MIECT: Integrated Management of Networks and Services 2011-12

2nd Project: Load Balanced Reverse Proxy

Deadline: December 30th v1.0

December 3, 2011

1 Objective

Acquire experience with reverse proxies, high availability and load balancing solutions.

2 Description

Reverse proxies are an vital part of todays web infrastructure. They allow hiding internal network structure, provide security mechanisms and can also load balance requests to a cluster of upstream nodes. With reverse proxies, systems can scale horizontally, simply by adding more upstream nodes, without introducing any perceived change to clients.

One of the most used proxy is NGINX. Netcraft estimates that this software is used to serve more than 43M websites, which also includes SAPO. NGINX is highly scalable, easy configurable and modular, supporting several methods to load balance requests to nodes.

The purpose of this project is to enhance the NGINX reverse proxy so that it is able to load balance requests to upstream nodes, while taking in consideration the actual load of each node. Currently, there are several modules providing load balancing mechanisms for HTTP requests, namely the `HTTPUpstream` and the `HTTPUpstreamFair`. The first selects nodes based on a round-robin approach (see see `ngx_http_upstream_get_round_robin_peer` function in `src/http/ngx_http_upstream_round_robin.c` in NGINX source), while the second takes in consideration the number of requests each server actual serves. This number is a coarse estimation of the load the node has.

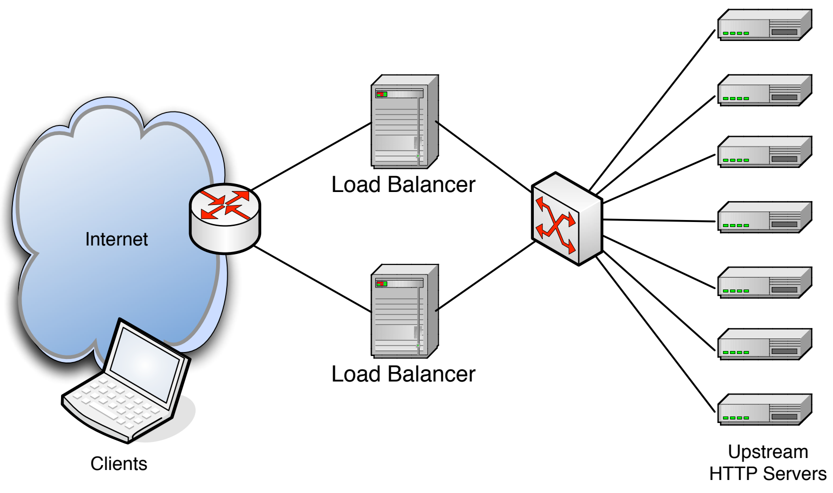

Figure 1: A simple load balance architecture for Web Services.

Students should develop a method to estimate the load of upstream servers, either by direct probing or by other indirect heuristic, and use this information to enhance the NGINX load balancing algorithms. The `HTTPUpstream` module, can be easily modified so that additional metrics are taken in consideration.

After the solution is developed, students should benchmark their solution and compare it with the `HTTPUpstream` module. We recommend that students make use of small virtual instances of Debian with `lighttpd`. These images can take less than 1GB of disk space and less than 256MB of RAM. Therefore, several can be deployed as upstream nodes serving some test pages. A virtual appliance which includes `lighttpd` and `PHP` is available at `http://atnog.av.it.pt/~jpbarraca/classes/girs/`. For benchmarking, the Apache Benchmark `ab` command line tool can be used to benchmark systems and generate reports.

If using PHP, delay can be introduced using the `sleep()` function. Processing load can be produced with simple loops, while IO Load can be produced by opening and reading files. The code present in the end of this document can be used to create a constant CPU occupancy.

Any programming language can be used to implement this project. Due to the `NGINX` code base, at least some components will require the use of C.

Delivery should include the code developed, description, validation and evaluation of the solution proposed.

3 References

- NGINX: `http://wiki.NGINX.org/Main`
- NGINX Upstream module: `http://wiki.NGINX.org/HttpUpstreamModule`
- NGINX Upstream Fair module: `https://github.com/gnosek/NGINX-upstream-fair`
- Lighttpd: `http://www.lighttpd.net/`
- Apache Benchmark: `http://httpd.apache.org/docs/2.2/programs/ab.html`

4 Support code

```
// Author: Joao Paulo Barraca <jpbarraca@ua.pt> 2011
// License: GNU GPLv3
// This program produces a controlled amount of CPU usage in a single core.
#include <unistd.h>
#include <stdio.h>
#include <sys/time.h>

#define INTERVAL 0.04   //Interval between recalculation (40ms)
static int dummy = 0;   //To avoid optimization by some compilers

void print_usage(char* name)
{
  printf("Usage: %s <load_percentage>\n",name);
  printf("\tload_percentage\t\t: The load percentage. Range: 1-100.\n",name);
}

int main( int argc, char *argv[] )
{
  unsigned long i, cycles = 100000, target_load = 0;
  timeval start, stop;
  double spentTime;

  if(argc == 2) {
    int ret = sscanf(argv[1], "%u",&target_load);
    if(ret != 1 || target_load == 0 || target_load > 1000) {
      print_usage(argv[0]);
      return -1;
    }
  } else {
    print_usage(argv[0]);
    return 0;
  }

  for(;;) {
    gettimeofday(&start,NULL);
    for(i=0; i<cycles; i++) {
      dummy += i*i;
    }
    gettimeofday(&stop,NULL);

    spentTime = (stop.tv_sec - start.tv_sec) + (stop.tv_usec-start.tv_usec) / 1000000.0;
    cycles = (cycles / spentTime ) * (target_load / 100.0)  * INTERVAL;
    usleep((1000000 * INTERVAL - spentTime*1000000) );
  }
  return 0;
}
```

Listing 1: Program to create a controlled amount of CPU occupancy