Load balancing (computing)

In computing, **load balancing** improves the distribution of workloads across multiple computing resources, such as computers, a computer cluster, network links, central processing units, or disk drives.[1] Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource. Using multiple components with load balancing instead of a single component may increase reliability and availability through redundancy. Load balancing usually involves dedicated software or hardware, such as a multilayer switch or a Domain Name System server process.

Load balancing differs from channel bonding in that load balancing divides traffic between network interfaces on a network socket (OSI model layer 4) basis, while channel bonding implies a division of traffic between physical interfaces at a lower level, either per packet (OSI model Layer 3) or on a data link (OSI model Layer 2) basis with a protocol like shortest path bridging.

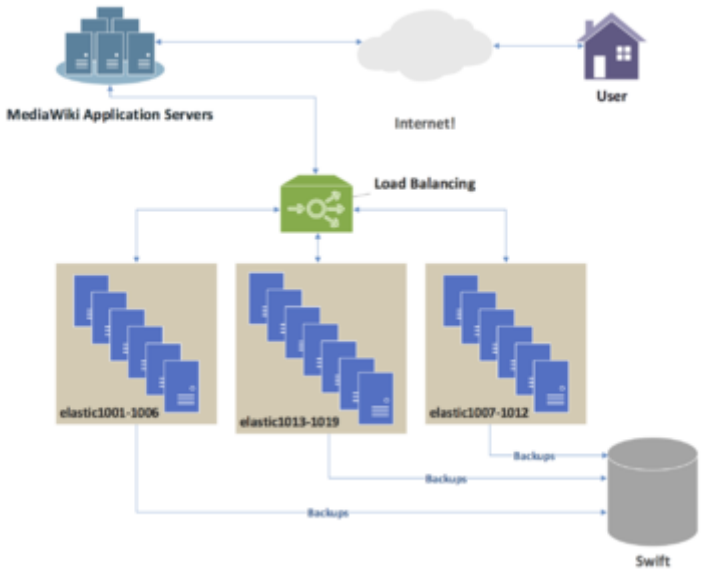

Diagram illustrating user requests to an Elasticsearch cluster being distributed by a load balancer. (Example for Wikipedia.)

Contents

Internet-based services
- Round-robin DNS
- DNS delegation
- Client-side random load balancing
- Server-side load balancers
  - Scheduling algorithms
  - Persistence
  - Load balancer features

Use in telecommunications
- Shortest Path Bridging
- Routing 1

Use in datacenter networks

Relationship to failovers

See also

References

External links

Internet-based services

One of the most commonly used applications of load balancing is to provide a single Internet service from multiple servers, sometimes known as a server farm. Commonly load-balanced systems include popular web sites, large Internet Relay Chat networks, high-bandwidth File Transfer Protocol sites, Network News Transfer Protocol (NNTP) servers, Domain Name System (DNS) servers, and databases.

Round-robin DNS

An alternate method of load balancing, which does not require a dedicated software or hardware node, is called *round robin DNS*. In this technique, multiple IP addresses are associated with a single domain name; clients are given IP in round robin fashion. IP is assigned to clients for a time quantum.

DNS delegation

Another more effective technique for load-balancing using DNS is to delegate `www.example.org` as a sub-domain whose zone is served by each of the same servers that are serving the web site. This technique works particularly well where individual servers are spread geographically on the Internet. For example:

```
one.example.org A 192.0.2.1
two.example.org A 203.0.113.2
www.example.org NS one.example.org
www.example.org NS two.example.org
```

However, the zone file for `www.example.org` on each server is different such that each server resolves its own IP Address as the A-record.[2] On server *one* the zone file for `www.example.org` reports:

```
@ in a 192.0.2.1
```

On server *two* the same zone file contains:

```
@ in a 203.0.113.2
```

This way, when a server is down, its DNS will not respond and the web service does not receive any traffic. If the line to one server is congested, the unreliability of DNS ensures less HTTP traffic reaches that server. Furthermore, the quickest DNS response to the resolver is nearly always the one from the network's closest server, ensuring geo-sensitive load-balancing. A short TTL on the A-record helps to ensure traffic is quickly diverted when a server goes down. Consideration must be given the possibility that this technique may cause individual clients to switch between individual servers in mid-session.

Client-side random load balancing

Another approach to load balancing is to deliver a list of server IPs to the client, and then to have client randomly select the IP from the list on each connection.[3][4] This essentially relies on all clients generating similar loads, and the Law of Large Numbers[4] to achieve a reasonably flat load distribution across servers. It has been claimed that client-side random load balancing tends to provide better load distribution than round-robin DNS; this has been attributed to caching issues with round-robin DNS, that in case of large DNS caching servers, tend to skew the distribution for round-robin DNS, while client-side random selection remains unaffected regardless of DNS caching.[4]

With this approach, the method of delivery of list of IPs to the client can vary, and may be implemented as a DNS list (delivered to all the clients without any round-robin), or via hardcoding it to the list. If a "smart client" is used, detecting that randomly selected server is down and connecting randomly again, it also provides fault tolerance.

Server-side load balancers

For Internet services, a server-side load balancer is usually 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.

Some load balancers provide a mechanism for doing something special in the event that all backend servers are unavailable. This might include forwarding to a backup load balancer, or displaying a message regarding the outage.

It is also important that the load balancer itself does not become a single point of failure. Usually load balancers are implemented in high-availability pairs which may also replicate session persistence data if required by the specific application.[5]

Scheduling algorithms

Numerous scheduling algorithms, also called load-balancing methods, are used by load balancers to determine which back-end server to send a request to. Simple algorithms include random choice, round robin, or least connections.[6] More sophisticated load balancers may take additional factors into account, such as a server's reported load, least response times, up/down status (determined by a monitoring poll of some kind), number of active connections, geographic location, capabilities, or how much traffic it has recently been assigned.

Persistence

An important issue when operating a load-balanced service is how to handle information that must be kept across the multiple requests in a user's session. If this information is stored locally on one backend server, then subsequent requests going to different backend servers would not be able to find it. This might be cached information that can be recomputed, in which case load-balancing a request to a different backend server just introduces a performance issue.[6]

Ideally the cluster of servers behind the load balancer should be session-aware, so that if a client connects to any backend server at any time the user experience is unaffected. This is usually achieved with a shared database or an in-memory session database, for example Memcached.

One basic solution to the session data issue is to send all requests in a user session consistently to the same backend server. This is known as "persistence" or "stickiness". A significant downside to this technique is its lack of automatic failover: if a backend server goes down, its per-session information becomes inaccessible, and any sessions depending on it are lost. The same problem is usually relevant to central database servers; even if web servers are "stateless" and not "sticky", the central database is (see below).

Assignment to a particular server might be based on a username, client IP address, or be random. Because of changes of the client's perceived address resulting from DHCP, network address translation, and web proxies this method may be unreliable. Random assignments must be remembered by the load balancer, which creates a burden on storage. If the load balancer is replaced or fails, this information may be lost, and assignments may need to be deleted after a timeout period or during periods of high load to avoid exceeding the space available for the assignment table. The random assignment method also requires that clients maintain some state, which can be a problem, for example when a web browser has disabled storage of cookies. Sophisticated load balancers use multiple persistence techniques to avoid some of the shortcomings of any one method.

Another solution is to keep the per-session data in a database. Generally this is bad for performance because it increases the load on the database: the database is best used to store information less transient than per-session data. To prevent a database from becoming a single point of failure, and to improve scalability, the database is often replicated across multiple machines, and load balancing is used to spread the query load across those replicas. Microsoft's ASP.net State Server technology is an example of a session database. All servers in a web farm store their session data on State Server and any server in the farm can retrieve the data.

In the very common case where the client is a web browser, a simple but efficient approach is to store the per-session data in the browser itself. One way to achieve this is to use a browser cookie, suitably time-stamped and encrypted. Another is URL rewriting. Storing session data on the client is generally the preferred solution: then the load balancer is free to pick any backend server to handle a request. However, this method of state-data handling is poorly suited to some complex business logic scenarios, where session state payload is big and recomputing it with every request on a server is not feasible. URL rewriting has major security issues, because the end-user can easily alter the submitted URL and thus change session streams.

Yet another solution to storing persistent data is to associate a name with each block of data, and use a distributed hash table to pseudo-randomly assign that name to one of the available servers, and then store that block of data in the assigned server.

Load balancer features

Hardware and software load balancers may have a variety of special features. The fundamental feature of a load balancer is to be able to distribute incoming requests over a number of backend servers in the cluster according to a scheduling algorithm. Most of the following features are vendor specific:

Asymmetric load
A ratio can be manually assigned to cause some backend servers to get a greater share of the workload than others. This is sometimes used as a crude way to account for some servers having more capacity than others and may not always work as desired.

Priority activation
When the number of available servers drops below a certain number, or load gets too high, standby servers can be brought online.

TLS Offload and Acceleration
TLS (or its predecessor SSL) acceleration is a technique of offloading cryptographic protocol calculations onto a specialized hardware. Depending on the workload, processing the encryption and authentication requirements of an TLS request can become a major part of the demand on the Web Server's CPU; as the demand increases, users will see slower response times, as the TLS overhead is distributed among Web servers. To remove this demand on Web servers, a balancer can terminate TLS connections, passing HTTPS requests as HTTP requests to the Web servers. If the balancer itself is not overloaded, this does not noticeably degrade the performance perceived by end users. The downside of this approach is that all of the TLS processing is concentrated on a single device (the balancer) which can become a new bottleneck. Some load balancer appliances include specialized hardware to process TLS. Instead of upgrading the load balancer, which is quite expensive dedicated hardware, it may be cheaper to forgo TLS offload and add a few Web servers. Also, some server vendors such as Oracle/Sun now incorporate cryptographic acceleration hardware into their CPUs such as the T2000. F5 Networks incorporates a dedicated TLS acceleration hardware card in their local traffic manager (LTM) which is used for encrypting and decrypting TLS traffic. One clear benefit to TLS offloading in the balancer is that it enables it to do balancing or content switching based on data in the HTTPS request.

Distributed Denial of Service (DDoS) attack protection
Load balancers can provide features such as SYN cookies and delayed-binding (the back-end servers don't see the client until it finishes its TCP handshake) to mitigate SYN flood attacks and generally offload work from the servers to a more efficient platform.

HTTP compression
HTTP compression reduces the amount of data to be transferred for HTTP objects by utilising gzip compression available in all modern web browsers. The larger the response and the further away the client is, the more this feature can improve response times. The trade-off is that this feature puts additional CPU demand on the load balancer and could be done by web servers instead.

TCP offload
Different vendors use different terms for this, but the idea is that normally each HTTP request from each client is a different TCP connection. This feature utilises HTTP/1.1 to consolidate multiple HTTP requests from multiple clients into a single TCP socket to the back-end servers.

TCP buffering
The load balancer can buffer responses from the server and spoon-feed the data out to slow clients, allowing the web server to free a thread for other tasks faster than it would if it had to send the entire request to the client directly.

Direct Server Return
An option for asymmetrical load distribution, where request and reply have different network paths.

Health checking
The balancer polls servers for application layer health and removes failed servers from the pool.

HTTP caching
The balancer stores static content so that some requests can be handled without contacting the servers.

Content filtering
Some balancers can arbitrarily modify traffic on the way through.

HTTP security
Some balancers can hide HTTP error pages, remove server identification headers from HTTP responses, and encrypt cookies so that end users cannot manipulate them.

Priority queuing
Also known as rate shaping, the ability to give different priority to different traffic.

Content-aware switching
Most load balancers can send requests to different servers based on the URL being requested, assuming the request is not encrypted (HTTP) or if it is encrypted (via HTTPS) that the HTTPS request is terminated (decrypted) at the load balancer.

Client authentication
Authenticate users against a variety of authentication sources before allowing them access to a website.

Programmatic traffic manipulation
: At least one balancer allows the use of a scripting language to allow custom balancing methods, arbitrary traffic manipulations, and more.

Firewall
: Firewalls can prevent direct connections to backend servers, for network security reasons.

Intrusion prevention system
: Intrusion prevention systems offer application layer security in addition to network/transport layer offered by firewall security.

Use in telecommunications

Load balancing can be useful in applications with redundant communications links. For example, a company may have multiple Internet connections ensuring network access if one of the connections fails. A failover arrangement would mean that one link is designated for normal use, while the second link is used only if the primary link fails.

Using load balancing, both links can be in use all the time. A device or program monitors the availability of all links and selects the path for sending packets. The use of multiple links simultaneously increases the available bandwidth.

Shortest Path Bridging

The IEEE approved the IEEE 802.1aq standard May 2012,[7] also known and documented in most books as Shortest Path Bridging (SPB). SPB allows all links to be active through multiple equal cost paths, provides faster convergence times to reduce down time, and simplifies the use of load balancing in mesh network topologies (partially connected and/or fully connected) by allowing traffic to load share across all paths of a network.[8][9] SPB is designed to virtually eliminate human error during configuration and preserves the plug-and-play nature that established Ethernet as the de facto protocol at Layer 2.[10]

Routing 1

Many telecommunications companies have multiple routes through their networks or to external networks. They use sophisticated load balancing to shift traffic from one path to another to avoid network congestion on any particular link, and sometimes to minimize the cost of transit across external networks or improve network reliability.

Another way of using load balancing is in network monitoring activities. Load balancers can be used to split huge data flows into several sub-flows and use several network analyzers, each reading a part of the original data. This is very useful for monitoring fast networks like 10GbE or STM64, where complex processing of the data may not be possible at wire speed.[11]

Use in datacenter networks

Load balancing is widely used in datacenter networks to distribute traffic across many existing paths between any two servers.[12] It allows more efficient use of network bandwidth and reduces provisioning costs. In general, load balancing in datacenter networks can be classified as either static or dynamic. Static load balancing distributes traffic by computing a hash of the source and destination addresses and port numbers of traffic flows and using it to determine how flows are assigned to one of the existing paths. Dynamic load balancing assigns traffic flows to paths by monitoring bandwidth utilization of different paths. Dynamic assignment can also be proactive or reactive. In the former case, the assignment is fixed once made, while in the latter the network logic keeps monitoring available paths and shifts flows across them as network utilization changes (with arrival of new flows or completion of existing ones). A comprehensive overview of load balancing in datacenter networks has been made available.[12]

Relationship to failovers

Load balancing is often used to implement failover—the continuation of a service after the failure of one or more of its components. The components are monitored continually (e.g., web servers may be monitored by fetching known pages), and when one becomes non-responsive, the load balancer is informed and no longer sends traffic to it. When a component comes back online, the load balancer begins to route traffic to it again. For this to work, there must be at least one component in excess of the service's capacity

(N+1 redundancy). This can be much less expensive and more flexible than failover approaches where each single live component is paired with a single backup component that takes over in the event of a failure (dual modular redundancy). Some types of RAID systems can also utilize hot spare for a similar effect.[13]

See also

- Affinity mask
- Application Delivery Controller
- Autoscaling
- Cloud computing
- Common Address Redundancy Protocol
- Edge computing
- Fog computing
- Network Load Balancing
- Network Load Balancing Services
- Partition (database)
- Processor affinity
- SRV record

References

1. Performance Tradeoffs in Static and Dynamic Load Balancing Strategies (https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19860014876.pdf), NASA March 1986
2. IPv4 Address Record (A) (http://www.zytrax.com/books/dns/ch8/a.html)
3. Pattern: Client Side Load Balancing (https://gameserverarchitecture.com/2015/10/pattern-client-side-load-balancing/)
4. MMOG Server-Side Architecture. Front-End Servers and Client-Side Random Load Balancing (http://ithare.com/chapter-vib-server-side-architecture-front-end-servers-and-client-side-random-load-balancing/)
5. "High Availability" (http://www.linuxvirtualserver.org/HighAvailability.html). linuxvirtualserver.org. Retrieved 2013-11-20.
6. "Load Balancing 101: Nuts and Bolts" (https://web.archive.org/web/20171205223948/https://f5.com/resources/white-papers/load-balancing-101-nuts-and-bolts). F5 Networks. 2017-12-05. Retrieved 2018-03-23.
7. Shuang Yu (8 May 2012). "IEEE APPROVES NEW IEEE 802.1aq™ SHORTEST PATH BRIDGING STANDARD" (http://standards.ieee.org/news/2012/802.1aq.html). IEEE. Retrieved 2 June 2012.
8. Peter Ashwood-Smith (24 Feb 2011). "Shortest Path Bridging IEEE 802.1aq Overview" (http://meetings.apnic.net/__data/assets/pdf_file/0012/32007/APRICOT_SPB_Overview.pdf) (PDF). Huawei. Retrieved 11 May 2012.
9. Jim Duffy (11 May 2012). "Largest Illinois healthcare system uproots Cisco to build $40M private cloud" (http://www.pcadvisor.co.uk/news/internet/3357242/largest-illinois-healthcare-system-uproots-cisco-build-40m-private-cloud/). PC Advisor. Retrieved 11 May 2012. "Shortest Path Bridging will replace Spanning Tree in the Ethernet fabric."
10. "IEEE Approves New IEEE 802.1aq Shortest Path Bridging Standard" (http://www.techpowerup.com/165594/IEEE-Approves-New-IEEE-802.1aq-Shortest-Path-Bridging-Standard.html). Tech Power Up. 7 May 2012. Retrieved 11 May 2012.
11. Mohammad Noormohammadpour, Cauligi S. RaghavendraMinimizing Flow Completion Times using Adaptive Routing over Inter-Datacenter Wide Area Networks (https://www.researchgate.net/publication/323723167_Minimizing_Flow_Completion_Times_using_Adaptive_Routing_over_Inter-Datacenter_Wide_Area_Networks) *IEEE INFOCOM 2018 Poster Sessions, DOI:10.13140/RG.2.2.36009.90720* 6 January 2019
12. M. Noormohammadpour, C. S. Raghavendra, "Datacenter Traffic Control: Understanding Techniques and Trade-offs," (https://www.researchgate.net/publication/321744877_Datacenter_Traffic_Control_Understanding_Techniques_and_Trade-offs) IEEE Communications Surveys & Tutorials, vol. PP, no. 99, pp. 1-1.
13. Failover and load balancing (https://www.ibm.com/support/knowledgecenter/en/SSVJJU_6.4.0/com.ibm.IBMDS.doc_6.4/ds_ag_srv_adm_dd_failover_load_balancing.html) *IBM* 6 January 2019

External links

- Server routing for load balancing with full auto failure recovery (http://www.udaparts.com/document/articles/snpisec.htm)