Project 4: Load Balancing for Web Servers Using PyDirector

Due: 11:59pm, Sunday, April 27, 2014

I. Overview

The objective of this machine problem is to study load balancing across multiple servers. In this project, a student will modify an existing load-balancing application to work in a distributed way. This means that incoming requests for a website will be addressed to two or more load-balancers (PyDirector), which forward requests and responses to the web servers themselves.

II. Project Requirements

A. Introductory Material

A load-balancer for web servers is an application that forwards incoming requests and outgoing responses. The purpose of the load-balancer is to reduce bottlenecks in performance by sharing computational burden. This is achieved by tracking some statistics about the server-utilization and determining where to forward requests based on those statistics. The load-balancer that will be used for this project is called PyDirector. It is written in Python and is capable of forwarding HTTP connections to arbitrary servers. That is, the web servers do not require special software installed on them. It is assumed that the web servers for this project necessitate load balancing.

B. Execution Scenario

At least two load-balancers must be run, each on a separate machine (for testing/demo purpose you can run them on same machine, but on different ports). There will be several web servers, assuming that all mirroring the same content (synchronization between these web servers is not part of this project). There will be at least twice as many web servers as there are load-balancers. The load-balancers will each be configured to forward to the entire set of all available web server mirrors. The load balancers must share information between each other in order to coordinate balanced connection forwarding. Specifically, the load balancers will share their connection information between each other. Figure 1 shows the connection path from a client (in this case, the load driver) to the web servers.

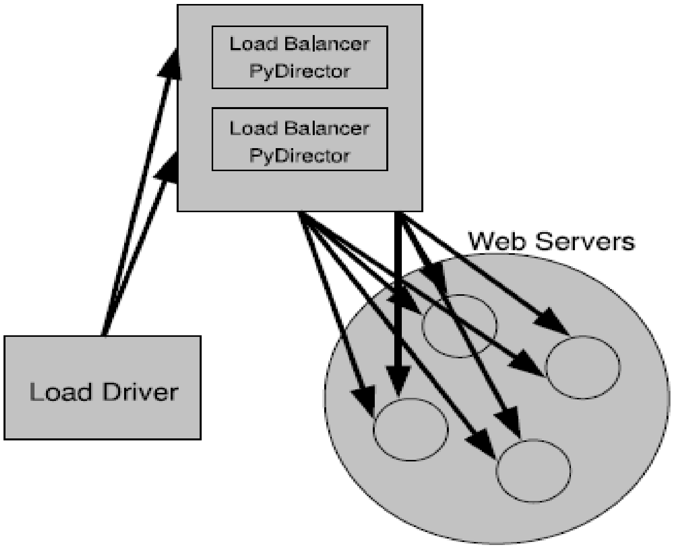

Fig. 1. System Architecture: Block Diagram

C. Requirements

1) The load-balancers must share their current connection status among each other.
2) The load-balancers must coordinate connection forwarding to avoid the scenario where one load-balancer forwards connections to a server that is already overburdened by another load-balancer's connections.
3) The modifications must use the existing framework put in place by PyDirector. This framework provides a means for: configuration, statistics tracking, and connection forwarding. Other capabilities can extend the framework, but shouldn't replace or change the existing framework. If a change is felt to be needed, as long as it can be justified, it will be allowed.

D. Assumptions

To help simplify the design process, the following assumptions can be made:
1) The load-balancers can consider the web servers to be capable of equal performance. That is, no server is more or less powerful than another. (Handling a heterogeneous group of servers is optional.)
2) The load-balancers will be able to forward to the entire set of all available web servers (mirrors), if they are active and running. That is, the forwarding capabilities between the load balancers will be identical.
3) For development and testing, it is acceptable to run the load-balancers on the same machine, but on different ports. However, do not implement it in such a way that it becomes limited to running on the same machine.

E. Components

1) PyDirector (The Load Balancer): PyDirector can be run on any system that has Python installed, and some minimal networking packages. PyDirector must be run in at least two instances. Each will be configured identically, except for their references to each other. The modification to PyDirector should make use of the existing configuration capabilities (XML-based) to let each instance of the load balancer know about its peer(s). They should refer to each other by IP address and port number.

To communicate between instances of PyDirector, a few options exist. There's XML-RPC, which is part of the standard python libraries in module: xmlrpclib. Also, there's a good remote object package called PYRO which works much like Java RMI. Lastly, sockets can be used. Other techniques would likely include less common packages and are not recommended, because this project has to be testable on a limited number of machines. Development for this component can take place on any capable system. Test cases will be run on a Linux machine capable of using the Twisted networking package.

2) Web Servers: The web servers use a content management system to ensure that processing is the bottleneck instead of network bandwidth. This effect is necessary in order to test the effectiveness of load balancing.

3) Load Driver : This is a testing tool that sends a series of HTTP requests and measures responsiveness. Configured with a list of IP addresses and ports. Given a target request rate, it will continue to send for a specified period of time and measure the delay in response from the servers.

F. Testing

The project will be tested to primarily measure how well the loads are balanced. This will be accomplished by running the Load Driver on several configurations.
1) Only one web server configured into place with two load-balancers. Drive load to saturation and mark the rate.
2) Two web servers configured with two load-balancers. Drive load to saturation again and mark the rate.
3) Four web servers configured with two load-balancers. Drive to saturation and mark the rate.
4) 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).

The purpose of these test cases is to create a characterization of how close to optimally the load balancers perform.

III. What to Hand In

Please compose a system level design document and include your test results. Make sure it convinces the reader that you know how to attack the problem. The source code and report should be submitted through WebCT. Please schedule a time with the TA to show the project demo between April 28, 2014 and May 2, 2014.