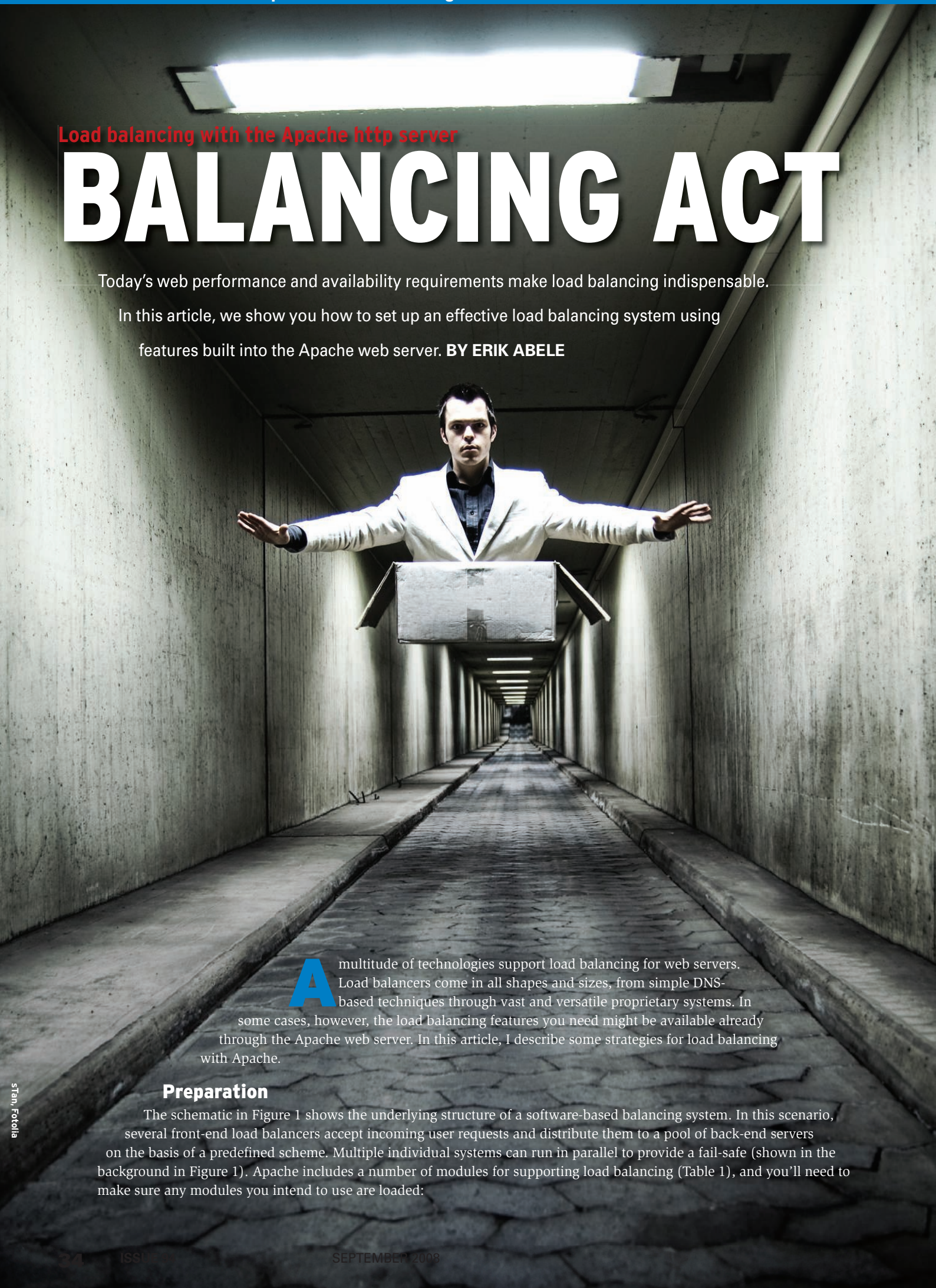
Load balancing with the Apache http server

BALANCING ACT

Today's web performance and availability requirements make load balancing indispensable. In this article, we show you how to set up an effective load balancing system using features built into the Apache web server. **BY ERIK ABELE**



sTan, Fotolia

A multitude of technologies support load balancing for web servers. Load balancers come in all shapes and sizes, from simple DNS-based techniques through vast and versatile proprietary systems. In some cases, however, the load balancing features you need might be available already through the Apache web server. In this article, I describe some strategies for load balancing with Apache.

Preparation

The schematic in Figure 1 shows the underlying structure of a software-based balancing system. In this scenario, several front-end load balancers accept incoming user requests and distribute them to a pool of back-end servers on the basis of a predefined scheme. Multiple individual systems can run in parallel to provide a fail-safe (shown in the background in Figure 1). Apache includes a number of modules for supporting load balancing (Table 1), and you'll need to make sure any modules you intend to use are loaded:

```
LoadModule xyz_module ➥
modules/mod_xyz.so
```

As you can see in Table 1, Apache's basic load balancing capabilities include features such as caching, compression, URL rewriting, and header processing. Some of the modules in Table 1 are loaded by default.

Consult your own Apache configuration for more on which modules might already be present on your system.

If JServ-capable application servers, such as Apache Tomcat or Jetty, are used, the gateway can also use the Apache JServ Protocol (ajp). All you need to do is load the *mod_proxy_ajp* module instead of *mod_proxy_http* and change the URLs from *http://* to *ajp://*.

The use of this binary-format protocol offers a couple of advantages with regard to back-end connection performance and lower resource overheads, but this functionality is bought at the price of more permanent connections to the back ends.

Incidentally, you can run ajp and http back ends at the same time as members of the same pool. For the sake of completeness, keep in mind also that Apache supports ftp proxy with the *mod_proxy_ftp* module.

For additional details, check out the Apache http server documentation [2].

Configuration

The sample configuration shown in Listing 1 includes the basic front-end server settings for load balancing in Apache.

This configuration starts with *ProxyRequests* to disable the normal proxy mode and setting up what is known as a reverse proxy, or *gateway*, to be more precise. Disabling Via headers (*ProxyVia*) makes the gateway invisible.

The *ProxyPreserveHost* and *ProxyErrorOverride* commands ensure that the host headers included in the request are passed on to the back ends and that any error messages generated by the back ends are replaced by the load balancer and thus standardized. The output of a suitable timeout, with *ProxyTimeout*, rounds off the basic configuration.

The core definition, that of a back-end pool and its members, is handled by a *Proxy* container and the specification of a special *balancer://* schema followed by the pool name. The *BalancerMember* instructions and parameters in the container specify the individual members along with their properties.

At the end of the configuration, the back-end pool defined previously is assigned a separate URL space; more parameters define the load balancer's generic approach. To enable regular expressions, you could use the advanced *ProxyPassMatch* command instead of *ProxyPass*.

As an alternative, the rewrite module (*mod_rewrite*) and custom rules would unleash the full power of regular expressions. However, in this case, you will need to use *ProxySet* because load balancer parameters cannot be modified by rules:

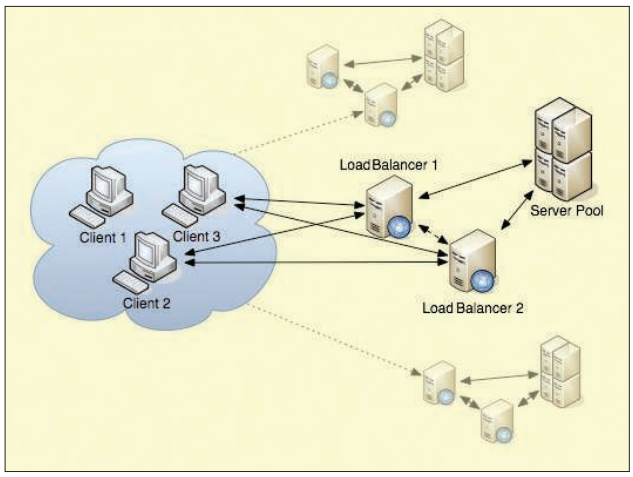


Figure 1: Schematic overview of a typical http-based balancing system.

```
ProxySet balancer://pool1 ➥
  lbmethod=bytraffic
...
RewriteEngine On
RewriteRule ^/+(.*)$➥
  balancer://pool1/$1 [P,L]
```

Listing 1 thus defines two back-end servers for *pool1*. Requests are distributed on the basis of the number of requests (see the *lbmethod* parameter). The load factor setting assigns twice as many requests to *server1* compared with *server2*. Connections are reused but also restricted to a maximum value. The URL space is defined as the complete URL space below */shop*.

Table 2 provides a summary of the most common *ProxyPass* and *BalancerMember* commands. For more information, see the Apache http server documentation [2].

Options

The Apache http server's proxy module (*mod_proxy*) provides an unbelievable range of special settings. Tools are available for many different scenarios. For

Listing 1: Sample Configuration

```
01 ProxyRequests Off
02 ProxyVia Off
03
04 ProxyPreserveHost On
05 ProxyErrorOverride On
06
07 ProxyTimeout 30
08
09 <Proxy balancer://pool>
10   BalancerMember http://
   server1:8080 \
11     min=10 max=50 loadfactor=2
12
13   BalancerMember http://
   server2:8080 \
14     min=5 max=25 loadfactor=1
15 </Proxy>
16
17 ProxyPass /shop balancer://
   pool1 \
18   lbmethod=byrequests \
19   nofailover=Off maxattempts=3
   \
20   stickysession=PHPSESSIONID
```

Listing 2: mod_cache Sample Configuration

```
01 CacheEnable disk /
02 CacheDisable /users
03 CacheRoot /var/cache/httpd
04 ...
05 AddOutputFilterByType DEFLATE
   text/html
```

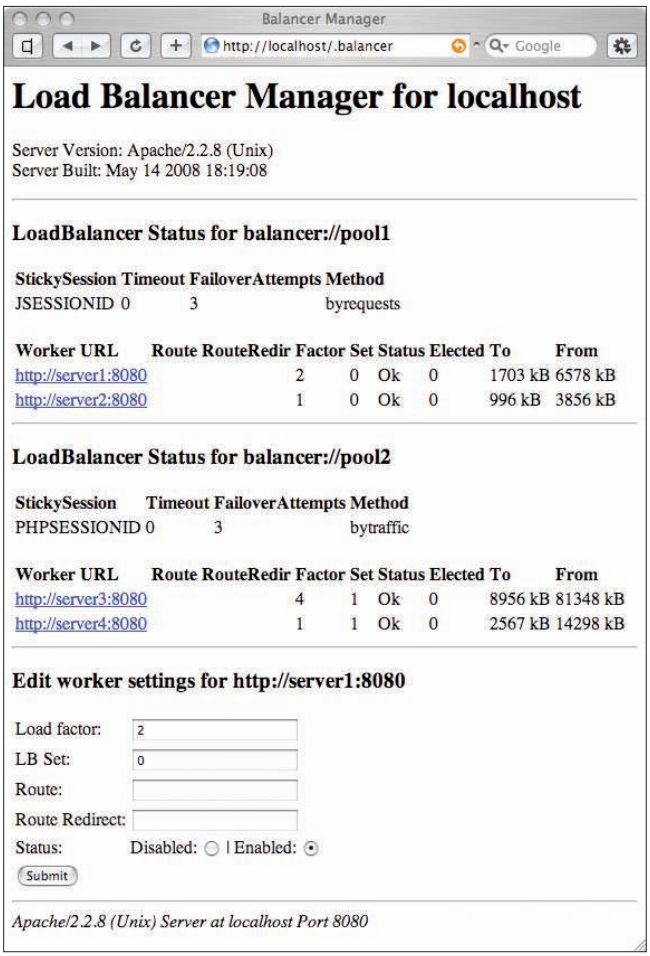


Figure 2: The balancer manager web interface supports basic load balancer management.

example, you can use the *status* parameter to operate a hot standby server:

```
BalancerMember ➥
http://server4:1080... status=+H
```

This command specifies that *server4* is only enabled if all the remaining pool members fail.

This server is the last line of defense and can be used to serve up a restricted version of a web application or to forward requests to a substitute system.

Sticky Sessions

Another typical configuration is used to support sticky sessions:

```
BalancerMember ➥
http://server6:➥
1080... ➥
stickysession=➥
JSESSIONID
```

The *stickysession* parameter, combined with the name of a cookie supported by the back ends, means that requests originating with individual users are always sent to the same back-end server.

This kind of limited distribution ensures the persistence of the requests, but it does interfere with the actual task of load balancing.

To forward information to the back-end servers in a targeted way, or to influence communications with the the back end, the proxy module also supports custom environmental variables and http headers, which you can use to restrict the connections to the back end or to advertise the use of SSL:

```
SetEnv proxy-nokeepalive 1
...
RequestHeader ➥
set Front-End-Https "On"
```

A number of standard variables and headers, such as *proxy-nokeepalive*, *proxy-sendcl*, *X-Forwarded-For*, or *X-Forwarded-Server*, saves typing and makes life easier for administrators.

Other modules support caching or filtering of content generated by the back ends. In addition to improving performance, caching also reduces the overall traffic volume and generally offloads some of the work from the back-end servers. Listing 2 enables a simple, file-based cache, including compression, for the whole URL space /. (Just to demonstrate how the exclusion feature works, the whole URL space below */users* has been excluded.)

Administration

If you loaded the status module (*mod_status*) when you launched the server, the proxy module also provides a simple but practical web interface (Figure 2). The simple configuration involves assigning a handler:

```
<Location "/.balancer-manager">
  SetHandler balancer-manager
</Location>
...
ProxyPass /.balancer-manager !
```

However, it is important to take access control into consideration and to exclude

Table 1: Required Modules

Module	Function
mod_proxy	Generic proxy module
mod_proxy_balancer	Balancer functions for the proxy module
mod_proxy_http	Http support for the proxy module
mod_cache	Generic caching module
mod_disk_cache	File-based cache for the caching module
mod_deflate	Content compression module
mod_rewrite	Module for parsing and processing URLs
mod_headers	Module for parsing and processing http headers

INFO

[1] Hypertext transfer protocol 1.1: *http://www.w3.org/Protocols/rfc2616/rfc2616.html*

[2] Apache documentation, httpd 2.2: *http://httpd.apache.org/docs/2.2/en/*

[3] Apache Software Foundation: *http://www.apache.org/*

THE AUTHOR

Erik Abele has worked for many years as a freelance IT consultant. His international projects cover a full range of architectures and large web farm operations. Erik is a long-standing member of the Apache Software Foundation, where he takes an active part in the http Server and HttpComponents projects. You can contact Erik via his websites: *http://www.eatc.de/* or *http://www.codefaktor.de/*.

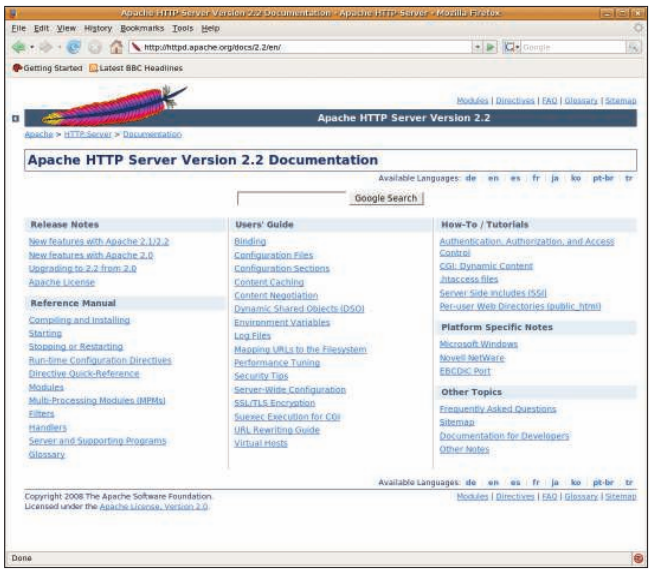


Figure 2: See the Apache Foundation website for more resources.

processing individual URLs within the load balancer, which is achieved by means of a negative *ProxyPass* command. If you use the Apache rewrite module (*mod_rewrite*), you can define a separate rule to handle this case.

Management Restrictions

Management is restricted to viewing the status of all configured balancers, disabling individual pool members, or modifying some basic settings, but it is extremely useful if you encounter a problem, or if you wish to monitor multiple load balancers.

Conclusions

Version 2.2 of the Apache http server offers a trouble-free, efficient, elegant, and scalable approach to load balancing in an http environment. Availability, a short learning curve, and nearly infinite flexibility all speak in favor of Apache. All told, the Apache load balancing system is a very sensible alternative to popular commercial or open source alternatives. ■

Table 2: Common Balancer Parameters

Name	Explanation
`status`	Balancer member status
`loadfactor`	Normalized balancer member weighting
`lbset`	The cluster set assigned to the balancer member
`lbmethod`	The request distribution method used by the balancer on the basis of either the number of requests (*byrequests*) or the traffic volume (*bytraffic*)
`min`	Minimum number of permanent back-end connections
`max`	Maximum number of permanent back-end connections
`maxattempts`	Maximum number of retries before denying a request
`stickysession`	Name of a persistent cookie used by the back-end server