White Paper

Load Balancing 101: The Evolution to Application Delivery Controllers

As load balancers continue evolving into today's Application Delivery Controllers (ADCs), it's easy to forget the basic problem for which load balancers were originally created—producing highly available, scalable, and predictable application services. ADCs' intelligent application routing, virtualized application services, and shared infrastructure deployments can obscure these original goals—but it's still critical to understand the fundamental role load balancing plays in ADCs.

by KJ (Ken) Salchow, Jr.
Sr. Manager, Technical Marketing and Syndication

Contents

Introduction

One of the unfortunate effects of the continued evolution of the load balancer into today's application delivery controller (ADC) is that it is often too easy to forget the basic problem for which load balancers were originally created—producing highly available, scalable, and predictable application services. We get too lost in the realm of intelligent application routing, virtualized application services, and shared infrastructure deployments to remember that none of these things are possible without a firm basis in basic load balancing technology. So how important is load balancing, and how do its effects lead to streamlined application delivery?

Load Balancing Drivers

The entire intent of load balancing is to create a system that virtualizes the "service" from the physical servers that actually run that service. A more basic definition is to balance the load across a bunch of physical servers and make those servers look like one great big server to the outside world. There are many reasons to do this, but the primary drivers can be summarized as "scalability," "high availability," and "predictability."

Scalability is the capability of dynamically, or easily, adapting to increased load without impacting existing performance. Service virtualization presented an interesting opportunity for scalability; if the service, or the point of user contact, was separated from the actual servers, scaling of the application would simply mean adding more servers which would not be visible to the end user.

High Availability (HA) is the capability of a site to remain available and accessible even during the failure of one or more systems. Service virtualization also presented an opportunity for HA; if the point of user contact was separated from the actual servers, the failure of an individual server would not render the entire application unavailable.

Predictability is a little less clear as it represents pieces of HA as well as some lessons learned along the way. However, predictability can best be described as the capability of having confidence and control in how the services are being delivered and when they are being delivered in regards to availability, performance, and so on.

Load Balancing: A Historical Perspective

Back in the early days of the commercial Internet, many would-be dot-com millionaires discovered a serious problem in their plans. Mainframes didn't have web server software (not until the AS/400e, anyway) and even if they did, they couldn't afford them on their start-up budgets. What they could afford was standard, off-the-shelf server hardware from one of the ubiquitous PC manufacturers. The problem for most of them? There was no way that a single PC-based server was ever going to handle the amount of traffic their idea would generate and if it went down, they were offline and out of business. Fortunately, some of those folks actually had plans to make their millions by solving that particular problem; thus was born the load balancing market.

In the Beginning, There Was DNS

Before there were any commercially available, purpose-built load balancing devices, there were many attempts to utilize existing technology to achieve the goals of scalability and HA. The most prevalent, and still used, technology was DNS round-robin. Domain name system (DNS) is the service that translates human-readable names (www.example.com) into machine recognized IP addresses. DNS also provided a way in which each request for name resolution could be answered with multiple IP addresses in different order.

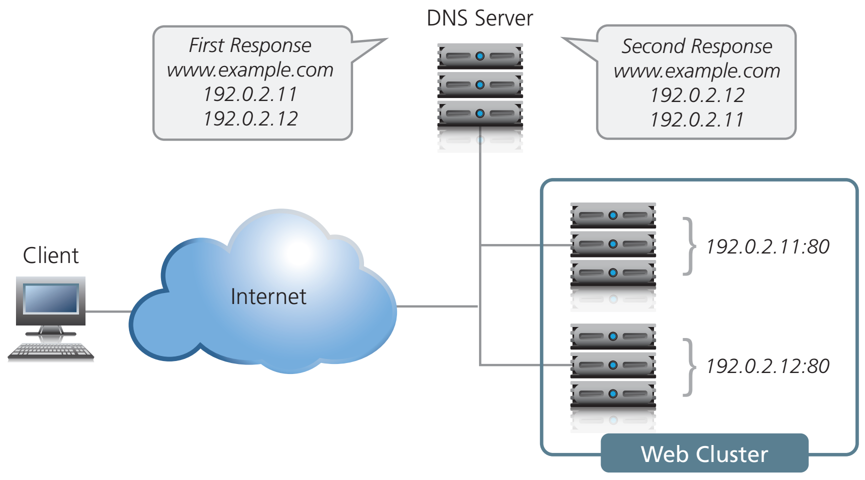

Figure 1: Basic DNS response for redundancy

The first time a user requested resolution for www.example.com, the DNS server would hand back multiple addresses (one for each server that hosted the application) in order, say 1, 2, and 3. The next time, the DNS server would give back the same addresses, but this time as 2, 3, and 1. This solution was simple and provided the basic characteristics of what customer were looking for by distributing users sequentially across multiple physical machines using the name as the virtualization point.

From a scalability standpoint, this solution worked remarkable well; probably the reason why derivatives of this method are still in use today particularly in regards to global load balancing or the distribution of load to different service points around the world. As the service needed to grow, all the business owner needed to do was add a new server, include its IP address in the DNS records, and voila, increased capacity. One note, however, is that DNS responses do have a maximum length that is typically allowed, so there is a potential to outgrow or scale beyond this solution.

This solution did little to improve HA. First off, DNS has no capability of knowing if the servers listed are actually working or not, so if a server became unavailable and a user tried to access it before the DNS administrators knew of the failure and removed it from the DNS list, they might get an IP address for a server that didn't work. In addition, clients tend to cache, or remember, name resolutions. This means that they don't always ask for a new IP address and simply go back to the server they used before—regardless of whether it is working and irrespective of the intention to virtualize and distribute load.

This solution also highlighted several additional needs in the arena of load balancing. As mentioned above, it became clear that any load balancing device needed the capability to automatically detect a physical server that had malfunctioned and dynamically remove it from the possible list of servers given to clients. Similarly, any good mechanism must be able to ensure that a client could not bypass load balancing due to caching or other means unless there was a good reason for it. More importantly, issues with intermediate DNS servers (which not only cached the original DNS entries but would themselves reorder the list of IPs before handing them out to clients) highlighted a striking difference between "load distribution" and "load balancing;" DNS round-robin provides uncontrolled distribution, but very poor balancing. Lastly, a new driver became apparent—predictability.

Predictability, if you remember, is the capability of having a high-level of confidence that you could know (or predict) to which server a user was going to be sent. While this relates to load balancing as compared to uncontrolled load distribution, it centered more on the idea of persistence. Persistence is the concept of making sure that a user doesn't get load balanced to a new server once a session has begun, or when the user resumes a previously suspended session. This is a very important issue that DNS round-robin has no capability of solving.

Proprietary Load Balancing in Software

One of the first purpose-built solutions to the load balancing problem was the development of load balancing capabilities built directly into the application software or the operating system (OS) of the application server. While there were as many different implementations as there were companies who developed them, most of the solutions revolved around basic network trickery. For example, one such solution had all of the servers in a cluster listen to a "cluster IP" in addition to their own physical IP address.

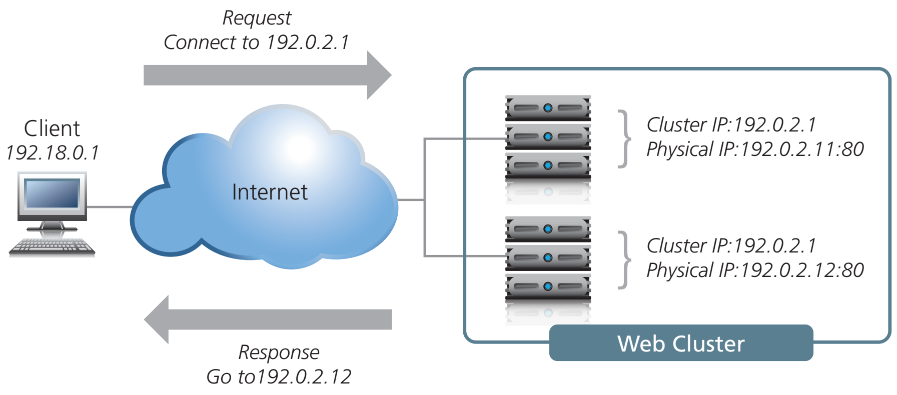

Figure 2: Proprietary cluster IP load balancing

When the user attempted to connect to the service, they connected to the cluster IP instead of to the physical IP of the server. Whichever server in the cluster responded to the connection request first would redirect them to a physical IP address (either their own or another system in the cluster) and the service session would start. One of the key benefits of this solution is that the application developers could use a variety of information to determine which physical IP address the client should connect to. For instance, they could have each server in the cluster maintain a count of how many sessions each clustered member was already servicing and have any new requests directed to the least utilized server.

Initially, the scalability of this solution was readily apparent. All you had to do was build a new server, add it to the cluster, and you grew the capacity of your application. Over time, however, the scalability of application-based load balancing came into question. Because the clustered members needed to stay in constant contact with each other concerning who the next connection should go to, the network traffic between the clustered members increased exponentially with each new server added to the cluster. After the cluster grew to a certain size (usually 5–10 hosts), this traffic began to impact end-user traffic as well as the processor utilization of the servers themselves. So, the scalability was great as long as you didn't need to exceed a small number of servers (incidentally, less than DNS round-robin could do).

HA was dramatically increased with these solutions. Because the clustered members were in constant communication with each other, and because the application developers could use their extensive application knowledge to know when a server was running correctly, this virtually eliminated the chance that a user would ever reach a server that was unable to service their request. It must be pointed out, however, that each iteration of intelligence-enabling HA characteristics had a corresponding server and network utilization impact, further limiting scalability. The other negative HA impact was in the realm of reliability. Many of the network tricks used to distribute traffic in these systems were complex and required considerable network-level monitoring; accordingly, they often encountered issues which affected the entire application and all traffic on the application network.

Predictability was also enhanced by these solutions. Since the application designers knew when and why users needed to be returned to the same server instead of being load balanced, they were able to embed logic that helped to ensure that users would stay persistent as long as needed. They also used the same "clustering" technology to replicate user state information between servers, eliminating many of the instances that required persistence in the first place. Lastly, because of their deep application knowledge, they were better able to develop load balancing algorithms based on the true health of the application instead of things like connections, which were not always a good indication of server load.

Besides the potential limitations on true scalability and issues with reliability, proprietary application-based load balancing also had one additional drawback—it was reliant on the application vendor to develop and maintain. The primary issue here is that not all applications provided load balancing, or clustering, technology and those that did often did not work with those provided by other application vendors. While there were several organizations that produced vendor-neutral, OS-level load balancing software, they unfortunately suffered from the same scalability issues. And without tight integration with the applications, these software "solutions" also experienced additional HA challenges.

Network-Based Load balancing Hardware

The second iteration of purpose-built load balancing came about as network-based appliances. These are the true founding fathers of today's Application Delivery Controllers. Because these boxes were application-neutral and resided outside of the application servers themselves, they could achieve their load balancing using much more straight-forward network techniques. In essence, these devices would present a virtual server address to the outside world and when users attempted to connect, it would forward the connection on the most appropriate real server doing bi-directional network address translation (NAT).

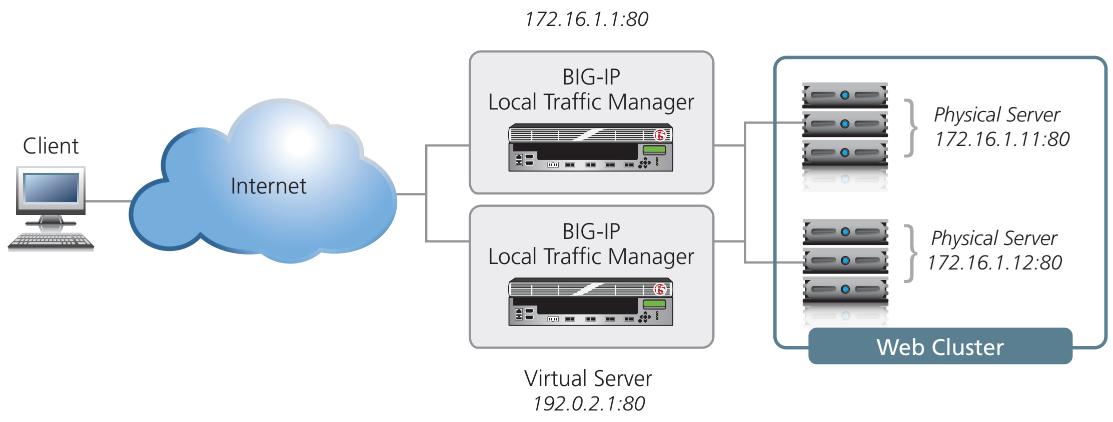

Figure 3: Load balancing with network-based hardware

The load balancer could control exactly which server received which connection and employed "health monitors" of increasing complexity to ensure that the application server (a real, physical server) was responding as needed; if not, it would automatically stop sending traffic to that server until it produced the desired response (indicating that the server was functioning properly). Although the health monitors were rarely as comprehensive as the ones built by the application developers themselves, the network-based hardware approach could provide at least basic load balancing services to nearly every application in a uniform, consistent manner—finally creating a truly virtualized service entry point unique to the application servers serving it.

Scalability with this solution was only limited by the throughput of the load balancing equipment and the networks attached to it. Health monitoring, while still potentially impacting the network, no longer grew exponentially because only the load balancer needed to maintain health information on the entire cluster, not each server. This reduced the overhead costs on the network and the servers providing additional headroom. It was not uncommon for organization replacing software-based load balancing with a hardware-based solution to see a dramatic drop in the utilization of their servers preventing them from having to purchase additional servers in the short-term and generating increased ROI in the long-term.

HA was also dramatically reinforced with a hardware-based solution. Granted, it did require these systems to be deployed as HA pairs to provide for their own fault tolerance, but simply reducing the complexity of the solution as well as providing application-impartial load balancing provided greater reliability and increased depth as a solution. Network-based load balancing hardware enabled the business owner to provide the high-levels of availability to all their applications instead of merely the select few with built-in load balancing.

Predictability was a core component added by the network-based load balancing hardware. Because the load balancing decisions were all deterministic (consisting of real-world measurements of connection load, response times, and so on) as opposed to the synthetic approach of most application-based solutions, it was much easier to predict where a new connection would be directed and much easier to manipulate. These devices were also capable of giving real-world usage and utilization statistics which provided insight for the capacity planning team and helped to document the results of load balancing operations. Interestingly enough, this solution reintroduced the positive potential of load distribution versus load balancing. Load balancing is an ideal goal if all your servers are identical; however, as a site grows and matures, that is often not the case. The added intelligence to create controlled load distribution (as opposed to the uncontrolled distribution of dynamic DNS) allowed business owners to finally use load distribution in a positive way, sending more connections to the bigger server and less to the smaller one.

The advent of the network-based load balancer ushered in a whole new era in the architecture of applications. HA discussions that once revolved around "uptime" quickly became arguments about the meaning of "available" (if a user has to wait 30 seconds for a response, is it available? What about one minute?). They also brought about new benefits for security and management like masking the true identity of application servers from the Internet community and providing the ability to "bleed" connections off of a server so it could be taken offline for maintenance without impacting users. This is the basis from which Application Delivery Controllers (ADCs) originated.

Application Delivery Controllers

Simply put, ADCs are what all good load balancers grew up to be. While most ADC conversations rarely mention load balancing, without the capabilities of the network-based hardware load balancer, they would be unable to affect application delivery at all. Today, we talk about security, availability, and performance, but the underlying load balancing technology is critical to the execution of all.

When discussing ADC security, the virtualization created by the base load balancer technology is absolutely critical. Whether we discuss SSL/TLS encryption offload, centralized authentication, or even application-fluent firewalls, the power of these solutions lies in the fact that a hardware load balancer is the aggregate point of virtualization across all applications. Centralized authentication is a classic example. Traditional authentication and authorization mechanisms have always been built directly into the application itself. Similar to the application-based load balancing, each implementation was dependent on, and unique to, each application's implementation resulting in numerous and different methods. Instead, by applying authentication at the virtualized entry point to all applications, a single, uniform method of authentication can be applied. Not only does this drastically simplify the design and management of the authentication system, it also improves the performance of the application servers themselves by eliminating the need to perform that function. Furthermore, it also eliminates the need, especially in home-grown applications, to spend the time and money to develop authentication processes in each separate application.

Availability is the easiest ADC attribute to tie back to the original load balancer as it relates to all of the basic load balancer attributes: scalability, high availability, and predictability. However, ADCs take this even further than the load balancer did. Availability for ADCs also represents advanced concepts like application dependency and dynamic provisioning. ADCs are capable of understanding that in today's world, applications rarely operate in a self-contained manner; they often rely on other applications in order to fulfill their design. This knowledge increases the ADC's capability to provide application availability by taking these other processes into account as well. The most intelligent ADCs on the market also provide programmatic interfaces that allow them to dynamically change the way they provide services based on external input. These interfaces enable such things as dynamic provisioning and the addition and/or subtraction of available servers based on utilization and need.

Performance enhancement was another obvious extension to the load balancing concept. Load balancers inherently improved performance of applications by ensuring that connections where not only distributed to services that were available (and responding in an acceptable timeframe), but also to the services with the least amount of connections and/or processor utilization. This made sure that each new connection was being serviced by the system that was best able to handle it. Later, as SSL/TLS offload (using dedicated hardware) became a common staple of load balancing offerings, it reduced the amount of computational overhead of encrypted traffic as well as the load on back-end servers—improving their performance as well.

Today's ADCs, however, go even further. These devices often include caching, compression, and even rate-shaping technology to further increase the overall performance and delivery of applications. In addition, rather than being the static implementations of traditional stand-alone appliances providing these services, an ADC can use its innate application intelligence to only apply these services when they will yield a performance benefit—thereby optimizing their use. For instance, compression technology—despite the common belief—is not necessarily beneficial to all users of the application. Certainly, users with small bandwidth (like dial-up or mobile packet-data) can benefit tremendously from smaller packets since the bottleneck is actual throughput. Even connections that must traverse long distances can benefit as smaller packets mean less round-trips to transport data decreasing the impact of network latency. However, short-distance connections (say, within the same continent) with large bandwidth (broadband cable/DSL) actually take a performance hit in applying compression; since throughput is not necessarily the bottleneck, the additional overhead of compression and decompression adds latency that the increased throughput does not make up for from a performance perspective. In other words, if not properly managed, compression technology as a solution can be worse than the original problem. But by intelligently applying compression only when it will benefit overall performance, an ADC optimizes the use and cost of compression technology, leaving more processor cycles for functions that will get the most use out of them.

The Future of ADCs

ADCs are the natural evolution to the critical network real estate that load balancers of the past held, and while they owe a great deal to those bygone devices, they are a distinctly new breed providing not just availability, but performance and security. As their name suggests, they are concerned with all aspects of delivering an application in the best way possible.

Just as load balancers evolved to become ADCs, the ever-changing needs of the technical world will continue to mold and shape ADCs into something even more capable of adapting to the availability, performance, and security requirements of application delivery. Ideas of integrating Network Access Control (generic NAC), new ideas in application caching/compression, and even the increasing importance of applying business rules to the management and control of application delivery will continue to stretch the boundaries of the benefits that these devices can offer an organization. The increasing pressure to consolidate and minimize the number of devices in the network between the user and the application will continue to collapse traditional stand-alone technologies (like firewalls, anti-virus, and IPS) into the realm of the ADC. As new technologies and protocols are developed in an attempt to satiate the increasing desire for on-anywhere, from-anywhere access to applications and data, the ADCs of tomorrow will likely provide the intelligence to determine how these (and other) technologies will be integrated into the existing network, as well as where and when they'd be most effective.

While it remains unclear exactly how many technologies will be directly replaced by ADC delivered components, it is clear that ADCs will evolve into the primary conduit and integration point through which these integrated technologies interface with the applications being delivered and the users who use them.

F5 Networks, Inc. 401 Elliott Avenue West, Seattle, WA 98119 888-882-4447 www.f5.com

F5 Networks, Inc. Corporate Headquarters info@f5.com	F5 Networks Asia-Pacific apacinfo@f5.com	F5 Networks Ltd. Europe/Middle-East/Africa emeainfo@f5.com	F5 Networks Japan K.K. f5j-info@f5.com