroff and Tim Bell. On the DNS side Alexander Karlov performed the initial evaluation of the integration of DDNS into the existing CERN DNS infrastructure.