Problem 4: Load Balancing for Web Servers Using PyDirector

Chris Baumler

Description:

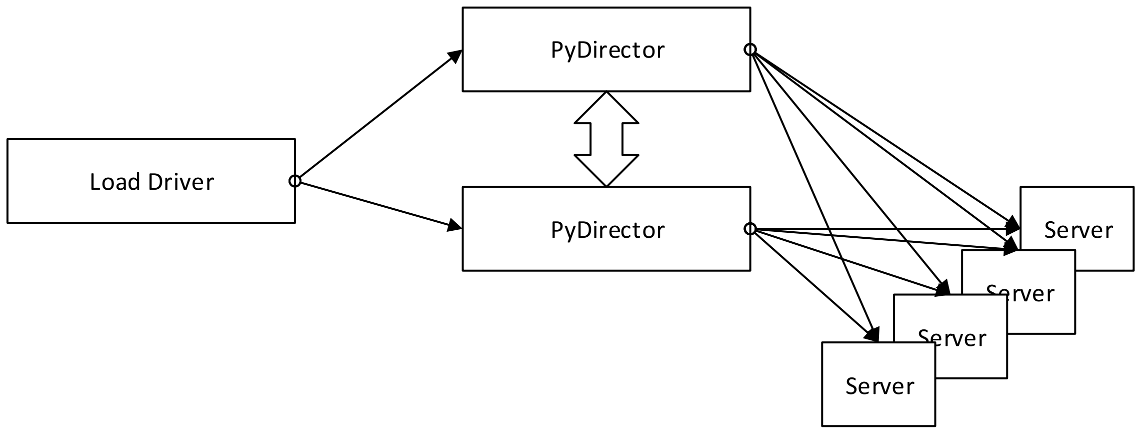
This modified version of PyDirector is designed to provide web server load balancing in a distributed fashion. Multiple load balancers may be used to direct web traffic to a set of host servers. These load balancers communicate with each other, using their combined connection data to decide which host server to forward requests to.

Modified PyDirector (Load Balancer): This component is a load balancer that receives http requests and forwards them to one of several web servers based on an algorithm that determines which server is experiencing the lightest load. The vanilla PyDirector has been modified to allow multiple instances of PyDirector to communicate with each other, passing connection data that is fed to the algorithm for determining loading.

Web Servers: The web servers chosen for this experiment were the following Google servers:

- 173.194.46.65:80
- 173.194.46.66:80
- 173.194.46.67:80
- 173.194.46.68:80
- 173.194.46.69:80
- 173.194.46.70:80

Load Driver: This component is a testing tool that transmits a series of HTTP requests. It is configurable via an XML file.



Requirements:

The load balancer **shall** provide the basic functionality offered by PyDirector.

The load balancer **shall** provide connection statistics through an API callable using XML RPC.

The IP address and port of the RPC server **shall** be configurable.

The list of peer load balancers in the distributed network **shall** be configurable.

The load balancer **shall** obtain connection data from its peers using RPC.

The load balancer **shall** use connection data obtained from peers to determine which web server has the fewest connections.

The load balancer **shall** direct a new HTTP request to the web server with the fewest connections.

Instructions for Use:

Configuration files used during testing are available with the source code.

Execute the load balancer with XML configuration file “LoadBalancer1.xml”

```
sudo python pydir.py LoadBalancer1.xml
```

Execute the load driver with XML configuration file “config.xml”

```
sudo python driveload.py config.xml
```

Modifying the load balancer configuration file:

As shown in the example configuration file (LoadBalancer1.xml), to set the RPC server IP and port, specify the following within <pdconfig> where “ip” is the IP and “port” is the port:

```
<rpcserver ip="ip:port" />
```

To add load balancer peers, add the following within <group:

```
<peer name="the_name” ip="ip:port" />
```

Testing:

Test 1:
Procedure: Only one web server configured into place with two load-balancers. Drive load to saturation and mark the rate.

Results:

Requests Per Second	Average Response Time (s)
25	0.423
50	0.426
100	0.447
200	0.973

Test 2:
Procedure: Two web servers configured with two load-balancers. Drive load to saturation again and mark the rate.

Results:

Requests Per Second	Average Response Time (s)
25	0.409
50	0.421
100	0.447
200	1.094

Test 3:
Procedure: Four web servers configured with two load-balancers. Drive to saturation and mark the rate.

Results:

Requests Per Second	Average Response Time (s)
25	0.413
50	0.418
100	0.466
200	1.124

Test 4:
Procedure: Full configuration, maximum load-balancers, maximum web servers. Drive a heterogeneous load with one of the load balancers taking many more connections than the other(s).

Results:

Requests Per Second	Average Response Time (s)
25	0.405
50	0.421
100	0.419
200	0.452

Conclusion:
Since the load balancers experience very little deviation in average response time until they hit saturation, they may be considered to perform fairly optimally. Also, the fact that the performance in the case of the unbalanced load was not worse than the balanced loads suggests that the communication between the load balancers allowed them to optimize performance.