Load balancing (computing)

In computing, **load balancing** is the process of distributing a set of tasks over a set of resources (computing units), with the aim of making their overall processing more efficient. Load balancing can optimize the response time and avoid unevenly overloading some compute nodes while other compute nodes are left idle.

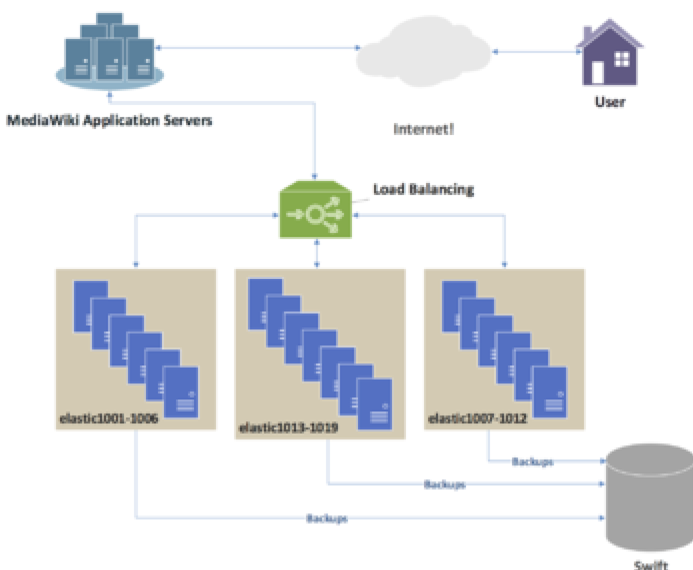


Diagram illustrating user requests to an Elasticsearch cluster being distributed by a load balancer. (Example for Wikipedia.)

Load balancing is the subject of research in the field of parallel computers. Two main approaches exist: static algorithms, which do not take into account the state of the different machines, and dynamic algorithms, which are usually more general and more efficient but require exchanges of information between the different computing units, at the risk of a loss of efficiency.

Problem overview

A load-balancing algorithm always tries to answer a specific problem. Among other things, the nature of the tasks, the algorithmic complexity, the hardware architecture on which the algorithms will run as well as required error tolerance, must be taken into account. Therefore compromise must be found to best meet application-specific requirements.

Nature of tasks

The efficiency of load balancing algorithms critically depends on the nature of the tasks. Therefore, the more information about the tasks is available at the time of decision making, the greater the potential for optimization.

Size of tasks

Perfect knowledge of the execution time of each of the tasks allows to reach an optimal load distribution (see algorithm of prefix sum).[1] Unfortunately, this is in fact an idealized case. Knowing the exact execution time of each task is an extremely rare situation.

For this reason, there are several techniques to get an idea of the different execution times. First of all, in the fortunate scenario of having tasks of relatively homogeneous size, it is possible to consider that each of them will require approximately the average execution time. If, on the other hand, the execution time is very irregular, more sophisticated techniques must be used. One technique is to add some metadata to each task. Depending on the previous execution time for similar metadata, it is possible to make inferences for a future task based on statistics.[2]

Dependencies

In some cases, tasks depend on each other. These interdependencies can be illustrated by a directed acyclic graph. Intuitively, some tasks cannot begin until others are completed.

Assuming that the required time for each of the tasks is known in advance, an optimal execution order must lead to the minimization of the total execution time. Although this is an NP-hard problem and therefore can be difficult to be solved exactly. There are algorithms, like job scheduler, that calculate optimal task distributions using metaheuristic methods.

Segregation of tasks

Another feature of the tasks critical for the design of a load balancing algorithm is their ability to be broken down into subtasks during execution. The "Tree-Shaped Computation" algorithm presented later takes great advantage of this specificity.

Static and dynamic algorithms

Static

A load balancing algorithm is "static" when it does not take into account the state of the system for the distribution of tasks. Thereby, the system state includes measures such as the load level (and sometimes even overload) of certain processors. Instead, assumptions about the overall system are made beforehand, such as the arrival times and resource requirements of incoming tasks. In addition, the number of processors, their respective power and communication speeds are known. Therefore, static load balancing aims to associate a known set of tasks with the available processors in order to minimize a certain performance function. The trick lies in the concept of this performance function.

Static load balancing techniques are commonly centralized around a router, or Master, which distributes the loads and optimizes the performance function. This minimization can take into account information related to the tasks to be distributed, and derive an expected execution time.

The advantage of static algorithms is that they are easy to set up and extremely efficient in the case of fairly regular tasks (such as processing HTTP requests from a website). However, there is still some statistical variance in the assignment of tasks which can lead to the overloading of some computing units.

Dynamic

Unlike static load distribution algorithms, dynamic algorithms take into account the current load of each of the computing units (also called nodes) in the system. In this approach, tasks can be moved dynamically from an overloaded node to an underloaded node in order to receive faster processing. While these algorithms are much more complicated to design, they can produce excellent results, in particular, when the execution time varies greatly from one task to another.

Dynamic load balancing architecture can be more modular since it is not mandatory to have a specific node dedicated to the distribution of work. When tasks are uniquely assigned to a processor according to their state at a given moment, it is a unique assignment. If, on the other hand, the tasks can be permanently redistributed according to the state of the system and its evolution, this is called dynamic assignment.[3] Obviously, a load balancing algorithm that requires too much communication in order to reach its decisions runs the risk of slowing down the resolution of the overall problem.

Hardware architecture

Heterogenous machines

Parallel computing infrastructures are often composed of units of different computing power, which should be taken into account for the load distribution.

For example, lower-powered units may receive requests that require a smaller amount of computation, or, in the case of homogeneous or unknown request sizes, receive fewer requests than larger units.

Shared and distributed memory

Parallel computers are often divided into two broad categories: those where all processors share a single common memory on which they read and write in parallel (PRAM model), and those where each computing unit has its own memory (distributed memory model), and where information is exchanged by messages.

For shared-memory computers, managing write conflicts greatly slows down the speed of individual execution of each computing unit. However, they can work perfectly well in parallel. Conversely, in the case of message exchange, each of the processors can work at full speed. On

the other hand, when it comes to collective message exchange, all processors are forced to wait for the slowest processors to start the communication phase.

In reality, few systems fall into exactly one of the categories. In general, the processors each have an internal memory to store the data needed for the next calculations and are organized in successive clusters. Often, these processing elements are then coordinated through distributed memory and message passing. Therefore, the load balancing algorithm should be uniquely adapted to a parallel architecture. Otherwise, there is a risk that the efficiency of parallel problem solving will be greatly reduced.

Hierarchy

Adapting to the hardware structures seen above, there are two main categories of load balancing algorithms. On the one hand, the one where tasks are assigned by “master” and executed by “workers” who keep the master informed of the progress of their work, and the master can then take charge of assigning or reassigning the workload in case of the dynamic algorithm. The literature refers to this as "Master-Worker" architecture. On the other hand, the control can be distributed between the different nodes. The load balancing algorithm is then executed on each of them and the responsibility for assigning tasks (as well as re-assigning and splitting as appropriate) is shared. The last category assumes a dynamic load balancing algorithm.

Since the design of each load balancing algorithm is unique, the previous distinction must be qualified. Thus, it is also possible to have an intermediate strategy, with, for example, "master" nodes for each sub-cluster, which are themselves subject to a global "master". There are also multi-level organizations, with an alternation between master-slave and distributed control strategies. The latter strategies quickly become complex and are rarely encountered. Designers prefer algorithms that are easier to control.

Adaptation to larger architectures (scalability)

In the context of algorithms that run over the very long term (servers, cloud...), the computer architecture evolves over time. However, it is preferable not to have to design a new algorithm each time.

An extremely important parameter of a load balancing algorithm is therefore its ability to adapt to scalable hardware architecture. This is called the scalability of the algorithm. An algorithm is

called scalable for an input parameter when its performance remains relatively independent of the size of that parameter.

When the algorithm is capable of adapting to a varying number of computing units, but the number of computing units must be fixed before execution, it is called moldable. If, on the other hand, the algorithm is capable of dealing with a fluctuating amount of processors during its execution, the algorithm is said to be malleable. Most load balancing algorithms are at least moldable.[4]

Fault tolerance

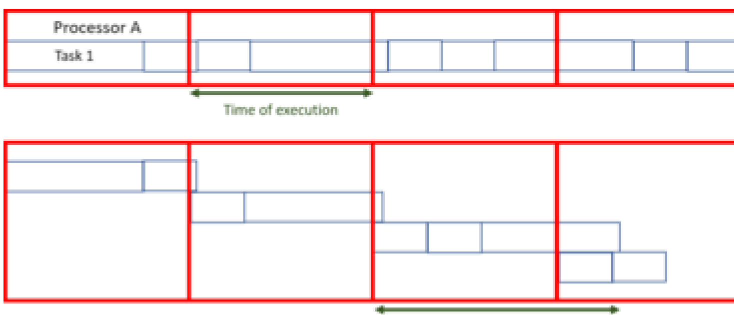
Especially in large-scale computing clusters, it is not tolerable to execute a parallel algorithm that cannot withstand the failure of one single component. Therefore, fault tolerant algorithms are being developed which can detect outages of processors and recover the computation.[5]

Approaches

Static distribution with full knowledge of the tasks: prefix sum

If the tasks are independent of each other, and if their respective execution time and the tasks can be subdivided, there is a simple and optimal algorithm.

By dividing the tasks in such a way as to give the same amount of computation to each processor, all that remains to be done is to group the results together. Using a prefix sum algorithm, this division can be calculated in logarithmic time with respect to the number of processors.



Load balancing algorithm depending on divisibility of tasks

If, however, the tasks cannot be subdivided (i.e., they are atomic), although optimizing task assignment is a difficult problem, it is still possible to approximate a relatively fair distribution of tasks, provided that the size of each of them is much smaller than the total computation performed by each of the nodes.[1]

Most of the time, the execution time of a task is unknown and only rough approximations are available. This algorithm, although particularly efficient, is not viable for these scenarios.

Static load distribution without prior knowledge

Even if the execution time is not known in advance at all, static load distribution is always possible.

Round-robin scheduling

In a round-robin algorithm, the first request is sent to the first server, then the next to the second, and so on down to the last. Then it is started again, assigning the next request to the first server, and so on.

This algorithm can be weighted such that the most powerful units receive the largest number of requests and receive them first.

Randomized static

Randomized static load balancing is simply a matter of randomly assigning tasks to the different servers. This method works quite well. If, on the other hand, the number of tasks is known in advance, it is even more efficient to calculate a random permutation in advance. This avoids communication costs for each assignment. There is no longer a need for a distribution master because every processor knows what task is assigned to it. Even if the number of tasks is unknown, it is still possible to avoid communication with a pseudo-random assignment generation known to all processors.

The performance of this strategy (measured in total execution time for a given fixed set of tasks) decreases with the maximum size of the tasks.

Others

Of course, there are other methods of assignment as well:

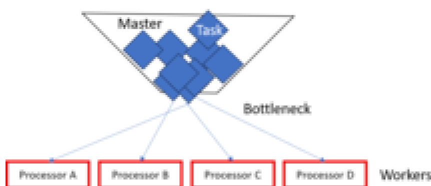
- Less work: Assign more tasks to the servers by performing less (the method can also be weighted).
- Hash: allocates queries according to a hash table.
- Power of Two Choices: pick two servers at random and choose the better of the two options.[6][7]

Master-Worker Scheme

Master-Worker schemes are among the simplest dynamic load balancing algorithms. A master distributes the workload to all workers (also sometimes referred to as "slaves"). Initially, all workers are idle and report this to the master. The master answers worker requests and distributes the tasks to them. When he has no more tasks to give, he informs the workers so that they stop asking for tasks.

The advantage of this system is that it distributes the burden very fairly. In fact, if one does not take into account the time needed for the assignment, the execution time would be comparable to the prefix sum seen above.

The problem with this algorithm is that it has difficulty adapting to a large number of processors because of the high amount of necessary communications. This lack of scalability makes it quickly inoperable in very large servers or very large parallel computers. The master acts as a bottleneck.



Master-Worker and bottleneck

However, the quality of the algorithm can be greatly improved by replacing the master with a task list that can be used by different processors. Although this algorithm is a little more difficult to implement, it promises much better scalability, although still insufficient for very large computing centers.

Non-hierarchical architecture, without knowledge of the system: work stealing

Another technique to overcome scalability problems when the time needed for task completion is unknown is work stealing.

The approach consists of assigning to each processor a certain number of tasks in a random or predefined manner, then allowing inactive processors to "steal" work from active or overloaded processors. Several implementations of this concept exist, defined by a task division model and by the rules determining the exchange between processors. While this technique can be particularly effective, it is difficult to implement because it is necessary to ensure that communication does not become the primary occupation of the processors instead of solving the problem.

In the case of atomic tasks, two main strategies can be distinguished, those where the processors with low load offer their computing capacity to those with the highest load, and those were the most loaded units wish to lighten the workload assigned to them. It has been shown[8] that when the network is heavily loaded, it is more efficient for the least loaded units to offer their availability and when the network is lightly loaded, it is the overloaded processors that require support from the most inactive ones. This rule of thumb limits the number of exchanged messages.

In the case where one starts from a single large task that cannot be divided beyond an atomic level, there is a very efficient algorithm "Tree-Shaped computation",[9] where the parent task is distributed in a work tree.

Principle

Initially, many processors have an empty task, except one that works sequentially on it. Idle processors issue requests randomly to other processors (not necessarily active). If the latter is able to subdivide the task it is working on, it does so by sending part of its work to the node making the request. Otherwise, it returns an empty task. This induces a tree structure. It is then necessary to send a termination signal to the parent processor when the subtask is completed

so that it, in turn, sends the message to its parent until it reaches the root of the tree. When the first processor, i.e. the root, has finished, a global termination message can be broadcast. In the end, it is necessary to assemble the results by going back up the tree.

Efficiency

The efficiency of such an algorithm is close to the prefix sum when the job cutting and communication time is not too high compared to the work to be done. To avoid too high communication costs, it is possible to imagine a list of jobs on shared memory. Therefore, a request is simply reading from a certain position on this shared memory at the request of the master processor.

Use cases

In addition to efficient problem solving through parallel computations, load balancing algorithms are widely used in HTTP request management where a site with a large audience must be able to handle a large number of requests per second.

Internet-based services

One of the most commonly used applications of load balancing is to provide 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 (FTP) sites, Network News Transfer Protocol (NNTP) servers, Domain Name System (DNS) servers, and databases.

Round-robin DNS

Round-robin DNS is an alternate method of load balancing that does not require a dedicated software or hardware node. In this technique, multiple IP addresses are associated with a single domain name; clients are given IP in a round-robin fashion. IP is assigned to clients with a short expiration so the client is more likely to use a different IP the next time they access the Internet service being requested.

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 website. 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.[10] 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 to 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 the client randomly select the IP from the list on each connection.[11][12] This essentially relies on all clients generating similar loads, and the Law of Large Numbers[12] 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 the 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.[12]

With this approach, the method of delivery of a 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 a 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.[13] Certain applications are programmed with immunity to this problem, by offsetting the load balancing point over differential sharing platforms beyond the defined network. The sequential algorithms paired to these functions are defined by flexible parameters unique to the specific database.[14]

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.[15] 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), a 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.[15]

Ideally, the cluster of servers behind the load balancer should not 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 like 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 random. Because of changes in 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 the 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. This is generally 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 the 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 specialized hardware. Depending on the workload, processing the encryption and authentication requirements of a 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 webserver 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 priorities 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 the network/transport layer offered by firewall security.

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

TRILL (TRansparent Interconnection of Lots of Links) facilitates an Ethernet to have an arbitrary topology, and enables per flow pair-wise load splitting by way of Dijkstra's algorithm, without configuration and user intervention. The catalyst for TRILL was an event at Beth Israel Deaconess Medical Center which began on 13 November 2002.[16][17] The concept of Rbridges[18] [sic] was first proposed to the Institute of Electrical and Electronics Engineers in the

year 2004,[19] whom in 2005[20] rejected what came to be known as TRILL, and in the years 2006 through 2012[21] devised an incompatible variation known as Shortest Path Bridging.

The IEEE approved the IEEE 802.1aq standard in May 2012,[22] also known as Shortest Path Bridging (SPB). SPB allows all links to be active through multiple equal-cost paths, provides faster convergence times to reduce downtime, 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.[23][24] 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.[25]

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.[26]

Data center networks

Load balancing is widely used in data center networks to distribute traffic across many existing paths between any two servers.[27] 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 use on different paths. Dynamic assignments 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.[27]

Failovers

Load balancing is often used to implement failover—the continuation of 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 unresponsive, the load balancer is informed and no longer sends traffic to it. When a component comes back online, the load balancer starts rerouting traffic to it. 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 every single live component is paired with a single backup component that takes over in the event of a failure (dual modular redundancy). Some RAID systems can also utilize hot spare for a similar effect.[28]

See also

- Affinity mask
- Application Delivery Controller
- Autoscaling
- Cloud computing
- Cloud load balancing
- Common Address Redundancy Protocol
- Edge computing
- InterPlanetary File System
- Network Load Balancing
- SRV record

References

1. *Sanders, Peter; Mehlhorn, Kurt; Dietzfelbinger, Martin; Dementiev, Roman (11 September 2019). Sequential and parallel algorithms and data structures : the basic toolbox. ISBN 978-3-030-25208-3.*

2. Liu, Qi; Cai, Weidong; Jin, Dandan; Shen, Jian; Fu, Zhangjie; Liu, Xiaodong; Linge, Nigel (30 August 2016). "Estimation Accuracy on Execution Time of Run-Time Tasks in a Heterogeneous Distributed Environment" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5038664). Sensors. **16** (9): 1386. Bibcode:2016Senso..16.1386L (https://ui.adsabs.harvard.edu/abs/2016Senso..16.1386L). doi:10.3390/s16091386 (https://doi.org/10.3390%2Fs16091386). PMC 5038664 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5038664). PMID 27589753 (https://pubmed.ncbi.nlm.nih.gov/27589753). S2CID 391429 (https://api.semanticscholar.org/CorpusID:391429).
3. Alakeel, Ali (November 2009). "A Guide to Dynamic Load Balancing in Distributed Computer Systems" (https://www.researchgate.net/publication/268200851). International Journal of Computer Science and Network Security (IJCSNS). **10**.
4. Asghar, Sajjad; Aubanel, Eric; Bremner, David (October 2013). "A Dynamic Moldable Job Scheduling Based Parallel SAT Solver". 2013 42nd International Conference on Parallel Processing: 110–119. doi:10.1109/ICPP.2013.20 (https://doi.org/10.1109%2FICPP.2013.20). ISBN 978-0-7695-5117-3. S2CID 15124201 (https://api.semanticscholar.org/CorpusID:15124201).
5. Punetha Sarmila, G.; Gnanambigai, N.; Dinadayalan, P. (2015). "Survey on fault tolerant — Load balancing algorithmsin cloud computing". 2nd International Conference on Electronics and Communication Systems (ICECS): 1715–1720. doi:10.1109/ECS.2015.7124879 (https://doi.org/10.1109%2FECS.2015.7124879). ISBN 978-1-4799-7225-8. S2CID 30175022 (https://api.semanticscholar.org/CorpusID:30175022).
6. "NGINX and the "Power of Two Choices" Load-Balancing Algorithm" (https://web.archive.org/web/20191212194243/https://www.nginx.com/blog/nginx-power-of-two-choices-load-balancing-algorithm/). nginx.com. 2018-11-12. Archived from the original (https://www.nginx.com/blog/nginx-power-of-two-choices-load-balancing-algorithm/) on 2019-12-12.
7. "Test Driving "Power of Two Random Choices" Load Balancing" (https://web.archive.org/web/20190215173140/https://www.haproxy.com/blog/power-of-two-load-balancing/). haproxy.com. 2019-02-15. Archived from the original (https://www.haproxy.com/blog/power-of-two-load-balancing/) on 2019-02-15.
8. Eager, Derek L; Lazowska, Edward D; Zahorjan, John (1 March 1986). "A comparison of receiver-initiated and sender-initiated adaptive load sharing". Performance Evaluation. **6** (1): 53–68. doi:10.1016/0166-5316(86)90008-8 (https://doi.org/10.1016%2F0166-5316%2886%2990008-8). ISSN 0166-5316 (https://www.worldcat.org/issn/0166-5316).

9. *Sanders, Peter (1998). "Tree Shaped Computations as a Model for Parallel Applications". Workshop on Application Based Load Balancing (Alv '98), München, 25. - 26. März 1998 - Veranst. Vom Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung Paralleler Rechnerarchitekturen". Ed.: A. Bode: 123. doi:10.5445/ir/1000074497 (https://doi.org/10.5445%2Fir%2F1000074497) .*
10. *IPv4 Address Record (A) (http://www.zytrax.com/books/dns/ch8/a.html)*
11. *Pattern: Client Side Load Balancing (https://gameserverarchitecture.com/2015/10/pattern-client-side-load-balancing/)*
12. *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/)*
13. *"High Availability" (http://www.linuxvirtualserver.org/HighAvailability.html) . linuxvirtualserver.org. Retrieved 2013-11-20.*
14. *Ranjan, R (2010). "Peer-to-peer cloud provisioning: Service discovery and load-balancing". Cloud Computing.*
15. *"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. Archived from the original (https://f5.com/resources/white-papers/load-balancing-101-nuts-and-bolts) on 2017-12-05. Retrieved 2018-03-23.*
16. *"All Systems Down" (https://web.archive.org/web/20200923200221if_/https://community.cisco.com/legacyfs/online/legacy/0/9/8/140890-All%20Systems%20Down%20-%20Scott%20Berinato(CIO).pdf) (PDF). cio.com. IDG Communications, Inc. Archived from the original (https://community.cisco.com/legacyfs/online/legacy/0/9/8/140890-All%20Systems%20Down%20-%20Scott%20Berinato(CIO).pdf) (PDF) on 23 September 2020. Retrieved 9 January 2022.*
17. *"All Systems Down" (https://web.archive.org/web/20220109020703/https://www.computerworld.com/article/2581420/all-systems-down.html) . cio.com. IDG Communications, Inc. Archived from the original (https://www.computerworld.com/article/2581420/all-systems-down.html) on 9 January 2022. Retrieved 9 January 2022.*

18. *"Rbridges: Transparent Routing" (https://web.archive.org/web/20220109030037/https://courses.cs.washington.edu/courses/cse590l/05sp/papers/rbridges.pdf) (PDF). courses.cs.washington.edu. Radia Perlman, Sun Microsystems Laboratories. Archived from the original (https://courses.cs.washington.edu/courses/cse590l/05sp/papers/rbridges.pdf) (PDF) on 9 January 2022. Retrieved 9 January 2022.*
19. *"Rbridges: Transparent Routing" (https://www.researchgate.net/publication/4102976). researchgate.net. Radia Perlman, Sun Microsystems; Donald Eastlake 3rd, Motorola.*
20. *"TRILL Tutorial" (http://www.postel.org/rbridge/trill-tutorial.pdf) (PDF). postel.org. Donald E. Eastlake 3rd, Huawei.*
21. *"IEEE 802.1: 802.1aq - Shortest Path Bridging" (https://ieee802.org/1/pages/802.1aq.html). ieee802.org. Institute of Electrical and Electronics Engineers.*
22. *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.*
23. *Peter Ashwood-Smith (24 Feb 2011). "Shortest Path Bridging IEEE 802.1aq Overview" (https://web.archive.org/web/20130515115628/http://meetings.apnic.net/__data/assets/pdf_file/0012/32007/APRICOT_SPB_Overview.pdf) (PDF). Huawei. Archived from the original (http://meetings.apnic.net/__data/assets/pdf_file/0012/32007/APRICOT_SPB_Overview.pdf) (PDF) on 15 May 2013. Retrieved 11 May 2012.*
24. *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."*
25. *"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.*
26. *Mohammad Noormohammadpour, Cauligi S. Raghavendra Minimizing 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*

27. *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.*
28. *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

Wikimedia Commons has media related to ***Load balancing (computing)***.

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

Retrieved from "https://en.wikipedia.org/w/index.php?title=Load_balancing_(computing)&oldid=1119414906"

Last edited 2 months ago by AnomieBOT

Wikipedia