Load Balancing for Heterogeneous Web Servers

Adam Piórkowski[1], Aleksander Kempny[2], Adrian Hajduk[1], and Jacek Strzelczyk[1]

[1] Department of Geoinfomatics and Applied Computer Science,
AGH University of Science and Technology, Cracow, Poland
{adam.piorkowski,jacek.strzelczyk}@agh.edu.pl
http://www.agh.edu.pl

[2] Adult Congenital and Valvular Heart Disease Center
University of Muenster, Muenster, Germany
aleksander.kempny@ukmuenster.de
http://www.ukmuenster.de

Abstract. A load balancing issue for heterogeneous web servers is described in this article. The review of algorithms and solutions is shown. The selected Internet service for on-line echocardiography training is presented. The independence of simultaneous requests for this server is proved. Results of experimental tests are presented[3].

Key words: load balancing, scalability, web server, minimum response time, throughput, on-line simulator

1 Introduction

Modern web servers can handle millions of queries, although the performance of a single node is limited. Performance can be continuously increased, if the services are designed so that they can be scaled. The concept of scalability is closely related to load balancing. This technique has been used since the beginning of the first distributed systems, including rich client architecture. Most of the complex web systems use load balancing to improve performance, availability and security [1–4].

2 Load Balancing in Cluster of web servers

Clustering of web servers is a method of constructing scalable Internet services. The basic idea behind the construction of such a service is to set the relay server

[3] This is the accepted version of:

Piorkowski, A., Kempny, A., Hajduk, A., Strzelczyk, J.: Load Balancing for Heterogeneous Web Servers. In: Kwiecien, A., Gaj, P., Stera, P. (eds.) CN 2010. CCIS, vol. 79, pp. 189-198. Springer, Heidelberg (2010)

The original publication is available on www.springerlink.com in a transparent connection to the cluster servers (Fig. 1). This server is called a dispatcher.

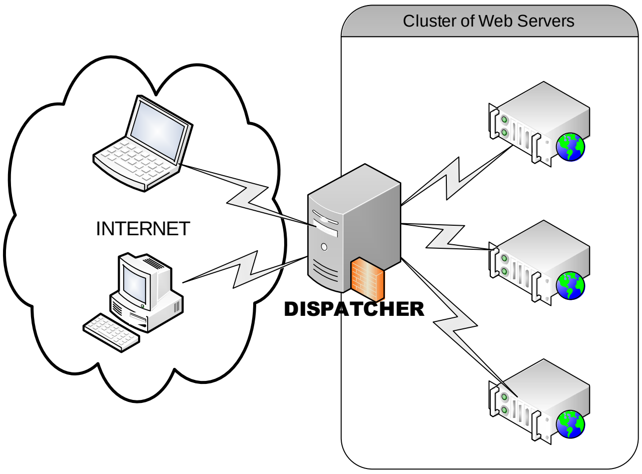

Fig. 1. Cluster of web servers schema.

There are many implementations of load balancing. Some of them (like OpenSSI for Linux or MS Windows Network Load Balancing) use additional software that have to be installed on each cluster node. This software monitors the load of these nodes, but is dedicated for selected operating systems or web server software. It makes a construction of heterogeneous web servers clusters impossible. This article focuses only on some implementations that allow to create heterogeneous web server clusters. This means that it is desirable to create services based on the nodes that use different operating systems and different environments, but process the same requests. Technique which realizes load balancing fulfilling these assumptions is proxy load balancing.

2.1 The algorithms of load balancing

Efficient load balancing requires an appropriate algorithm. There are several basic and common algorithms being discussed further in this paper:

- Round Robin [1],
- Weighted Round Robin,
- Least Loaded [1, 2],
- Least Connection [1],
- Weighted Least-Connection,
- Locality-Based Least-Connection,
- Destination Hashing [13],
- Source Hashing [12, 13],

– Fair [11],
– Never Queue,
– Shortest Queue First [14],
– Request Counting [9],
– Weighted Traffic Counting [9],
– Pending Request Counting [9].

The Round-Robin algorithm is a well-known algorithm and it is easy to implement. Request Counting algorithm distributes the requests among the various nodes to ensure that each node gets its configured share of the number of requests [9]. Weighted Traffic Counting algorithm works in a similar way, but the maximum number of requests per server is determined on the network traffic, in bytes. Pending Request Counting algorithm's idea is to keep track of how many requests each worker is assigned at the time. New requests are assigned to the server with the lowest number of active requests. The fair algorithm is based on the standard round-robin algorithm and it tracks busy back end servers and balances the load to non-busy servers [11]. There are many other algorithms, some of them require special knowledge to predict the best scheduling [2].

2.2 Solutions

There are a few implementations of load balancing proxies that enable to create a heterogeneous cluster of web servers. The most interesting from the perspective of authors of scientific portals (non-commercial) are open software solutions. In this article we discussed six of them:

– Apache Server with Mod Proxy Balancer [9] - this is a standard solution used in the most popular web server, it implements three load balancing algorithms : Request Counting, Weighted Traffic Counting and Pending Request Counting,
– Pound [10] - a simple solution with Round-Robin algorithm, distributed under the GPL ,
– NGiNX [11] - this software implements Round Robin and Fair load balancing algorithms, NGiNX is licensed under 2-clause BSD-like license,
– Inlab Balance [12] - is an open source load balancer, under the GPL licensing terms, that implements two algorithms - Round-Robin and Client Hashing,
– HAProxy [13] - this solution implements standard Round Robin algorithm and others - Source Hashing and Destination Hashing,
– Lighttpd [14] - this is one of the famous and efficient web server, but also proxy balancer, that implements four algorithms: Static (fail-over), Round Robin, Cache Array Routing Protocol (similar to Source Hashing) and Shortest Queue First.

3 Features of the web servers

Performance is one of the most important aspects of scalable web services. To determine the performance the following factors should be considered:

- average response time (t_{avg}),
- minimum response time (t_{min}),
- throughput (th).

The average response time is a factor which value varies and depends on users load. Its value increases with the number of users.
The minimum response time is the minimum time in which a web request is completed. It depends on the server performance (hardware, software) and the request type. It can be constant for the same conditions. It should be measured at minimal load of a server.
The throughput is a very authoritative factor that describes the performance of a server. It tells how many requests can be processed at the unit of time at the saturation. However, the system that reached the maximum throughput cannot guarantee the acceptable response time.

3.1 The efficiency of request processing

The requests that are processed by a server can be of two types:

- independent request,
- related request.

The independent requests are requests, that do not affect one another. They share resources (for example CPU), which are shared fairly between them. At saturation of a web server with one processor the relationship of minimum response time and throughput for this case can be given by the efficiency factor (1):

$$E_1 = t_{min} * th. \tag{1}$$

The efficiency factor E_1 for series of identical independent request should have the value close to 1.0. For multiprocessor servers the value of E_1 factor should be close to the number of processors. In this case the efficiency factor is given by formula (2), where N - number of processors.

$$E = t_{min} * th/N. \tag{2}$$

Another type of queries are the related. Mechanisms of optimization like caching or spooling can make processing shorter for a group of requests. This is for example the case of queries with pool connections to databases [5, 6]. In this case the value of efficiency factor is above the number of processors. Some queries can generate a large overhead (for example allocating and deallocating big data tables, frequent context switching) - requests affect each other and the efficiency factor is below the number of processors.

3.2 CT2TEE - an example of a web server

CT2TEE is a novel Internet-based simulator for transesophageal echocardiography (TEE) training [7]. It loads CT data into the memory and processes projections [8] and USG simulations for each request individually. The process of creating a projection is described on Fig. 2.

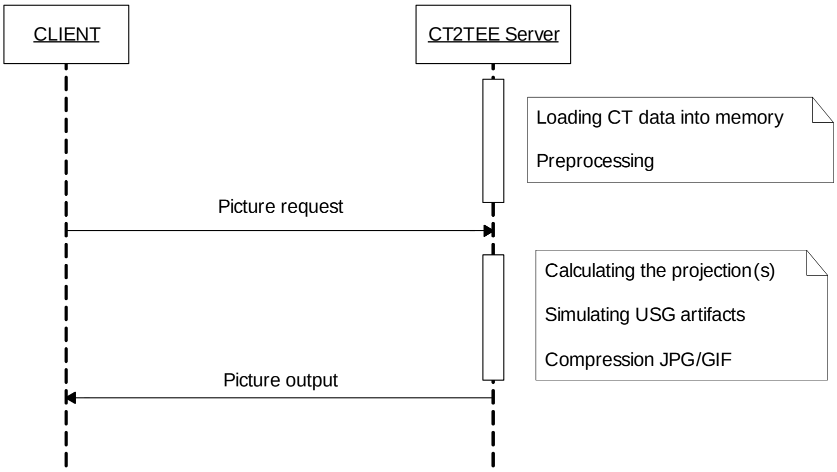

Fig. 2. Diagram for a request processing by CT2TEE server.

The output of CT2TEE application can be an image, that is a single frame (of JPG or GIF format, JPG quality: 80%) or an animation (of GIF format). The current version of CT2TEE generates the same projection with different noise pattern, but there will be motion implemented in the future. The GIF format generates bigger files than JPG. The one of the most interesting features of CT2TEE application is a fact, that the efficiency factor (2) in this case on the current Internet server of CT2TEE (2 processors) is very close to value 1 (0.99). It is caused by the character of requests - they are calculations that share CPU only. Therefore the CT2TEE application is a good example to test load balancing on a cluster of servers.

4 Tests

The tests have been carried out to assess performance.

4.1 Hardware and software environment

The following hardware has been used for the tests:

- for web servers/proxy server/test clients: IBM Blade HS 21, CPU: 2.0 GHz, Intel Xeon (8 cores), RAM 16GB,
- network: Ethernet 1Gb, switch.

The following software was used:

- operating systems: Linux Fedora Core 12, Windows Server 2008,
- component environments: Mono 2.4.2.3, .NET 2.0,
- web servers: Mono XSP 2.4.2, IIS 7.0,
- load balancers: Apache Mod Proxy 2.2, NGiNX 0.7.65, Pound 2.5-1, Inlab Balance 3.52, HAProxy 1.4.1 and Lighttpd 1.4.26,
- load testers: JMeter, Apache Bench.

The results given by JMeter and Apache Bench were very similar, so we decided to use JMeter in all cases. The tests were divided into two parts:

- determining individual parameters of servers,
- determining performance of load balancing.

4.2 The efficiency of servers

Initially the tests for the main parameters of cluster servers have been carried out. The results (minimum times of requests t_{min} [ms], throughputs th [req/s] and efficiency factors E) are presented in Table I.

Table 1. Minimum times of requests t_{min} [ms], throughputs th [req/s] and efficiency E for cluster servers with CT2TEE application.

OS	server	G0			G1			G4		
		t_{min}	th	E	t_{min}	th	E	t_{min}	th	E
Linux XSP	s1	193	33.1	0.799	203	38.5	0.977	760	10.0	0.950
	s2	188	34.1	0.801	200	37.6	0.940	762	10.4	0.991
	s3	190	34.8	0.827	201	36.8	0.925	767	10.3	0.988
	s4	191	33.4	0.797	197	36.8	0.906	776	10.1	0.980
WinSvr IIS	s1	140	52.5	0.919	137	47.4	0.812	517	14.9	0.963
	s2	141	49.8	0.878	137	48.2	0.825	519	15.2	0.986
	s3	138	53.2	0.918	136	46.3	0.787	519	14.9	0.967
	s4	139	51.0	0.886	137	50.3	0.861	518	14.8	0.958

4.3 The performance of load balancing

Experiments for the six solutions (Apache Mod Proxy, NGiNX, Pound, Inlab, HAProxy and Lighttpd) were done. In the case of Apache Mod Proxy we tested all three algorithms: Request Counting (RC), Weighted Traffic Counting (WTC) and Pending Request Counting (PRC). In the case of Inlab we tested only the

Round Robin algorithm. In the case of HAProxy we tested three algorithms: Round Robin (RR), Source Hashing (SRC) and Destination Hashing (URI). In the case of Lighttpd we tested four algorithms: Cache Array Routing Protocol (CARP), Round Robin (RR), Static (failover balancing, STAT) and Shortest Queue First (SQF). There were two kinds of heterogeneous environments:

– 3 servers running Linux+XSP and 1 server running Windows+IIS,
– 1 server running Linux+XSP and 3 servers running Windows+IIS.

We selected output to be a JPG (G0, small files of average size 20 KB) and GIF (G1, bigger files of average size 80 KB for 1 frame and 300 KB for 4 frames - G4). The G4 output was processed much longer than others.

The results are presented in table II and on the plots (Fig. 3, 4). To compare these results with the maximum performance of a system an additional column (MAX) is placed. It contains sums of all server throughputs for the tested cases.

Table 2. The performance of load balancing - throughputs [req/s].

Load Balancer	1 WinIIS + 3 LinXSP			3 WinIIS + 1 LinXSP		
	G0	G1	G4	G0	G1	G4
Pound	131.50	138.40	37.80	129.10	150.20	41.60
NGiNX RR	136.30	150.40	39.70	141.00	152.60	39.20
NGiNX FAIR	132.20	133.40	38.60	131.50	144.30	39.10
Inlab	155.40	155.40	47.10	195.40	189.30	50.80
Apache WTC	67.10	79.80	17.30	105.30	91.30	25.70
Apache RC	111.40	115.60	30.90	139.60	124.60	38.80
Apache PRC	150.20	145.30	45.70	194.30	151.40	48.40
HAProxy RR	133.90	131.70	38.70	130.00	122.90	37.60
HAProxy SRC	36.70	39.90	16.70	36.90	40.10	10.40
HAProxy URI	37.40	40.10	10.40	57.40	59.50	15.30
Lighttpd CARP	65.10	74.00	19.10	88.50	94.60	24.90
Lighttpd RR	128.20	132.50	36.80	156.90	155.60	40.30
Lighttpd STAT	88.50	91.00	24.00	101.30	101.50	26.60
Lighttpd SQF	158.70	167.10	43.80	172.90	171.80	48.80
MAX	165.30	168.50	47.60	211.10	190.30	56.40

5 Summary

The tests have proved that the use of proxy load balancers effectively increases system throughput. Some of the servers provide several algorithms, the choice of one of them is crucial for performance. For tested solutions using the CT2TEE application server the best results are reached by Inlab, Lighttpd with Shortest Queue First algorithm and Apache Mod Proxy with Pending Request Counting algorithm. Slightly smaller througputs were achieved for the other solutions

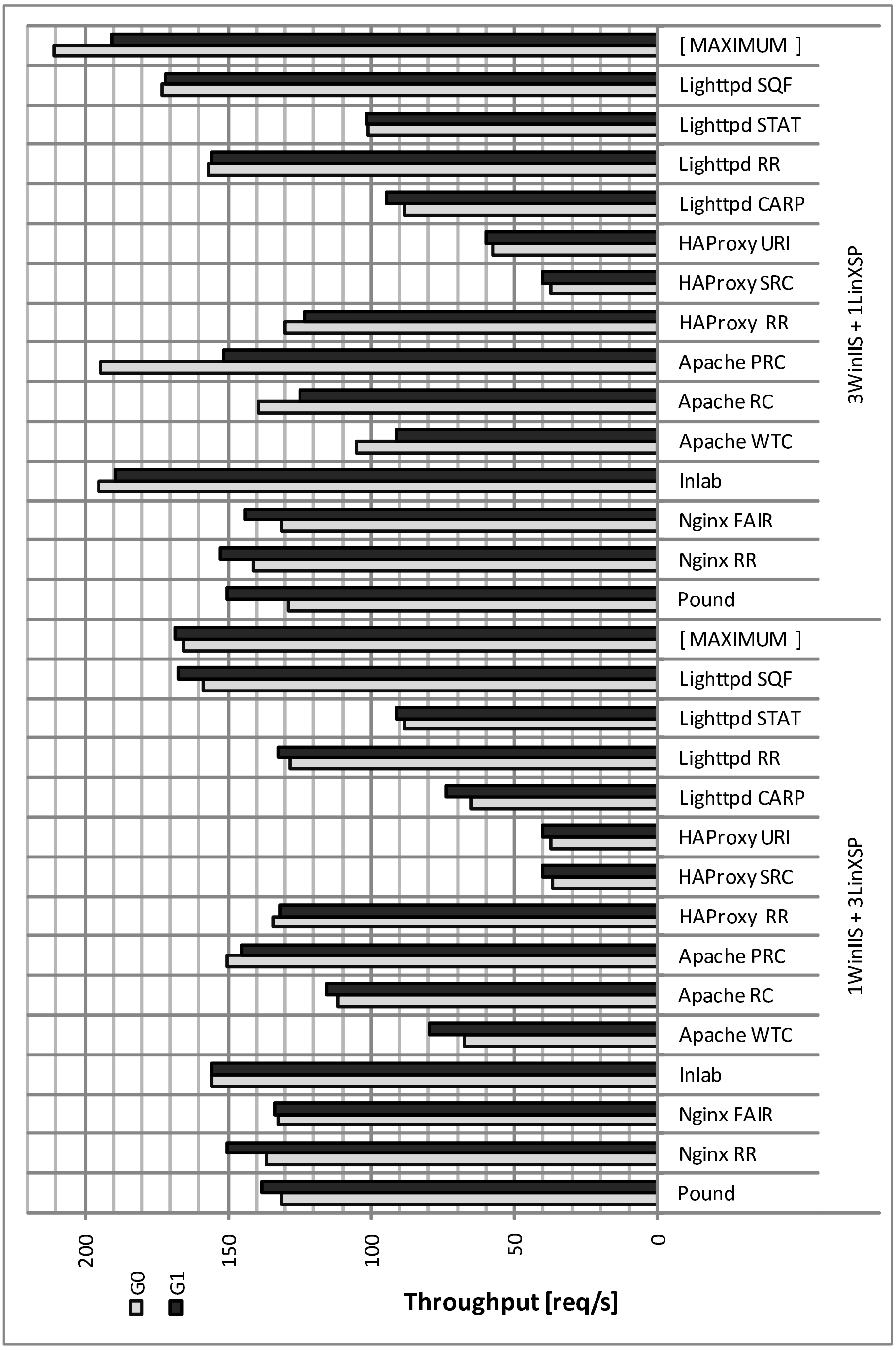

Fig. 3. The results of load balancing tests for G0 (JPG) and G1 (GIF, 1 frame) output.

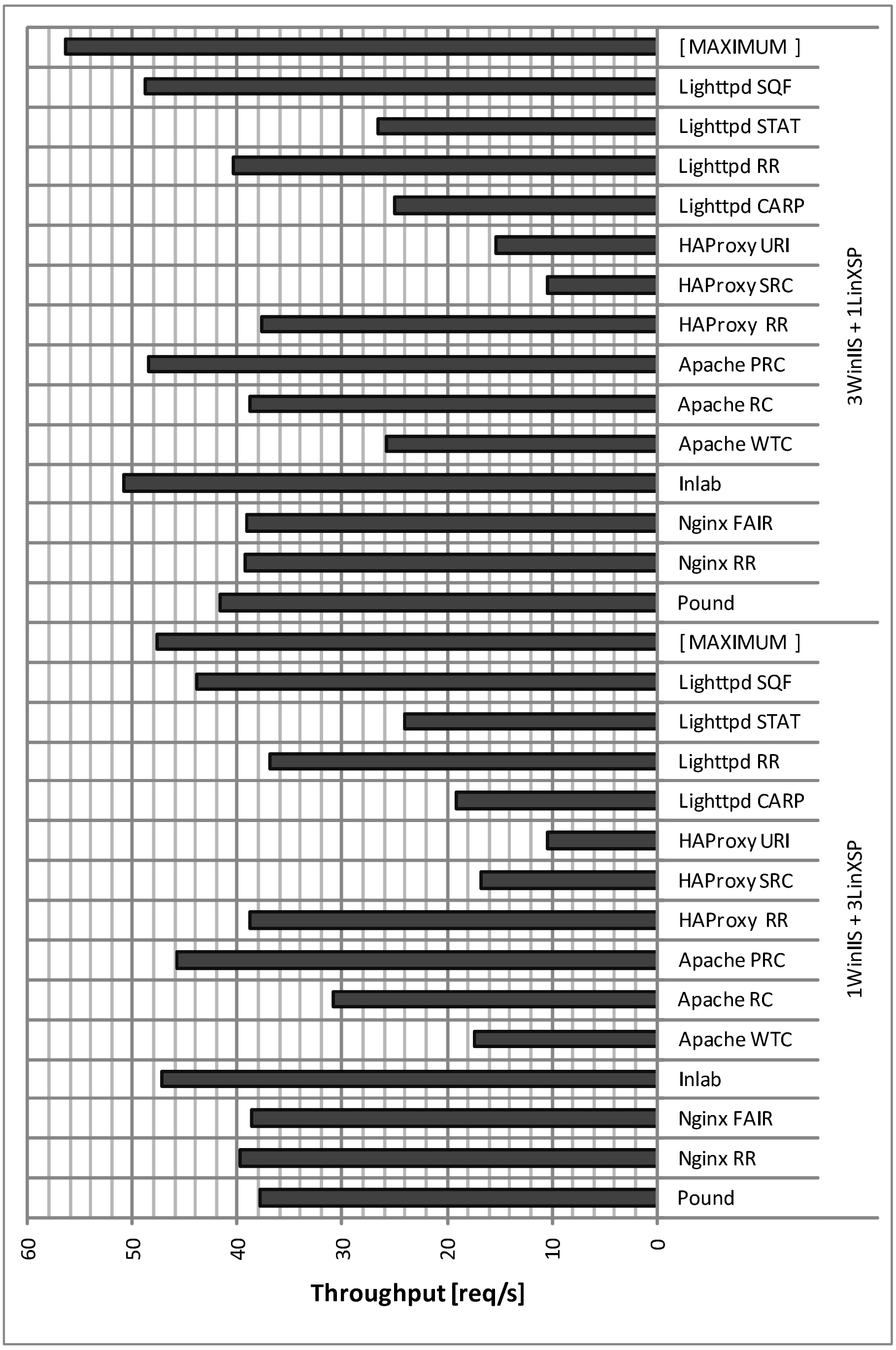

Fig. 4. The results of load balancing tests for G4 (an animated GIF, 4 frames) output.

that use Round Robin algorithm - NGiNX (Round Robin and Fair algorithms), Pound, HAProxy (with Round Robin algorithm) and Lighttpd (with Round Robin algorithm). The worst results were produced by proxy balancers with Source Hashing and Destination Hashing algorithms. Apache Mod Proxy with Weighted Traffic Counting algorithm is over two times slower than the best results, but this algorithm is better in case of variable size of outputs. As we proved the choice of solution and algorithm is very important to reach the maximum performance of web server clusters.

Acknowledgment. This work was financed by the AGH - University of Science and Technology, Faculty of Geology, Geophysics and Environmental Protection as a part of statutory project number 11.11.140.561.

References

1. Teo, Y.M. and Ayani, R., Comparison of load balancing strategies on cluster-based web servers. Simulation. vol. 77 issue 6. p. 185–195 (2001).
2. Guo J., Bhuyan L. N.: Load Balancing in a Cluster-Based Web Server for Multimedia Applications. IEEE Transactions On Parallel And Distributed Systems, vol. 17, no 11, (2006).
3. Ungureanu V., Melamed B. and Katehakis M.: Effective load balancing for cluster-based servers employing job preemption. Performance Evaluation, vol. 65, issue 8, p. 606–622 (2008).
4. Wrzuszczak, J.: Auction mechanism in management of processing nodes in a computer cluster. Contemporary Aspects of Computer Networks 2, 259-265 (2008).
5. Bogardi-Meszoly A., Szitas Z., Levendovszky T. , Charaf H.: Investigating Factors Influencing the Response Time in ASP.NET Web Applications. Lecture Notes in Computer Science (LNCS), Springer, vol. 3746., pp. 223-233 (2005).
6. Gabiga A., Piórkowski A., Danek T.: Efficiency analysis of servlet technology in selected database operations. Studia Informatica, nr 84, vol 30 issue 2B, (2009).
7. Kempny A., Piórkowski A.: CT2TEE - a Novel, Internet-Based Simulator of Transoesophageal Echocardiography in Congenital Heart Disease. Kardiol Pol 2010; 68: (2010).
8. Piorkowski A., Jajesnica L., Szostek K.: Creating 3D Web-Based Viewing Services for DICOM Images. Computer Networks, 16th Conference, CN 2009, Wisla, Poland, June 16-20, 2009, Communications in Computer and Information Science, Springer Berlin (2009).
9. Mod Proxy Balancer - Apache HTTP Server,
   `http://httpd.apache.org/docs/2.1/mod/mod_proxy_balancer.html`
10. Pound - Reverse-Proxy and Load-Balancer, `http://www.apsis.ch/pound/`
11. NGiNX - HTTP and reverse proxy server, `http://nginx.org/en/`
12. Inlab Balance, `http://www.inlab.de/balance.pdf`
13. HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer,
   `http://haproxy.1wt.eu/`
14. Lighttpd - fly light, `http://www.lighttpd.net/`