World Wide Web (2011) 14:105–131
DOI 10.1007/s11280-010-0101-5

An up-to-date survey in web load balancing

Katja Gilly · Carlos Juiz · Ramon Puigjaner

Received: 6 October 2009 / Revised: 21 October 2010 /
Accepted: 19 November 2010 / Published online: 2 December 2010

Abstract This survey is an up-to-date state-of-the-art in Web load balancing mechanisms that includes all the possible classifications and focuses on the advantages of using load balancing solutions to increase the performance of the Web system. A general description of the Web load balancing solutions is included and organised by differentiating the OSI protocol stack layer the load balancing is based on. The most important request distributing polices that are proposed in the literature are also included. This article summarises all previous surveys on the Web load balancing subject and updates the state-of-the-art with the most recent load balancing proposals.

Keywords internet · performance · web load balancing

1 Introduction

The main reason for the increasing popularity of server-based clusters, also called server farms, is due to the fact that Web applications must be able to run on multiple servers in order to accept an increasing number of users that demand Web content.

This work was partially funded by the Spanish Ministry of Education and Science under Grant TIN2006-02265.

K. Gilly (✉)
Miguel Hernández University, Avenida de la Universidad, s/n, Elche, Alicante, Spain
e-mail: katya@umh.es

C. Juiz · R. Puigjaner
University of Balearic Islands, Cra. de Valldemossa, km 7.5, Palma, Balearic Islands, Spain

C. Juiz
e-mail: cjuiz@uib.es

R. Puigjaner
e-mail: putxi@uib.es

In essence, load balancing is the ability to make several servers participate in the same service and do the same work, since the capacity of servers is finite. This implies important benefits such as *scalability*, *availability*, *manageability* and *security* of Web sites. First and foremost, load balancing improves *scalability* of an application or server cluster by distributing the load across multiple servers. Load balancing is also able to direct the traffic to alternate servers if a server or application fails. The ability of maintaining service unaffected during a predefined number of simultaneous failures is called *availability*. *Manageability* is improved by load balancing in several ways by allowing network and server administrators to move an application from one server to another easily. Last, but not least, load balancing solutions provide *security* improvement by protecting the server clusters against multiple forms of Denial-of-Service (DoS) attacks, as the request that arrives to the Web system can be analysed before sending it to a server for being served.

The rest of this article is organised as follows:

– Section 2 briefly revises the load balancing classifications that appear in the literature.
– Section 3 introduces the concept of scalability as a main requirement of a modern Web-based system.
– Section 4 covers the load balancing scheduling solutions that are based on the Transmission Control Protocol (TCP) layer, also called layer-4 or content-blind load balancing solutions.
– Section 5 deals with the architectures that balance the load based on the application layer, also called content-based platforms.
– Section 6 describes the existing content-aware distribution policies.
– Finally, we include the conclusions and open problems.

2 Load balancing classifications

Several classifications of load balancing techniques exist in the literature. In this section we sum them up.

– Depending on the taxonomy of the Web-server architectures, a distinction is made between the *local scale-out* and *global scale-out* approach [11]. The main difference is based on the geographical locations where the set of server nodes resides. In the global scale-out approach, the nodes are located at different geographical locations while the nodes are in the same location in local scale-out architecture. An example of a global scale-out organisation are Content Distribution Networks (CDNs) [7, 47]. The local scale-out architectures are also called *locally distributed Web systems*.
– Another distinction can be made in this group depending on the visible Internet Protocol (IP) addresses the Web system presents to the client. If the IP addresses of the Web server nodes are visible to the clients, then we are referring to a *Distributed Web System*. When the only visible address to the client application is the Virtual IP (VIP) of the front-end of the Web system, then the architecture is a *Cluster-based Web System* (or Web cluster).
– Cardellini et al. in [12] proposed a classification depending on where the distribution decision is taken when routing a request to one server of a locally distributed

Web-server system: at the *client*, at the *Domain Name System (DNS)*, at the *network* and at the *Web system*. In this work, we only consider the last option, dispatcher-based clusters, where a dispatching entity of the Web system receives the incoming requests and distributes them among the servers. More information about the other options can be found in [6, 10, 12, 26, 41]. Considering the dispatcher-based clusters, the dispatching policies are also organised in [12] depending on the information used to select the target server. Hence, they distinguish between *client-aware* and *server-aware* policies. Client-aware mechanisms consider the content of the incoming request; e.g. Client-Aware Policy (CAP) [14] that estimates the impact a request will have on the system resources and classifies it consequently. Server-aware policies take into account the servers' status when making distribution decisions. We deal with these mechanisms in Section 6.

- The classification proposed by Choi in [21] depends on the level the load balancing is applied to: *hardware level* (referred basically to commercial products), *system software level* (also called kernel level, as it implies modifications at the Operating System (OS) level of the Web system), *middleware software level* and *application software level*. Most implementations of load balancing mechanisms are at application level (e.g. [49]) or kernel level (e.g. [45, 73, 81]). Despite being a very interesting classification, we do not use it in this work as sometimes the proposals that appear in the literature do not specify the level they are implemented in. However, we introduce in this survey the implementation details of the proposals when they are available in the original documentation.
- Focusing on Web cluster load balancing, it is also quite usual to group the load balancing solutions depending on the Open Systems Interconnection (OSI) protocol stack layer the load balancer, also named Web switch or front-end, is based on. In the next sections we classify the load balancing solutions by distinguishing the layer they are based on.
- Also referring to Web cluster load balancing, there is an alternative classification depending on the return way of the data flow from the object server to the client. The response from the server can either go through the load balancer (*two-way* architecture) or it can follow an alternative path directly to the client avoiding the load balancer (*one-way* architecture or single arm server load balancing). This last one mainly consists of a different path of traffic flow from the server to the client rather than passing through the load balancer in order to avoid a possible bottleneck in the load balancer.

3 Introducing scalability in load balancing

Among all the reasons for using load balancing solutions, we are going to focus on scalability as user demands placed on Web services continue to grow and Web server systems are becoming more stressed than ever. There should therefore no longer be a limit on the performance of an application that is running on a single server. Load balancing avoids this bound by the ability of growing the number of servers that host the application.

Even though both network and server capacities have improved in recent years and new architectures have been developed, there are still some problems to be solved from the user point of view in terms of perceived response time. When a server is congested, the response times obtained by the user increase and this can lead to a lost sale operation when we are referring to an e-business site. Therefore response time continues to challenge the server system and cluster related research.

We are going to focus in this paper on the Web system infrastructure as it is the only component that can be under the direct control of the site administrator in a distributed network system as is the Internet. Other elements that compose the network such as DNS systems, backbones and routers are not controllable by a single organization and are beyond the scope of this work.

The Web system architecture we consider consists of a collection of server computers that are locally distributed and interconnected through a high speed network. This architecture provides a single interface to the outside; hence, it can be seen as a single host. The users are not aware of the names and addresses of the servers that compose the Web architecture, they access the applications hosted in the system directing their requests to the VIP address corresponding to the device that acts as the front-end of the Web architecture. This kind of architecture is also named Cluster-based Web System [11].

4 Content-blind load balancing

The load balancing solutions covered in this section are called content-blind because the load balancer is unaware of the application information contained in incoming requests. Load balancers that perform content-blind routing are normally referred to as layer-4 load balancers. The selection of the target server that is going to attend the request is done based on the information contained in the TCP SYN packet at the load balancer. The OSI layer used to forward the incoming packet to the target server can be either the link or the network layer.

Table 1 summarises all the solutions covered in this section.

We have divided this section into three subsections:

– Sections 4.1 and 4.2 detail the techniques that balance the load depending on the OSI layer (-2 or -3, respectively) the front-end uses to forward the packets to the servers.
– Section 4.3 details the scheduling policies that can be applied in a content-blind load balancing solution.

4.1 Layer-2 forwarding

Layer-2 forwarding (or *bridging Server Load Balancing (SLB)*) is the most simplistic solution for load balancing and can be considered when all interfaces of the Web system architecture are in the same Virtual LAN (VLAN) and IP network, including the client-side router. There is no need to alter the topology of the network or

Table 1 Content-blind load balancing solutions.

	Layer-2 forwarding	Layer-3 forwarding
Two-way		– Network address translation
One-way	– Direct routing	– IP tunneling

redefine the IP addressing map. It is only necessary to include a bridge device between the client-side router and the server side.

The client basically establishes a TCP connection with the server that is going to attend its request through the VIP of the Web site. The function of the front-end device, that acts as the load balancer, is to select the server and translate the destination Media Access Control (MAC) address to leave no evidence that there is an intermediary device in the communication [77].

Figure 1a illustrates an example of layer-2 forwarding, detailing the role of the load balancer that has to re-write the layer-2 destination address to the MAC address of the selected Web server and then forward the incoming request to that server. The destination MAC address of the response that comes from the server has to be re-written with the MAC address of the dispatcher in two-way implementations. It is important to notice that the load balancer does not change the IP address of the incoming request because all the devices in the IP subnet (the load balancer and the servers) share the same IP address. Hence, there is no need to change the network information of the packet and consequently, there is also no need to recompute the IP checksum, which means less overhead. It is important to disable the Address Resolution Protocol (ARP) when using layer-2 forwarding to avoid a possible collision because the same IP address has been assigned to all the nodes of the system. A layer-2 load balancer uses the MAC address available in the data link layer information to determine the output interface port for that packet [34, 46].

Layer-2 forwarding has been widely used in commercial solutions in its one-way architecture version, and is normally named *layer-4 switching with layer-2 packet forward* (L4/2) [17, 37, 69] or Direct Routing (DR) [82, 90]. As the same IP address has been assigned to all the devices of the subnet, the outgoing packets can be sent directly from the server to the clients without going through the load balancer. Figure 2a shows the TCP connection establishment with the DR technique.

Some pioneering prototype implementations of one-way L4/2 load balancing were ONE-IP [29] and LSMAC [34]. Nortel Networks also consider this mechanism in their actual Nortel Application Switches [58]. Linux Virtual Server (LVS) is a layer-4 load balancing solution that is included in an open source project that was started by

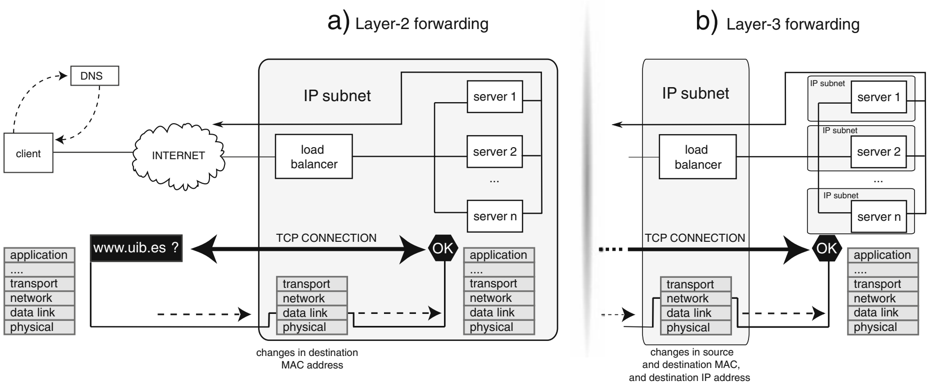

Figure 1 An example of layer-2 and -3 forwarding implementations.

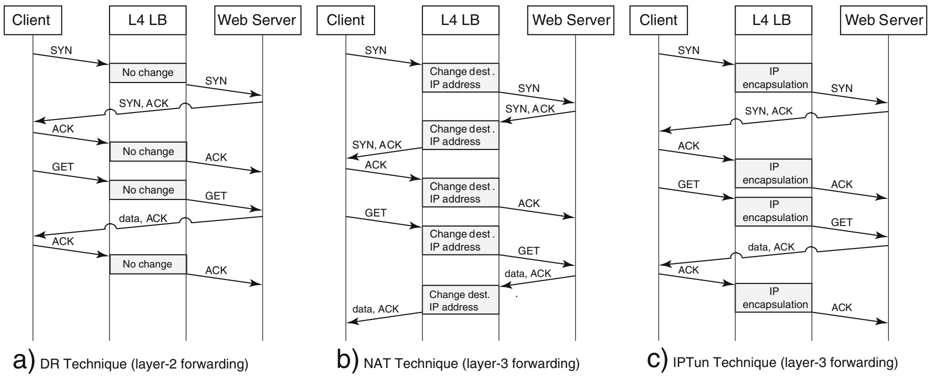
Client
L4 LB
Web Server
SYN
No change
SYN, ACK
ACK
GET
data, ACK
Change dest. IP address
IP encapsulation
a) DR Technique (layer-2 forwarding)
b) NAT Technique (layer-3 forwarding)
c) IPTun Technique (layer-3 forwarding)

Figure 2 TCP connection establishment when using layer-2 and -3 forwarding techniques.

Zhang in 1998 [70, 90]. It can be configured to support DR, and also other forwarding techniques that are included in the next subsection.

4.2 Layer-3 forwarding

Layer-3 forwarding (also called *routing SLB*) differs from bridging SLB in that the client-side router can be in different VLANs and IP subnets in the Web system architecture. In fact, in this case, the load balancer of the Web system architecture is now routing rather than bridging the frames between the client and the server. Figure 1b illustrates an example of layer-3 forwarding and shows that a layer-3 load balancer de-encapsulates a packet up to the network layer to determine where to send the packet.

Two forwarding techniques have been implemented in a dispatcher-based web cluster with layer-3 routing: *Network Address Translation (NAT)* and *IP Tunneling (IPTun)*. NAT is the simplest technique and consists of rewriting the layer-3 destination address of the incoming packet to the IP address of the real server selected by the load balancer. While NAT is implemented in two-way architecture, IPTun is implemented in a one-way architecture. This means that the response from the chosen Web server goes directly to the client. IPTun consists of the encapsulation of IP datagrams within IP datagrams with the source and destination IP address specifying the VIP address of the system and the target server IP address, respectively. More information about NAT and IPTun can be found in [11, 82, 84]. Figure 2b and c also show the TCP connection establishment when using NAT and IPTun, respectively.

The layer-3 forwarding solution based on NAT has also been classified as *layer-4 switching with layer-3 packet forwarding* (L4/3) by some authors [17, 37, 69].

The main disadvantage of this approach is the fact that the load balancer can become the bottleneck of the system as the workload increases. This is due to the overhead of recomputing the IP checksums for every packet that has to go through the load balancer in both ways.

Most of the commercial content-aware switches on the market today also provide NAT forwarding, that is the case of the CSS 11500 Series Content Services Switch of Cisco System, Inc. [23, 24], ServerIron layer 4–7 switches of Foundry Networks [57] and Nortel Layer 2/7 Gigabit Ethernet Switch Module (GbESM) for IBM

BladeCenter [40, 58], for example. The Linux software framework, LVS [70], also can be configured to support NAT and IPTun, but the most efficient mechanism is DR [19, 51]. Microsoft also implements a layer-3 forwarding mechanism named Network Load Balancing (NLB) that is included in the Windows Server 2003 Family [15].

4.3 Content-blind request distribution policies

Content-blind load balancing permits the front-end device to be aware of the TCP connections between the clients and the servers. Hence, the load balancer dispatches the requests according to the IP address and the TCP port. There are several load balancing scheduling policies that are normally used by content-blind load balancers. Some examples of these policies are:

- *Round Robin (RR)* Algorithm: the TCP connections are assigned on a RR basis, with the first connection going to server 1, the second to server 2, and so on. As the connections are assigned sequentially among the servers, each server receives the same number of connections over time independently of how fast it is able to process them. For this reason, RR is one of the best distributing methods for homogeneous servers but, unless used with a per-server weighting, it is less effective in environments where the servers are heterogeneous.
- *Weighted Round Robin (WRR)* Algorithm: the traffic will be assigned to the servers according to their configured relative capacities, in the case that the servers are heterogeneous. The administrator specifies the percentage of traffic to be directed to each of the servers.
- *Least Connection (LC)* Algorithm: connections are assigned to the server with the least number of connections. This is a dynamic scheduling algorithm as the load balancer needs to count the number of connections that are established between the clients and each Web server in the cluster.
- *Weighted Least-Connection (WLC)* Algorithm: similar to LC, in this algorithm apart from counting the number of connections, a weight assigned to each server is also considered. The servers with higher weight will receive a larger percentage of connections than the rest of the servers.
- *Least Loaded (LL)* Algorithm: the dispatcher assigns the next request to the server that has the lowest load. In this case an agent on the server keeps the load balancer updated on the server utilization and capacity. Connections are assigned to the server having the most spare capacity. It is also called baseline algorithm.
- *Random* Server Selection: connections are assigned uniformly among the servers but not in a deterministic sequence.

5 Content-aware load balancing architectures

A content-aware load balancer works at the application layer. This means that the load balancer is aware of the application content of the incoming request. The TCP connection must be established first between the client and the front-end of the Web system. It then receives the HyperText Transfer Protocol (HTTP) request and analyses the content of it (see Figure 3). This makes the content-aware routing more

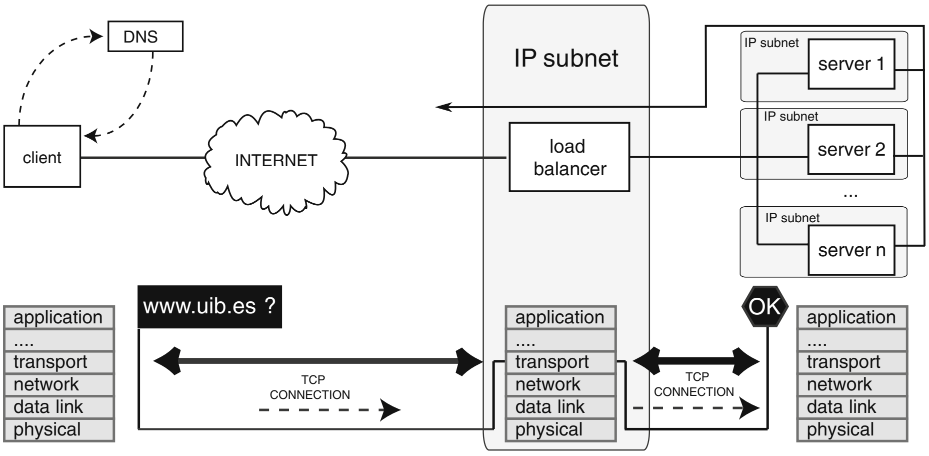

Figure 3 Example of layer-7 forwarding implementation.

specific to applications that can offer differentiated services, but also more complex than the content-blind approach.

Despite the fact that some content-aware load balancing commercial solutions have been implemented, it is a subject that is still being investigated. In this section, we describe and analyse the most recent architectures that implement content-aware load balancing. Some works prove that by using the content of requests and loading information from the servers, more flexible and intelligent distributing algorithms can be developed [1, 3, 11, 14, 18, 49, 60, 62, 74].

Let us introduce the fact that HTTP/1.1 permits persistent (keep-alive) connections [33]. This means that several HTTP requests from a client can go through the same TCP connection. Hence, this causes a reduction in response time, server overhead and network overhead of HTTP [56]. In order to take advantage of these benefits, some TCP modifications have been developed to permit an HTTP request granularity in the content-aware load balancing, instead of a connection granularity. These modifications depend on the one-way or two-way architecture of the Web cluster.

Table 2 summarises the solutions covered in this section, indicating if they include request granularity when using the HTTP/1.1 protocol. We have marked in bold the

Table 2 Request granularity in content-aware load balancing architectures.

	HTTP/1.0	Request granularity in HTTP/1.1
2-way	– TCP splicing [54]	– Asymmetric TCP splicing [44]
	– Redirect flows [27]	– TCP connection binding [83]
1-way	– TCP hand-off [38]	– Multiple connection TCP hand-off [3]
		– Back-end request forwarding [3]
	– TCP Connection hop [65]	– **One-packet TCP state migration to packet filter** [49]
	– **Socket cloning** [73]	– **Multiple-cloning** [73]
		– One-way TCP Splicing [55]
		– One-way connection binding [53]
	– TCP rebuilding [50]	– Multiple TCP rebuilding [51]

options that implement content-aware distribution based on a layer-4 front-end. The variations of these proposals have been omitted in the table.

Let us divide then this section into one-way and two-way architectures as it is crucial to describe their TCP behaviour to know how the load balancing is done. Hence, this section reviews the load balancing proposals depending on the return path of responses from the Web servers.

5.1 Two-way architectures

This subsection introduces three mechanisms to route the requests from the load balancer to the target Web server in two-way architecture, that are: *TCP Connection Binding* [83], *TCP Splicing* [54] and *Redirect Flows* [27].

5.1.1 TCP connection binding

Yang and Luo proposed TCP Connection Binding in [83]. It is also called *TCP Gateway* by other authors [11, 25], *Relaying front-end* in [3] or *Relaying with Packet Rewriting* in [74], and basically consists of maintaining two TCP connections: one between the client and the load balancer, and a second one between the load balancer and the Web server. Before receiving any request, the load balancer establishes a persistent connection with each Web server. When the load balancer receives a request from a client, one of these pre-established TCP connections is used to transfer the request to the selected target server. The main advantage of this proposal is that it permits a content-based distribution at the granularity of individual requests because the persistent connections between the load balancer and the back-end servers do not depend on the incoming traffic [3]. In a later work [52], the authors improve the request distribution and the reliability of the TCP Connection Binding mechanism, and also include a Java implementation of their proposal. The main problem of this approach is that the packets need to be analysed up to their application layer information when passing through the load balancer. This implies an overhead that can be avoided by the other mechanisms detailed below.

Figure 4a shows the TCP Connection Binding procedure, detailing Initial Send Sequence (ISS) number of the client and the Initial Receive Sequence (IRS) number of the load balancer used in the three-way handshake [30] of the client connection with the load balancer. Different numbers, ISS2 and IRS2, are used in the pre-forked connection between the load balancer and the Web server. All these numbers do not need to have any relation at all as the load balancer is responsible for changing the ISS and IRS TCP header field numbers depending on which connection it is using to transmit the packets.

IBM Network Dispatcher [39] is an example of a TCP Connection Binding commercial implementation, but was withdrawn from the market some years ago.

5.1.2 TCP splicing

The second mechanism, TCP Splicing, was proposed by Maltz and Bhagwat in [54]. This proposal is similar to TCP Connection Binding, but the performance is improved due to the fact that the TCP client connection and the TCP server connection with the load balancer are spliced together (at the TCP layer) and all the work can be carried out directly by the OS forwarding the data at the IP level.

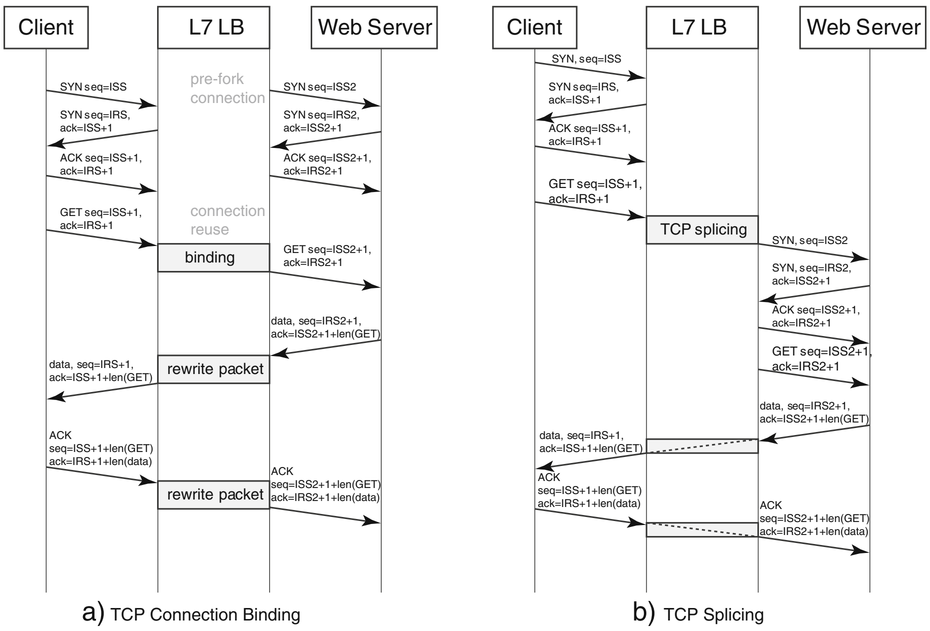
Client
L7 LB
Web Server
pre-fork connection
SYN seq=ISS
SYN seq=IRS, ack=ISS+1
ACK seq=ISS+1, ack=IRS+1
SYN seq=ISS2
SYN seq=IRS2, ack=ISS2+1
ACK seq=ISS2+1, ack=IRS2+1
GET seq=ISS+1, ack=IRS+1
connection reuse
binding
GET seq=ISS2+1, ack=IRS2+1
data, seq=IRS2+1, ack=ISS2+1+len(GET)
rewrite packet
data, seq=IRS+1, ack=ISS+1+len(GET)
ACK seq=ISS+1+len(GET) ack=IRS+1+len(data)
rewrite packet
ACK seq=ISS2+1+len(GET) ack=IRS2+1+len(data)
a) TCP Connection Binding
Client
L7 LB
Web Server
SYN, seq=ISS
SYN seq=IRS, ack=ISS+1
ACK seq=ISS+1, ack=IRS+1
GET seq=ISS+1, ack=IRS+1
TCP splicing
SYN, seq=ISS2
SYN, seq=IRS2, ack=ISS2+1
ACK seq=ISS2+1, ack=IRS2+1
GET seq=ISS2+1, ack=IRS2+1
data, seq=IRS2+1, ack=ISS2+1+len(GET)
data, seq=IRS+1, ack=ISS+1+len(GET)
ACK seq=ISS+1+len(GET) ack=IRS+1+len(data)
ACK seq=ISS2+1+len(GET) ack=IRS2+1+len(data)
b) TCP Splicing

Figure 4 Layer-7 load balancing techniques in two-way architecture: **a** TCP connection binding and **b** TCP splicing.

Figure 4b shows an example of TCP Splicing detailing the ISS and IRS TCP header field numbers of both connections.

Some software and hardware implementation designs of Web switches that use TCP Splicing have been proposed. Cohen et al. in [25] describe the implementation details of a Web switch based on Linux OS. Rosu and Rosu in [67] propose a socket-level implementation of TCP Splicing and compare it to an IP-level implementation on AIX RS/6000 machines concluding that the socket-level implementation provides more flexibility and improves the transfer rates. Other authors like Apostolopoulos et al. [2], Zhao et al. [92] and Kachris and Vassiliadis [42] propose a hardware design for a Web switch based on a PowerPC 603e processor, an Intel IXP2400 network processor and a multi-processor reconfigurable logic platform (Xilinx Virtex 4 FPGA), respectively.

An extension of TCP Splicing is documented by Chang et al. in [16]. It consists in having pre-forked connections between the load balancer and the Web Servers that are spliced to the connections between the clients and the load balancer when a request is received in the system. This approach is very similar to the TCP Connection Binding approach described above.

Kobayashi and Murase in [44] deal with persistent TCP connections trying to provide a request granularity in the load balancing. They propose an asymmetric TCP Splicing that permits it to receive pipelined HTTP requests through a TCP connection. After analysing the application layer of the requests, their proposal changes the target server that is serving the requests coming through that TCP connection for a more appropriated server in case it is necessary.

Similar mechanisms to TCP Splicing have been implemented in some commercial content-aware solutions such as the Cisco CSS 11500 Series Content Services Switch [23, 24], F5's BIG-IP [31], Foundry's ServerIron layer 4–7 switches [57], Radware OnDemand switches [63] and Nortel Layer 2/7 Gigabit Ethernet Switch Module (GbESM) for IBM BladeCenter [40, 58]. A Linux framework has also been developed to support TCP Splicing named Linux Layer-7 switching (L7SW) [76].

5.1.3 Redirect flows

The third and last two-way mechanism, Redirect Flows developed by Colby et al. [27], is very similar to the TCP Splicing approach but based on the NAT architecture [19, 51]. It was a proprietary mechanism of Arrowpoint Communications Inc., a company that was acquired by Cisco Systems Inc. [24] in the year 2000.

5.2 One-way architectures

The main disadvantage of the two-way architecture proposals is that response data must be forwarded by the load balancer that may become the bottleneck of the system when a high volume of traffic needs to be processed.

This subsection describes the approaches that permit the Web servers to return responses directly to clients. We have divided the content-aware architecture solutions in seven main groups based on the original mechanisms proposed to route the requests from a target Web server to clients in one-way architecture.: *TCP Hand-off* [38], *One-packet TCP State Migration to Packet Filter* [49], *TCP Connection Hop* [65], *Socket Cloning* [73], *One-way TCP Splicing* [55], *One-way Connection Binding* [53] and *TCP Rebuilding* [51]. Variations of these original proposals are also considered in each classification group.

With the aim of reducing the overhead produced by the load balancer, some authors propose content-aware load balancing architectures that are based on content-blind platform. This basically means the use of a layer-4 front-end in the Web system that receives the Web client request and assigns it to a Web server. As the content of the request is known once the packet is de-encapsulated in the end-node, the task of distributing or redirecting the request to another node is left to another process of the Web system (normally named *distributor* or *dispatcher*). Examples of this approach are detailed below (e.g. [5, 18, 43, 49, 61, 73, 78, 86]).

It is also important to mention that some of the papers referenced in this subsection propose both a content-aware architecture and a distribution policy. Hence, they might also be cited in the next section, that deals with request distribution proposals.

5.2.1 TCP hand-off

The most popular solution for a layer-7 one-way Web cluster architecture is TCP Hand-off, that was proposed by Hunt et al. in [38]. It requires some modifications in the OS of the load balancer and the Web servers because once the TCP connection between the client and the load balancer is established, the load balancer's end point of the connection is transferred to the selected server, as illustrated in Figure 5a. The IRS number of the first connection and the client's IP address are sent to the Web server because the handed-off connection between the load balancer and the Web server has to be a "copy" of the client's connection in order to permit the Web server

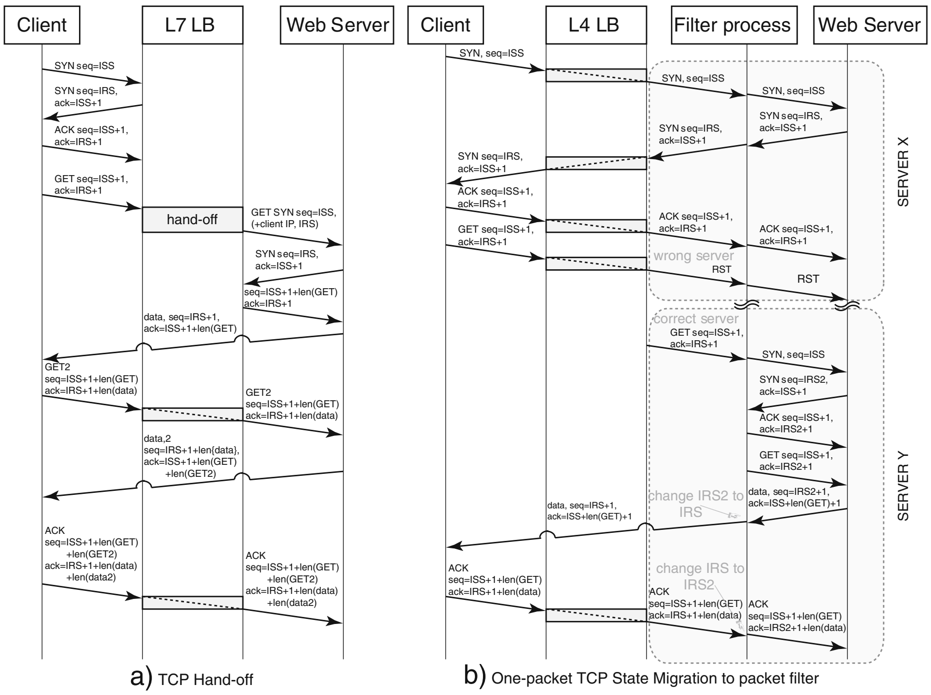
Client
L7 LB
Web Server
SYN seq=ISS
SYN seq=IRS, ack=ISS+1
ACK seq=ISS+1, ack=IRS+1
GET seq=ISS+1, ack=IRS+1
hand-off
GET SYN seq=ISS, (+client IP, IRS)
SYN seq=IRS, ack=ISS+1
seq=ISS+1+len(GET) ack=IRS+1
data, seq=IRS+1, ack=ISS+1+len(GET)
GET2 seq=ISS+1+len(GET) ack=IRS+1+len(data)
GET2 seq=ISS+1+len(GET) ack=IRS+1+len(data)
data,2 seq=IRS+1+len{data}, ack=ISS+1+len(GET) +len(GET2)
ACK seq=ISS+1+len(GET) +len(GET2) ack=IRS+1+len(data) +len(data2)
ACK seq=ISS+1+len(GET) +len(GET2) ack=IRS+1+len(data) +len(data2)
a) TCP Hand-off
Client
L4 LB
Filter process
Web Server
SYN, seq=ISS
SYN, seq=ISS
SYN, seq=ISS
SYN seq=IRS, ack=ISS+1
SYN seq=IRS, ack=ISS+1
SYN seq=IRS, ack=ISS+1
ACK seq=ISS+1, ack=IRS+1
GET seq=ISS+1, ack=IRS+1
ACK seq=ISS+1, ack=IRS+1
ACK seq=ISS+1, ack=IRS+1
wrong server
RST
RST
SERVER X
correct server
GET seq=ISS+1, ack=IRS+1
SYN, seq=ISS
SYN seq=IRS2, ack=ISS+1
ACK seq=ISS+1, ack=IRS2+1
GET seq=ISS+1, ack=IRS2+1
data, seq=IRS2+1, ack=ISS+len(GET)+1
change IRS2 to IRS
data, seq=IRS+1, ack=ISS+len(GET)+1
change IRS to IRS2
ACK seq=ISS+1+len(GET) ack=IRS+1+len(data)
ACK seq=ISS+1+len(GET) ack=IRS+1+len(data)
ACK seq=ISS+1+len(GET) ack=IRS2+1+len(data)
SERVER Y
b) One-packet TCP State Migration to packet filter

Figure 5 Content-aware load balancing techniques in one-way architecture: **a** TCP hand-off and **b** one-packet TCP state migration to packet filter.

to send the responses directly to the client. Hence, the same ISS and IRS numbers have to be used in both connections. Pai et al. in [60] introduce some modifications to Hunt's Hand-off and apply it to their request distribution algorithm (Locality-Aware Request Distribution (LARD)) that is covered in Section 6.1.

Aron et al. include some modifications to the TCP Hand-off mechanism to permit a granularity of individual requests when using HTTP/1.1 persistent connections in [3]. In this work, they propose the *Multiple Connection TCP Hand-off* and *Back-end Request Forwarding* mechanisms. Multiple Connection TCP Hand-off basically permits pipelined incoming requests to be attended by different Web servers by migrating the connection between the servers. Back-end Request Forwarding avoids the overhead introduced by Multiple TCP Hand-off by permitting the redirection of requests from one server to another after a decision of the load balancer. Simple Hand-off is used by both techniques when a new TCP connection request arrives to the load balancer. In [3], the authors study the performance of both methods and conclude that the Back-end Forwarding has better performance for small response sizes while Multiple TCP Hand-off is better for large responses.

In a later work [5], Aron et al. compare the performance of TCP Splicing and TCP Hand-off and show how TCP Hand-off is more scalable with the number of Web servers in the cluster. They also affirm in this work that the TCP Hand-off mechanism has a limited scalability of cluster sizes up to four Web servers, despite implementing one-way architecture. They propose in [5] an alternative distributed load balancing solution with a layer-4 front-end, where a centralised node called the "dispatcher" takes the load balancing decisions and the overhead is carried out by the several distributor nodes that are connected in the network as they distribute the client requests to the selected Web servers by handing off the connections.

Similar content-aware mechanisms based on a content-blind Web switch as [5] are proposed in [18, 43, 61, 86]. *Knowledgeable Node Initiated TCP Splicing (KNITS)* [61] is a proposal of Papathanasiou and Hensbergen that uses NAT to forward the client's requests from the front-end to the Web servers, and multiple hand-off mechanisms are used when a session has to be migrated to another server, splicing the connection between the client and the front-end. The authors implement a prototype within Linux Netfilter infrastructure. The Kerdlapanan and Khunkitti proposal [43] distributes the incoming requests to the servers by multicast through a layer-2 or -3 front-end (when the first SYN datagram is received). One of the servers of the cluster establishes the TCP connection with the client based on a hash function. The distribution decision is done by the servers, after becoming aware of the content requested. If the most appropriate server for the request is not the "connected server", then the connection needs to be transferred by using the Hand-off mechanism. Cherkasova and Karlsson describe a solution in [18] that reduces the forwarding overhead of the Multiple Hand-off in their strategy named Workload-Aware Request Distribution (WARD) (described in Section 6). Zeng-Kai et al. also propose in [86] a fault-tolerant layer-7 load balancing mechanism based on a layer-4 front-end. The content-aware distribution is done by the distributors that are located in the Web servers.

We can observe that most of these authors agree in the need of a layer-4 front-end in the Web system in order to obtain an scalable approach of the content-aware TCP Hand-off mechanism.

Considering request granularity on HTTP/1.1, Kokku et al. suggest another mechanism based on Hand-off, named *Half-pipe Anchoring*, that permits the distribution between different servers of individual requests that come through the same TCP connection [45]. Their proposal is based on the consideration of a TCP connection as two unidirectional half-pipes, one from the Web cluster to the client (data pipe) and one from the client to the Web cluster (control pipe). The selection of a different server to attend a pipelined request is possible by changing the origin of the data pipe of a connection. The authors extend the TCP header by including more TCP options in the message structure and implement a prototype in the Linux kernel. Comparison results between this proposal and KNITS [61] show that Half-pipe Anchoring has a maximum of 25% of the overhead produced by KNITS. However, the main drawback of Kokku's et al. proposal is the modification of the TCP protocol. Another implementation of TCP Hand-off that considers STREAMS-based TCP/IP is presented in [79]. This solution does not require any modification on the TCP/IP code.

TCP Hand-off has been implemented in Linux in the TCPHA project [81]. In this subproject of LVS, a layer-7 kernel level is implemented for Linux. Persistent HTTP connections are also supported in this one-way content-aware load balancing solution.

5.2.2 One-packet TCP state migration to packet filter

The second mechanism, One-packet TCP State Migration to Packet Filter, was proposed by Lin et al. in [49]. They include a Packet Filter process in each Web server to intercept the connection from the load balancer without modifying the OS kernel and implement their proposal in LVS [70]. In order to provide more scalability, a pre-allocation scheme is also proposed in this work. It establishes a TCP connection with a Web server when receiving the first SYN (acting as a content-blind load balancer). Once the HTTP request is received, the load balancer determines if the selected server is correct or not. If the server is the right one, then the request is attended. If it is wrong (possibly because the content the client is asking for is not stored in the selected server) then the load balancer redirects the request to a more appropriate server after a three-way handshake connection establishment as illustrated in Figure 5b. A RST packet is sent to the previous server in order to keep this connection silent. Request granularity through TCP persistent connections is also supported. In this proposal, a reduction of overhead is achieved when a connection that was previously used is needed again, as the three-way handshake can be avoided because the connection was previously established and it is in a waiting status after the RST.

5.2.3 TCP connection hop

TCP Connection Hop is a proprietary solution of Resonate [65] and the core of their Resonate Central Dispatch. A software component called Resonate Exchange Protocol (RXP) has been developed to provide several useful functions [64]. It is installed in all the components of the cluster. RXP establishes the TCP connection with the client and performs the load balancing by transferring the connection to the selected Web server. The mechanism used to transfer the connection to the server is based on the TCP connection migration protocol proposed in [9] and is proprietary of Resonate [65]. It is a similar mechanism to TCP Hand-off and is also referred to as TCP State Migration in [53].

5.2.4 Socket cloning

Sit et al. propose a mechanism to redirect the processing of a client request from one server to another by migrating to an open socket. This mechanism is named Socket Cloning and is proposed in [73]. The load balancer the authors consider in this work is a layer-4 switching with layer-2 packet forwarding that makes the distribution decision when the client SYN packet is received (see Figure 6a). If the decision was not correct and the request has to be attended by another server, then the socket is cloned to it. Hence, Socket Cloning supports HTTP/1.1 persistent connections by cloning multiple times the connection (*Multiple-Cloning*). A synchronisation process between the original and the cloned socket needs to be performed in order to update the sequence and ACK numbers after a response packet is sent. This synchronisation process is done by the packet router (that is an additional layer in all the nodes' architecture proposed by the authors), therefore it does not involve additional inter-node communication. The implementation of Socket Cloning requires some modifications to the OS kernel.

Takahashi et al. in [78] propose a similar solution. They use a layer-3 NAT forwarding mechanism to perform a layer-7 load balancing that provides TCP session redirection between the Web servers that compose the Web cluster, named *TCP-migration* by the authors. They physically separate the packet forwarding and request

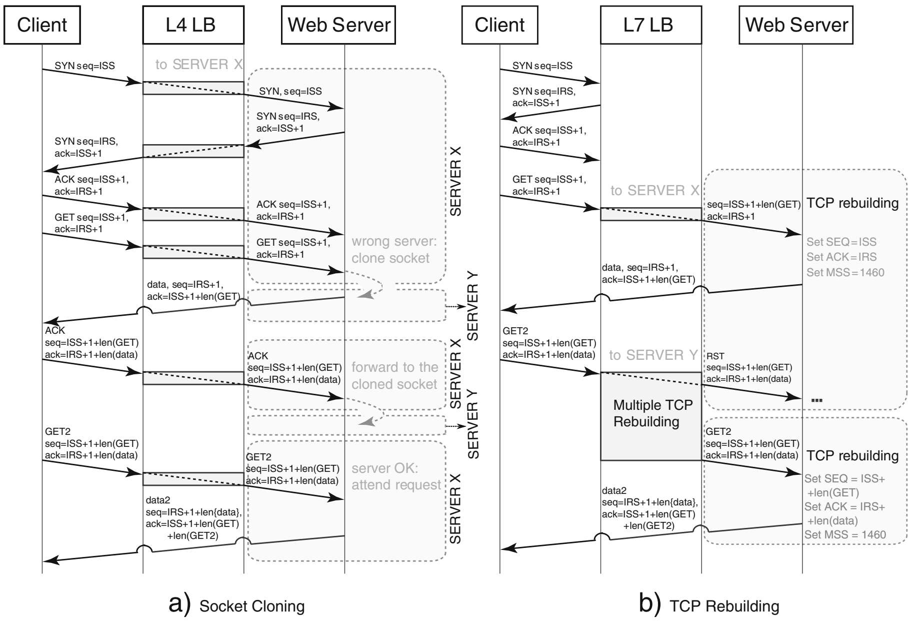

Figure 6 Content-aware load balancing techniques in one-way architecture: socket cloning and TCP rebuilding.

dispatching mechanisms. An implementation in the Linux kernel is also described and its results are reported in this paper.

Very similar mechanisms have been designed to avoid a failover [75, 89]. Snoeren et al. in [75] implement an architecture in Linux that provides a connection failover mechanism. The dispatching decision is taken in the Web servers, which can migrate the connection with the client to another, more appropriate server. Zang et al. design a transparent TCP connection failover mechanism to recover connections that were lost due to a failure in the server that is attending Web requests [89]. They propose that each connection in the Web cluster system has to be visible by at least two servers: the primary server and one or more backup servers that will act in case of a failure of the primary. In both cases [75, 89], the backup servers need constant synchronisation with the primary server.

5.2.5 One-way TCP splicing

Marwah et al. [55] propose some enhancements to the two-way TCP Splicing mechanism that include the possibility of splitting the splice in the Web server and then send the responses directly to the clients. This permits the implementation of one-way architecture when using TCP Splicing. The authors also propose in [55] to re-splice an already established TCP connection with one server to attend an incoming request in another (possibly more appropriate) target server, which permits request granularity in the load balancing instead of TCP connection granularity when using HTTP/1.1. The authors also provide a prototype implementation in Linux that includes some of the enhancements to the TCP Splicing mechanism they propose.

5.2.6 One-way connection binding

The One-way Connection Binding mechanism has some similarities with TCP Connection Binding described in the previous section (see Figure 4a). It is introduced by Luo et al. in [53] and solves the problem of persistent connections in content-aware routing mechanisms by establishing long-lived connections on the server side. When a request arrives to the front-end of the system, the client-side connection is bound to an appropriate server connection, and the request is sent to the server to be attended. The server-side connection is reused by other requests avoiding the connection migration and termination that would require the execution of the slow-start mechanism and, consequently, the loss of performance that would involve. The authors compare the performance of their mechanism to the original TCP Hand-off and show how their solution reduces the overhead in the Web servers.

5.2.7 TCP rebuilding

The seventh and last mechanism is TCP Rebuilding which was proposed by Liu et al. in [50, 51] to be implemented in a LVS with Content-Aware Dispatching (LVS-CAD) platform (that is described in Section 6). Figure 6b details its procedure: when the connection between the client and the load balancer is established and the HTTP request has arrived to the load balancer, it is sent to the selected Web server which starts rebuilding the TCP connection without the need of interchanging any more packets. Thus, the Web server has to guess the sequence and ACK number used in the client connection. The Maximum Segment Size (MSS) is set to the standard value of 1460. In order to handle HTTP/1.1 persistent connections, the authors propose the *multiple TCP Rebuilding* mechanism in [51]. As LVS cannot perform content-aware distribution, they have introduced a fast TCP module handshaking in the IP-layer of the front-end so that it can establish the connection with the client in order to learn the request type.

6 Content-aware request distribution policies

After studying the content-aware load balancing architectures, let us analyse in more detail the content-aware request distribution policies that have been proposed in the recent literature. We have considered a distinction between the policies that try to exploit the cache of the system, named locality-aware solutions, and the ones that do not include the cache performance in the evaluation, named non locality-aware solutions. We have also distinguished the policies that consider Quality of Service (QoS) in the last subsection.

Table 3 sums up, in chronological order, the policies that are described in this section including some of their most important characteristics. As most of the distribution policies proposed in the literature are related to a particular Web cluster architecture, we have also detailed in this table the main characteristics of the architecture proposed by the authors.

Table 3 Content-aware request distribution policy characteristics: (1) Year; (2) Locality-aware; (3) QoS-aware; (4) Admission Control; (5) Dynamic content considered in the distribution policy; Proposed architecture characteristics: (6) Request granularity with HTTP/1.1; (7) One-way architecture; (8) Web Switch layer; (9) Load balancing architecture.

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
LARD [60]	98	Y	N	N	N	N	Y	7	TCP Hand-off
extLARD [3]	99	Y	N	N	N	Y	Y	7	TCP Hand-off
HACC [91]	99	Y	N	N	Y	N	N	7	TCP Splicing
FLEX [17]	01	Y	N	N	N	N	Y	3	DNS load balancing
WARD [18]	01	Y	N	N	N	Y	Y	4	RR-DNS, multiple TCP hand-off
TAP^2 [62]	01	Y	N	N	N	N	N	7	TCP connection binding
PRESS [13]	01	Y	N	N	N	Y	Y	4	TCP hand-off
CAP [14]	01	N	N	N	Y	N	Y	7	TCP hand-off
EQUILOAD [22]	01	Y	N	N	N	N	N	?	Not specified
ADAPTLOAD [66]	01	Y	N	N	N	N	N	?	Not specified
E-FSPF [71]	02	N	Y	N	N	N	N	7	TCP splicing
FARD [8]	03	N	N	N	Y	N	N	7	TCP connection binding
Gage [48]	03	N	Y	N	N	N	Y	7	TCP splicing
BCB and SAA [80]	03	Y	N	N	N	Y	Y	7	TCP hand-off
Cyclone [74]	04	Y	N	N	N	Y	Y	4	Socket cloning and multiple-cloning
ALBM [21]	04	N	N	N	Y	N	N	4	DR and NAT
ADAPTLOAD v2 [88]	05	Y	N	N	Y	N	Y	7	Not specified
D_EQAL [87]	06	N	N	N	N	N	?	?	Not specified
Weblins [32]	06	Y	N	N	N	N	Y	4	TCP hand-off
Around *k*-bounded [59]	06	N	Y	N	Y	Y	Y	4	TCP hand-off
NPSSM [68]	06	N	N	N	N	N	Y	7	Not specified
CAWLL [51]	07	N	N	N	Y	Y	Y	7	Multiple TCP rebuilding
xLARD/R [51]	07	Y	N	N	N	Y	Y	7	Multiple TCP rebuilding
CAHRD [51]	07	Y	N	N	Y	Y	Y	7	Multiple TCP rebuilding
CWARD/CR [19]	08	Y	N	N	N	Y	Y	7	Multiple TCP rebuilding
CWARD/FR [19]	08	Y	N	N	N	Y	Y	7	Multiple TCP rebuilding
MAA [85]	08	N	N	N	Y	N	Y	7	Not specified
IQRD [72]	08	N	Y	Y	Y	N	N	7	Not specified
LARD/RC [20]	09	Y	N	N	Y	Y	Y	7	Multiple TCP rebuilding
GCAP [20]	09	N	N	N	Y	Y	Y	7	Multiple TCP rebuilding
APRA [36]	09	N	Y	Y	Y	Y	Y	7	Not specified

Hence, we have included in Table 3, as the first column, the year of the publication of the paper that describes the distribution policy. The second, third and fourth columns of this table indicate whether the policy has the locality-awareness feature, is QoS-aware and/or includes an admission control mechanism, respectively. The fifth column details if dynamically generated Web content is especially treated by the distribution policy. These four characteristics are completely related to the distribution policy. From the sixth column till the ninth, we include details of the Web architecture proposed for each distribution policy. Therefore, the sixth column deals with HTTP/1.1 persistent connections and request granularity. The seventh and eighth indicate one- or two-way architecture, and whether the Web switch is content- or blind-aware, respectively. And finally the ninth column indicates the load balancing architecture proposed for the dispatching policy.

6.1 Locality-aware solutions

Several locality-aware solutions have been proposed as distribution policies in a content-aware load balancing design. The aim of these policies is to exploit the cache locality of the Web servers by sending the requests that ask for a determined Web page to a server that is likely to have it in its cache module. As can be observed in Table 3, most of the solutions that are in the literature between the years 1998–2001 improve the performance of the Web cluster by increasing the number of cache hits in the Web server nodes.

Locality-aware policies deal with a working set that normally consists of static Web pages. Thus, in order to obtain an increase in the performance by improving cache hit rates and if the working set is too large to fit in one server's cache, it is divided among the cluster nodes. This is the idea that Pai et al. introduced in their distribution strategy named *LARD* [60]. They define a set of nodes to serve a target file, and modify the set dynamically depending on the number of active connections of the nodes. The content-aware load balancer sends the request to the least loaded node of the set. They implement TCP Hand-off to send the connections from the front-end to the selected back-end node. An extension of LARD (*extLARD*) is introduced in [3] which permits LARD to deal with persistent connections. LARD is also used to evaluate a prototype implementation that includes a resource management facility for cluster based Web servers named *cluster reserves* [4]. This facility permits the performance isolation of the different server resources in order to effectively reserve them for different classes of service.

As an alternative to using the TCP Hand-off, which introduces an important overhead in the Web Switch [5], DNS infrastructure is used by Cherkasova et al. in [17, 18]. *FLEX* is defined in [17] as a two-way locality-aware solution that does not require any modification in the protocol nor special hardware support. It is mainly based on DNS infrastructure that allocates different sites to the nodes of the cluster. Based on the access rates and the size of the files, the working set is dynamically partitioned among the servers equally. Periodical monitoring permits detecting changes in the access pattern and repartition of the working set. In a later work [18], the authors propose *WARD*, a locality-aware distribution that defines a core of files that contains the most frequently accessed files. These files can be served by any node in the cluster while the rest of the working set is divided among all the nodes. In this case, Multiple TCP Hand-off is used when a request has to be sent from one server to another. The optimal size of the core is determined by an algorithm that considers the workload, the Random Access Memory (RAM) of the cluster and the overhead of the Hand-off operation and disk access.

There are other proposals that also include a set of the most accessed files to be cached in all the servers [19, 20, 32, 51, 62, 74]. A prefetching policy named *Time and Access Probability-based Prefetch (TAP^2)* is proposed by Park et al. in [62]. It predicts the next Web requests the clients are going to ask for by considering the costs of prefetching and the probability of the Web object to be requested, and if worth it, it prefetches the requested objects from local disks. A client session is assigned to a back-end server by a RR policy, and a TCP connection between them remains during the entire session. Sit et al. in [74] describe a distribution policy based on Socket Cloning and Multiple-Cloning that uses a L4/2 front-end, named *Cyclone*. They also select a set of most frequently requested files to be served, in this case, by a set of servers, stored in a hash table in each of the servers. The replication of the files is done progressively as the reference count increases. Hence, if a file reference count reaches a certain level, the file ends up replicated in all the node's caches. Faour and Mansour propose a content-aware distribution policy named *Weblins* in [32] and implement their proposal in Gobelins OS. A cooperative caching mechanism is used to replicate the most requested files in all the nodes of the cluster. They distribute the requests from a layer-4 switch to the servers by the RR algorithm. When a request needs to be transferred to another Web server, it is done by using the TCP Hand-off protocol.

In a recent work [51], Multiple TCP Rebuilding is implemented in LVS platform and named LVS-CAD. As LVS cannot perform content-aware distribution, a fast TCP module handshaking has been introduced in the IP-layer of the front-end in order to establish the connection with the client. Liu et al., in this work, describe three different distribution policies: *Content-Aware Weighted Least Load (CAWLL)*, *Extended Locality-Aware Request Distribution with Replication Policy (xLARD/R)* and *Content-Aware Hybrid Request Distribution (CAHRD)*. CAWLL is not a locality-aware policy, it selects the least loaded back-end server. xLARD/R is an extension of LARD that considers some additional costs in order to estimate the load of the back-end nodes when taking load balancing decisions. Finally, CAHRD mixes both CAWLL and xLARD/R. It switches to CAWLL when a dynamic request arrives, and to xLARD/R when the request is static. Results show that the locality-awareness is more suitable for static than dynamic requests as CAWLL, and hence, CAHRD, performs better than xLARD/R when more than 30% of the workload corresponds to dynamic Web content.

Based on WARD policy, Chiang et al. implement two distribution policies (*Content-based Workload-Aware Request Distribution with Core Replication (CWARD/CR)* and *Content-based Workload-Aware Request Distribution with Frequency-based Replication (CWARD/FR)*) [19] in a cluster platform named LVS with Content-Aware Dispatching and File Caching (LVS-CAD/FC). All the nodes prefetch the set of the most frequently accessed files and the less frequently accessed files are partitioned among the nodes in the CWARD/CR policy. While in the CWARD/FR policy, the replication of the files in the nodes is proportional to their access frequencies. In both cases, the requests are assigned to servers based on the RR policy among the servers that have the requested file replicated. Results show that CWARD/FR outperforms CWARD/CR. Chiang et al., in a later paper [20], propose two other content-aware policies based on CAP (described in the next subsection) and LARD, named *Grouped Client-Aware Policy (GCAP)* and *Locality-Aware Request Distribution with Replication and Classification (LARD/RC)*, respectively. In these new policies, the authors improve the treatment of dynamic Web pages, database processing and multimedia stream data, that might easily overload the back-end servers, especially in heterogeneous environments. The results show that GCAP and LARD/RC improve the performance of their predecessors, particularly under these circumstances.

Persistent connections are not considered in some of the previous proposals [17, 60, 62] because they mean an increase in cost when distributing the load based on locality, as the connection has to be migrated from server to server. The overhead effort of the TCP connections portability between Web servers is studied by Carrera and Bianchini in [13]. They propose a solution named *PRESS* that has two modes of operation. It starts serving requests with a content-blind distribution policy until the cache miss rate reaches a certain threshold, then the policy switches to a locality-aware mechanism as the TCP Hand-off cost is now justified. TCP Hand-off is used when a request has to be served by another server in the cluster. The authors conclude that portability should be promoted in fast-communication clusters as it has a low cost in terms of performance, but efficiency should be promoted in slow-communication cluster as in this case portability is very expensive. Tang and Chanson investigate how request- and session-grained allocations affect caching performance under persistent connections in [80]. They compare two algorithms: *Balanced Content-Based (BCB)* and *Session Affinity-Aware (SAA)*. The authors conclude that the locality-aware performance benefits of request-grained content-aware distribution policies are not easily extended to session-grained allocations.

Ciardo et al. in [22] develop a distribution policy named *EQUILOAD* based on the sizes of the requested documents. They partition the sizes of the working set into some subsets (the same as number of servers in the cluster). EQUILOAD is modified in [66] in order to avoid the need of a priori knowledge of the working set size distribution. The modified solution, named ADAPTLOAD, dynamically sets empirically-based fuzzy boundaries to the intervals of the response sizes. Despite EQUILOAD and ADAPTLOAD not being designed as locality-aware mechanisms, they also obtain the benefits of caching as they always direct requests for the same document to the same server. EQUILOAD and ADAPTLOAD only consider static Web traffic. In a later work [88], ADAPTLOAD is tested with a workload that also includes dynamically generated Web content. As it is not possible to know the size of the dynamic content requested, the authors distribute dynamic requests based on the Join Shortest Queue (JSQ) policy. We have included this version as ADAPTLOAD v2 in Table 3.

Also considering dynamic traffic, *Harvard Array of Clustered Computers (HACC)* is a request distribution developed by Zhang et al. in [91] that dynamically divides the entire working set among the number of nodes in the cluster. The load balancing scheme of this proposal is based on two-way architecture that does not take into account persistent connections.

Most of the distribution policies referenced in this subsection do not especially treat dynamic Web content [3, 13, 17–19, 22, 32, 60, 62, 66, 74, 80], as shown in Table 3. This is due to the unpredictability of service times of generating dynamic Web pages. In the next subsection, dynamic Web traffic is considered in the design of most of the proposals.

6.2 Non locality-aware solutions

Since 2001, more non locality-aware solutions appear in the literature. They do not consider cache hits in the performance evaluation of the Web system. Therefore, other factors are included in these policies like, for instance, the evaluation of the resource utilisation by the different types of requests [14, 36, 72, 85], the introduction of QoS in the service provided [36, 48, 59, 71, 72], or the inclusion of an admission control in the load balancing solution [36, 72].

Considering only the content of the incoming request, Casalicchio and Colajanni propose a policy named *CAP* in [14] that obtains good performance results when serving dynamic and secure Web content. As the requests are classified depending on the expected impact they will have on the server resources, CAP takes distributing decisions depending on the service type required by them. The state of the servers is not considered in this solution. All the servers of the cluster provide all the service types considered.

Considering the content of incoming requests and obtaining some metrics from the Web switch that permit estimating the load of the Web servers, Yao et al. in [85] propose *Message-Aware Adaptive (MAA)*. It is a scheduling policy that is developed to be executed in the Application Oriented Networking (AON) product of Cisco Systems [24]. This policy distinguishes among types of messages and balances the load based on their resource consumption. Hence, when a request arrives to the system, an estimation of the completion time is calculated and the server that obtains the earliest is chosen to serve the request.

There are other proposals that do include monitored information obtained from the Web servers of the cluster to take distribution decisions. This is the case of *Fuzzy Adaptive Request Distribution (FARD)* [8]. This work of Borzemski and Zatwarnicki describes a content-aware load balancing mechanism that estimates the response time of each request in every server of the cluster using a fuzzy estimation mechanism. This estimation is based on some metrics obtained from the server, such as the Central Processing Unit (CPU), the disk and the communication link load. FARD selects the server with the lowest estimated response time in order to attend the request. The authors compare their proposal to LARD and WRR in a prototype, concluding that FARD improves their performance, especially in heterogeneous environments. Another work that also obtains performance information by monitoring the applications running on the nodes is *Adaptive Load Balancing Mechanism (ALBM)* [21]. Choi describes in this paper a load balancing mechanism that provides scalable services in a multi-cluster system. The front-end of the system is a layer-4 Web switch that balances the load by using DR and NAT, and then the content-aware distribution is done from the nodes of the cluster. ALBM is compared to LVS using RR, LC and WLC scheduling algorithms.

Monitored information as service times or throughput values in Web and back-end servers, or the arrival rate to the Web system is considered in another content-aware proposal [36]. In this case, the algorithm can enable or disable servers from the cluster depending on these monitored parameters. Instead of monitoring periodically, this algorithm follows an adaptive time slot scheduling that depends on the arrival rate to set the monitoring intervals [35]. Hence, when the system is receiving a high demand of Web requests, the intervals between monitoring times are reduced and viceversa when the system is idle. This method permits reducing the overhead of the algorithm when the system is not stressed.

When making decisions based on information obtained from the servers, it is very important that the information is updated. Some authors [28, 68, 72] express their concern about the accuracy of the information obtained as it might not be very current in the moment the decision is made. Worried about the problem of load balancing with stale information, Satake and Inai propose a scheduling method named *Non Probabilistic Server Selection Method (NPSSM)* that obtains the information periodically from the Web servers of the cluster in [68]. Based on this information, the authors try to compensate the differences in load among the nodes and once the load is balanced, they apply the RR policy for the next requests. By simulation, the authors conclude that their method is scalable with the number of servers and that it is immune to the impact of load information acquisition intervals.

Dynamic traffic is especially considered in [59, 87]. Zhang et al. in [87] propose an enhancement to the ADAPTLOAD policy. As the ADAPTLOAD policy described in [88] did not include an original treatment for dynamic requests, the authors study now how the autocorrelation in the arrival process affects the performance of the load balancing policies and propose a load unbalancing mechanism that tries to reduce the autocorrelation of the requested file sizes. They named their policy *D_EQAL*. In this case, the aim for being locality aware is less clear than in ADAPTLOAD, this is the reason to include D_EQAL as a non locality-aware solution. Ok and Park in [59] describe an algorithm that focuses on dynamically generated Web content (*Around k-Bounded*). As it is not possible to know the load that the execution of the scripts involves in the Web server in advance, they propose an algorithm running in a layer-4 Web switch that monitors the load of the servers. If the servers are attending the same number of requests then the Web switch balances the requests following a RR fashion. The switch sends the next request to the least loaded server in the cluster when the load values of the servers are not equal. Persistent connections when using HTTP/1.1 are considered, hence in case the target server for a request is other than the one that is connected to the client, the TCP Hand-off protocol is used to migrate the connection to the appropriate server. The authors show the benefits of their proposal in terms of scalability and QoS guarantees.

6.3 QoS-aware solutions

QoS requirements are included in some other proposals that also obtain monitored information from the Web server [36, 48, 59, 71, 72]. Considering Stochastic High-Level Petri Net (SHLPN) modelling, Shan et al. develop a QoS-aware load balancing strategy named *Extended Fewest Server Processes First (E-FSPF)* in [71]. They combine a process scheduling policy in the Web servers that considers the priority of the requests, with a load balancing mechanism in the front-end of the system that sends the request to the Web server with the fewest number of processes with higher or equal priorities. Li et al. describe in [48] a Web distribution system named *Gage* that balances the load among a set of Web servers and supports QoS. They use a front-end that splices the TCP connection with the chosen server in one-way architecture. The resource consumption is considered as the QoS metric and the Service Level Agreement (SLA) is guaranteed by the allocation of multiple system resources. The requests are scheduled following a WRR algorithm.

When including QoS requirements in a load balancing mechanism, often admission control policies are also introduced in order to avoid a sudden collapse of the servers due to an increase in the demand [36, 72]. The content-aware load balancing algorithm with QoS-aware admission control proposed by Shafirian et al. is named Intelligent Queue-based Request Dispatcher (IQRD) [72]. The authors monitor the Web servers in order to dynamically compute the remaining capacity of the Web system. The requests are classified depending on the resources they consume in the server nodes and are assigned to the nodes based on their load status. The authors compare their strategy to CAP and WRR and conclude that, despite IQRD introducing more overhead, it improves the performance of the system in terms of response time and throughput.

7 Conclusions and open problems

This article sums up the load balancing mechanisms that have been developed and classifies them by differentiating the OSI protocol stack layer the load balancing is based on. Content-blind load balancing mechanisms have been widely developed in the literature and have also been considered to be included in commercial products (some of which have already been withdrawn from the market). The most important content-blind load balancing mechanisms are DR as a one-way layer-2 forwarding solution and NAT as a two-way layer-3 forwarding solution.

As the content-aware load balancing solutions become more popular, mainly because they make differentiation in the workload possible and hence, a more accurate distribution of the requests can be performed, more effort is dedicated to avoid their drawbacks. The possibility that the layer-7 Web switch becomes the bottleneck of the system, even in one-way architecture, is the main drawback of using a content-aware front-end in the Web system. This problem can be solved by replacing it with a content-blind front-end and transferring the distribution task to other nodes of the system. One of the most popular proposals of one-way architecture is TCP Hand-off. It has been recently included in several studies that try to reduce the overhead produced by a layer-7 font-end. Therefore, a layer-4 Web switch is used instead and in case the TCP connection needs to be transferred between two servers, the original TCP Hand-off mechanism is used. Other one-way proposals, like Socket Cloning and One-packet TCP State Migration to packet filter, already introduced a layer-4 front-end in their original design.

Another important problem related to content-aware load balancing architectures is the difficulty to get request granularity when using HTTP1.1 protocol. This particular problem is solved individually for each of the one-way and two-way architecture proposals.

Considering content-aware distributing proposals, locality-aware policies were extensively investigated during the beginning years of the first decade of 2000 with the aim of exploiting the cache performance benefits in the Web servers. Most of these algorithms normally consider static content in the Web pages, as the fact of considering dynamic Web pages in the workload complicates the load balancing because service times of the scripts that generate the dynamic content are not easily predictable. However, Web workload has become more dynamic in the last few years, hence, some interesting locality-aware distributing policies have been proposed recently that also treat these kinds of traffic.

Focusing on non locality-aware solutions, we have discussed some proposals that monitor some performance metrics in the server nodes of the system in order to compute an estimation of future performance and distribute the requests accordingly. However, the cost of generating some kinds of Web pages, as a dynamic Web page that needs to access a database server, is more expensive in terms of performance than the cost of serving static Web pages and, hence, it is difficult to measure or predict. This type of predictions probably needs more research. The “freshness” of the information obtained from the Web servers that the algorithm uses to take the load balancing decisions is also a problem that needs more research, as most of the proposals described here do not consider the possibility that the load information is stale.

Some content-aware distributing proposals include QoS in their requirements and consequently, some admission control mechanisms are related to them. However, only two proposals include admission control mechanisms in a content-aware platform. Probably more research has to be done in this field too.

There are also some considerations about possible future research that involve Virtual Servers, as they are may improve the performance of the load balancing technique covered in this survey. Virtualisation permits to run specific servers in the same physical node, and can be used to differentiate the services provided by the Web system. We also consider an important issue for future study energy-efficiency metrics in the development of Web load balancing in order to provide a "greener" computing solution.

References

1. Andreolini, M., Colajanni, M., Nuccio, M.: Kernel-based web switches providing content-aware routing. In: Proc. of the 2nd IEEE International Symposium on Network Computing and Applications (NCA'03) (2003)
2. Apostolopoulos, G., Aubespin, D., Peris, V.G.J., Pradhan, P., Saha, D.: Design, implementation and performance of a content-based switch. In: Proc. of INFOCOM (2000)
3. Aron, M., Druschel, P., Zwaenepoel, W.: Efficient support for P-HTTP in cluster-based web servers. In: Proc. of the Annual Conference on USENIX Annual Technical Conference (1999)
4. Aron, M., Druschel, P., Zwaenepoel, W.: Cluster reserves: a mechanism for resource management in cluster-based network servers. In: Proc. of ACM SIGMETRICS (2000)
5. Aron, M., Sanders, D., Druschel, P., Zwaenepoel, W.: Scalable content-aware request distribution in cluster-based network servers. In: Proc. of the USENIX 2000 Annual Technical Conference (2000)
6. Barroso, L., Dean, J., Hoelzle, U.: Web search for a planet: the google cluster architecture. IEEE Micro **23**, 22–28 (2003)
7. Bent, L., Rabinovich, M., Voelker, G.M., Xiao, Z.: Characterization of a large web site population with implications for content delivery. In: Proc. of the 13th International Conference on World Wide Web (2004)
8. Borzemski, L., Zatwarnicki, K.: A fuzzy adaptive request distribution algorithm for cluster-based web systems. In: Proc. of the 11th Euromicro Conference on Parallel, Distributed and Network-Based Processing (Euro PDP) (2003)
9. Brendel, J.: Client-side resource-based load-balancing with delayed-resource-binding using TCP state migration to WWW server farm. United States Patent 6,182,139. Resonate Inc (2001)
10. Brisco, T.P.: DNS support for Load Balancing. RFC 1794 (1995)
11. Cardellini, V., Casalicchio, E., Colajanni, M., Yu, P.S.: The state of the art in locally distributed web-server systems. ACM Comput. Surv. **34**, 263–311 (2002). doi:10.1145/508352.508355
12. Cardellini, V., Colajanni, M., Yu, P.S.: Dynamic load balancing on web-server systems. IEEE Int. Comp. **3**(3), 28–39 (1999)
13. Carrera, E., Bianchini, R.: Efficiency vs. portability in cluster-based network servers (2001)
14. Casalicchio, E., Colajanni, M.: A client-aware dispatching algorithm for web clusters providing multiple services. In: Proc. of the 10th International Conference on World Wide Web (2001)
15. Cavale, M.R.: Introducing Microsoft Cluster Service (MSCS) in the Windows Server 2003 Family. Microsoft Corporation (2002)
16. Chang, Y.K., Cheng, W.H., Young, C.P.: Fully pre-splicing TCP for web switches. In: Proc. of the 1st International Conference on Innovative Computing, Information and Control (ICICIC) (2006)

17. Cherkasova, L., DeSouza, M., Ponnekanti, S.: Performance analysis of “content-aware” load balancing strategy FLEX: two case studies. In: Proc. of the 34th Hawaii International Conference on System Sciences (2001)
18. Cherkasova, L., Karlsson, M.: Scalable web server cluster design with workload-aware request distribution strategy WARD. In: Proc. of the Third International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS) (2001)
19. Chiang, M.L., Lin, Y.C., Guo, L.F.: Design and implementation of an efficient web cluster with content-based request distribution and file caching. J. Syst. Softw. **81**, 2044–2058 (2008)
20. Chiang, M.L., Wu, C.H., Liao, Y.J., Chen, Y.F.: New content-aware request distribution policies in web clusters providing multiple services. In: Proc. of the ACM Symposium on Applied Computing (2009)
21. Choi, E.: Performance test and analysis for an adaptive load balancing mechanism on distributed server cluster systems. Future Gener. Comput. Syst. **20**, 237–247 (2004)
22. Ciardo, G., Riska, A., Smirni, E.: EQUILOAD: a load balancing policy for clustered web servers. Perform. Eval. **46**(2–3), 101–124 (2001)
23. Cisco Systems, I.: Scalable Content Switching. A discussion of the cisco css 11500 series content services switch architecture. White Paper (2002)
24. Cisco systems, inc. http://www.cisco.com/ (2010). Accessed 26 Nov 2010
25. Cohen, A., Rangarajan, S., Slye, H.: On the performance of TCP splicing for URL-aware redirection. In: Proc. of the 2nd Conference on USENIX Symposium on Internet Technologies and Systems (1999)
26. Colajanni, M., Yu, P.S.: A performance study of robust load sharing strategies for distributed heterogeneous web server systems. IEEE Trans. Knowl. Data Eng. **14**(2), 398–414 (2002)
27. Colby, S., Krawezyk, J.J., Nair, R.K., Royee, K., Siegel, K.P., Stevens, R.C., Wasson, S.: Method and System for Directing a Flow Between a Client and a Server. United States Patent 6,006,264 (2001). Arrowpoint Communications, Inc
28. Dahlin, M.: Interpreting stale load information. IEEE Trans. Parallel Distrib. Syst. **11**(10), 1033–1047 (2000)
29. Damani, O.P., Chung, E., Huang, Y., Kintala, C., Wang, Y.M.: ONE-IP: techniques for hosting a service on a cluster of machines. Comput. Netw. ISDN Syst. **29**, 1019–1027 (1997)
30. (DARPA), D.A.R.P.A.: Transmission Control Protocol. RFC 793 (1981)
31. F5 Networks, Inc.: http://www.f5.com/ (2010)
32. Faour, A., Mansour, N.: Weblins: A scalable www cluster-based server. Adv. Eng. Softw. **37**, 11–19 (2006)
33. Fielding, R.T., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P.J., Berners-Lee, T.: Hypertext transfer protocol—HTTP/1.1. RFC **2616** (1999)
34. Gan, X., Schroeder, T., Goddard, S., Ramamurthy, B.: Highly available and scalable cluster-based web servers. In: Proc. of the 8th IEEE International Conference on Computer Communications and Networks (1999)
35. Gilly, K., Alcaraz, S., Juiz, C., Puigjaner, R.: Analysis of burstiness monitoring and detection in an adaptive web system. Comput. Networks **53**, 668–679 (2009)
36. Gilly, K., Juiz, C., Alcaraz, S., Puigjaner, R.: Adaptive admission control algorithm in a QoS-aware web system. In: Proc. of IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS) (2009)
37. Goddard, S., Schroeder, T.: The SASHA architecture for network-clustered web servers. In: Proc. of the 6th IEEE International Symposium on High Assurance Systems Engineering (2001)
38. Hunt, G., Nahum, E., Tracey, J.: Enabling Content-based Load Distribution for Scalable Services. Tech. rep., IBM T.J. Watson Research Center (1997)
39. IBM: http://www.ibm.com/ (2010)
40. IBM: Application Switching with Nortel Networks Layer 2–7 gigabit Ethernet Switch Module for IBM Bladecenter. IBM Redbook (2006)
41. Iyengar, A., Challenger, J., Dias, D., Dantzig, P.: High-performance web site design techniques. IEEE Int. Comp. **4**, 17–26 (2000)
42. Kachris, C., Vassiliadis, S.: Design of a web switch in a reconfigurable platform. In: Proc. of the 2006 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (2006)
43. Kerdlapanan, D., Khunkitti, A.: Content-based load balancing with multicast and tcp-handoff. In: Proc. of International Symposium on Circuits and Systems (2003)

44. Kobayashi, M., Murase, T.: Asymmetric tcp splicing for content-based switches. In: Proc. of IEEE International Conference on Communications (ICC) (2002)
45. Kokku, R., Rajamony, R., Harrick Vin, L.A.: Half-pipe anchoring: an efficient technique for multiple connection handoff. In: Proc. of the 10th IEEE International Conference on Network Protocols (2002)
46. Kopparapu, C.: Load Balancing Servers, Firewalls and Caches. Wiley (2001)
47. Krishnamurthy, B., Wills, C., Zhang, Y.: On the use and performance of content distribution networks. In: Proc. of the 1st ACM SIGCOMM Workshop on Internet Measurement (2001)
48. Li, C., Peng, G., Gopalan, K., cker Chiueh, T.: Performance guarantee for cluster-based internet services. In: Proc. of the 23rd International Conference on Distributed Computing Systems (2003)
49. Lin, Y.D., Tsai, P.T., Lin, P.C., Tien, C.M.: Direct web switch routing with state migration, TCP masquerade, and cookie name rewriting. In: Proc. of Global Telecommunications Conference (2003)
50. Liu, H.H., Chiang, M.L.: Tcp rebuilding for content-aware request dispatching in web clusters. Journal of Internet Technology **6**, 231–240 (2005)
51. Liu, H.H., Chiang, M.L., Wu, M.C.: Efficient support for content-aware request distribution and persistent connection in Web clusters. Softw. Pract. Exp. **37**, 1215–1241 (2007)
52. Luo, M.Y., Yang, C.S.: System support for scalable, reliable and highly manageable web hosting service. In: Proc. of the 3rd conference on USENIX Symposium on Internet Technologies and Systems (2001)
53. Luo, M.Y., Yang, C.S., Tseng, C.W.: Analysis and improvement of content-aware routing mechanisms. IEICE Trans. Commun. **E88**, 227–238 (2005)
54. Maltz, D.A., Bhagwat, P.: TCP Splicing for Application Layer Proxy Performance. Tech. Rep., IBM (1998)
55. Marwah, M., Mishra, S., Fetzer, C.: Fault-tolerant and scalable TCP splice and web server architecture. In: Proc. of the 25th IEEE Symposium on Reliable Distributed Systems (2006)
56. Mogul, J.C.: The case for persistent-connection HTTP. In: Proc. of SIGCOMM (1995)
57. Networks, F.: http://www.foundrynet.com (2010)
58. Networks, N.: http://www.nortel.com/ (2010)
59. Ok, M., Park, M.S.: Distributing requests by (around k)-bounded load-balancing in web server cluster with high scalability. IEICE Trans. Inf. Sys. **E89-D**, 663–672 (2006)
60. Pai, V.S., Aron, M., Banga, G., Svendsen, M., Druschel, P., Zwaenepoel, W., Nahum, E.M.: Locality-aware request distribution in cluster-based network servers. In: Proc. of Architectural Support for Programming Languages and Operating Systems (ASPLOS) (1998)
61. Papathanasiou, A.E., Hensbergen, E.V.: KNITS: switch-based connection hand-off. In: Proc. of IEEE INFOCOM (2002)
62. Park, S.Y., Park, D., Lee, J., Cho, J.W.: Efficient inter-backend prefetch algorithms in cluster-based web servers. In: Proc. of International Conference/Exhibition on High Performance Computing (2001)
63. Radware: http://www.radware.com (2010)
64. Resonate: Resonate central dispatch technology advantage: TCP connection HOP. White Paper (2001)
65. Resonate, inc. http://www.resonate.com/ (2010)
66. Riska, A., Sun, W., Smirni, E., Ciardo, G.: ADAPTLOAD: effective balancing in clustered web servers under transient load conditions. In: Proc. of the 22nd International Conference on Distributed Computing Systems (2002)
67. Rosu, M.C., Rosu, D.: An evaluation of TCP splice benefits in web proxy servers. In: Proc. of WWW (2002)
68. Satake, S., Inai, H.: Special issue on internet architecture technology papers: a nonprobabilistic server selection method based on periodically obtained load information for web server clusters. Electron. Commun. Jpn. **89**, 1–12 (2006)
69. Schroeder, T., Goddard, S., Ramamurthy, B.: Scalable web server ciustering technologies. IEEE Netw. **May**, 38–46 (2000)
70. Server, L.V.: http://www.linuxvirtualserver.org/ (2006). Accessed 26 Nov 2010
71. Shan, Z., Lin, C., Marinescu, D.C., Yang., Y.: Modeling and performance analysis of QoS-aware load balancing of web-server clusters. Comput. Networks **40**, 235–256 (2002)
72. Sharifian, S., Motamedi, S.A., Akbarib, M.K.: A content-based load balancing algorithm with admission control for cluster web servers. Future Gener. Comput. Syst. **24**, 775–787 (2008)

73. Sit, Y.F., Wang, C.L., Lau, F.: Socket cloning for cluster-based web servers. In: Proc. of IEEE International Conference on Cluster Computing (2002)
74. Sit, Y.F., Wang, C.L., Lau, F.: Cyclone: a high-performance cluster-based web server with socket cloning. Cluster Comput. **7**, 21–37 (2004)
75. Snoeren, A.C., Andersen, D.G., Balakrishnan, H.: Fine-grained failover using connection migration. In: Proc. of 3rd USENIX Symp. on Internet Technologies and Systems (2001)
76. Switching, L.L.: http://www.linuxvirtualserver.org/software/ktcpvs/ktcpvs.html (2010)
77. Syme, M., Goldie, P.: Optimizing Network Performance with Content Switching. Server, Firewall and Cache Load Balancing. Prentice Hall (2004)
78. Takahashi, M., Kohiga, A., Sugawara, T., Tanaka, A.: Tcp-migration with application-layer dispatching: a new http request distribution architecture in locally distributed web server systems. In: Proc. of the 1st International Conference on Communication System Software and Middleware (2006)
79. Tang, W., Cherkasova, L., Russell, L., Mutka, M.W.: Modular tcp handoff design in streams-based tcp/ip implementation. In: Proc. of the 1st International Conference on Networking-Part 2 (2001)
80. Tang, X., Chanson, S.T.: On caching effectiveness of web clusters under persistent connections. J. Parallel Distrib. Comput. **63**, 981–995 (2003)
81. TCPHA project. http://dragon.linux-vs.org/~dragonfly/htm/tcpha.htm (2004). Accessed 26 Nov 2010
82. Teo, Y.M., Ayani, R.: Comparison of load balancing strategies on cluster-based web servers. Trans. of the Soc. for Model. and Sim. **77**, 185–195 (2001)
83. Yang, C.S., Luo, M.Y.: Efficient support for content-based routing in web server clusters. In: Proc. of the 2nd Conference on USENIX Symposium on Internet Technologies and Systems, vol. 2 (1999)
84. Yang, J., Jin, D., Li, Y., Hielscher, K.S., German, R.: Modeling and simulation of performance analysis for a cluster-based web server. Simulation Modelling Practice and Theory **14**, 188–200 (2006)
85. Yao, J., Ding, J.J., Bhuyan, L.N.: Intelligent message scheduling in application oriented networking systems. In: Proc. of IEEE International Conference on Communications (ICC) (2008)
86. Zeng-Kai, D., Jiu-Bin, J.: A completely distributed architecture for cluster-based web servers. In: Proc. of the 4th International Conference on Parallel and Distributed Computing, Applications and Technologies (2003)
87. Zhang, Q., Mi, N., Riska, A., Smirni, E.: Load unbalancing to improve performance under autocorrelated traffic. In: Proc. of the 26th IEEE International Conference on Distributed Computing Systems (2006)
88. Zhang, Q., Riska, A., Sun, W., Smirni, E., Ciardo, G.: Workload-aware load balancing for clustered web servers. IEEE Trans. Parallel Distrib. Syst. **3**, 219–233 (2005)
89. Zhang, R., Abdelzaher, T.F., Stankovic, J.A.: Efficient TCP connection failover in web server clusters. In: Proc. of IEEE INFOCOM (2004)
90. Zhang, W.: Linux virtual server for scalable network services. In: Proc. of OTTAWA Linux Symposium (2000)
91. Zhang, X., Barrientos, M., Chen, J.B., Seltzer, M.: HACC: an architecture for cluster-based web servers. In: Proc. of the 3rd USENIX Windows NT Symposium (1999)
92. Zhao, L., Luo, Y., Bhuyan, L., Iyer, R.: Design and implementation of a content-aware switch using a network processor. In: Proc. of the 13th Symposium on High Performance Interconnects (2005)