CS 4700 / CS 5700 - Network Fundamentals

Project 5: Roll Your Own CDN

This project is due at 11:59pm on December 9, 2016. The milestone is due at 11:59pm on November 25, 2016.

Description

By now, you have learned that most of the content we consume over the Interent is served by content delivery/distribution networks (CDNs). In this project, you will create the building blocks of a CDN. Unlike previous projects, a portion of your grade in this project will come from how your code performs compared to the solutions of your classmates, i.e. this assignment is a competition.

CDNs consist of 1) a large number of servers geographically distributed worldwide; 2) a system that maps clients to "good" replica servers and 3) a system that determines those mappings. In this project, you will implement the basic functionality in each of these areas. Thanks to generous support from Amazon EC2, you will build a CDN that uses EC2 sites as replica servers. Your performance will be compared to two baselines: origin (a single server in a single EC2 location) and random server selection. You should do better than both.

Project Requirements

You must implement a CDN using the following features. First, you will use DNS redirection to send clients to the replica server with the fastest response time. Second, you will write a simple Web server that returns content requested by clients. Third, you will implement a system that uses information about network performance, load on servers, and cached data at servers to determine the best replica server. Performance will be measured in terms of how long it takes to download a piece of content. Similar to most Web sites, most content will be small in terms of bytes.

I will test your code using simple clients that 1) ask your DNS server for the IP address of *cs5700cdn.example.com* and 2) uses an HTTP get to fetch a file from that address. These clients will run from servers located all over the world. The request frequency for each piece of content will follow a Zipf distribution (http://en.wikipedia.org/wiki/Zipf's_law), and the size of content at the origin server is much larger than the size of your cache on each replica server. In other words, you must implement a cache replacement strategy at each replica server because you can't fit all the requested content at each cache. If your replica server receives a request for content not in its cache, the server must fetch it from the origin and return that to the client.

When determining the replica server to use for each client request, you should determine which one will give the best time-to-completion (TTC) for downloading a Web page. This can depend on the network latency, loss, available bandwidth and load on the server. You can choose any technique(s) you want for estimating these properties, but online (i.e., active measurement) techniques are the most likely to suceed if implmented properly.

Important!

This project requires you to access EC2 nodes and use them as content caches. For this project, I have made a large corpus of public-domain data available (i.e., Wikipedia). **Do not use any other content in your system, particulary copyrighted work.** Further, you cannot use accounts on EC2 for any purpose other than running your replica server code. **If we find you using our EC2 VMs for any other purpose, your account will be deleted and you will receive a zero on this project.**

Your Programs

For this project, you will submit three independent pieces: a DNS server, an HTTP server with cache management, and code that manages the mappings between clients and replicas.

The DNS server will be called using:

```
./dnsserver -p <port> -n <name>
```

where *port* is the port number that your DNS server will bind to and *name* is the CDN-specific name that your server translates to an IP. In this case, I will use *cs5700cdn.example.com*. The DNS servers must run on **cs5700cdnproject.ccs.neu.edu**. You must run this server on a high-numbered port (40000-65535). **You should assume that your HTTP server will run on the same port that you specify here.**

Note that unlike in a traditional CDN, I will be requesting name translations directly from the same host as the Web client. Thus you do not have to correct for the distance between Web clients and their DNS servers. **Note that the server will be rebooted every night, so you cannot rely on long-running code on this server.**

The HTTP server will be called using

```
./httpserver -p <port> -o <origin>
```

where *port* is the port that your HTTP server will bind to and *origin* is the name of the origin server for your CDN. This code is responsible for delivering the requested content to clients. It may do so by fetching uncached content for the origin, or a local cache of Web pages. Disk quotas have been set such that the size of the cache on each replica server is much less than the size of all the requestable content. Your server is also responsible for managing the cache of Web pages on the host where it runs. Again, note that you must run this web server on a high-numbered port (40000-65535) for testing. **You may use an in-memory cache no larger than your disk quota (10MB).**

You may execute other measurement and mapping code on the replica servers and DNS server. To facilitate this, you should provide a *deployCDN*, a *runCDN* script and a *stopCDN* script. For example, the *deployCDN* script will copy code to an EC2 instance and run a Makefile. The *runCDN* script will cause the server to run.

Your scripts will be called as follows:

```
./[deploy|run|stop]CDN -p <port> -o <origin> -n <name> -u <username> -i <keyfile>
```

where *port*, *origin* and *name* are the same as above, *username* is the account name you use for logging in and *keyfile* is the path to the private key you use for logging into nodes. Your scripts should use the last two arguments as follows:

```
ssh -i keyfile username@<some EC2 server> ...
```

Testing Your Code

To test your code, you will be given access to each of the EC2 VMs. The list of these VMs is located at:

```
/course/cs5700f16/ec2-hosts.txt
```

Note that the origin server you should use is described in this file. Do NOT use any other server for testing.

In addition, we will set up *DNS lookup beacons* and *HTTP client beacons* that periodically perform DNS lookups and fetch content from each VM on a range of ports. By instantiating a Web server on a port in that range, you will periodically receive requests for name translations and Web pages. You may also use CCIS servers, EC2 servers and any other servers you have available to test the quality of your mappings.

To perform a DNS lookup, use the *dig* tool. For example, you would want to call:

```
dig @[your DNS server IP] -p [your port]
```

To fetch and time Web page downloads, you can use:

```
wget http://[your server name]:[your port name]/[path to content]
```

Preceding this call with *time* will allow you dettermine how long a download takes to complete.

Your grade will in large part depend on the time it takes to download content from the server you send my client to. Because you are download text Web pages, you can expect the flows to be short so latency and loss are likely to dominate your performance. You should develop a strategy for identifying the best server for a client and test it by comparing with performance from other EC2 sites.

Tips and Tricks

To locate the server with the lowest latency, you can use active measurements, passive measurements and/or exogenous information such IP geolocations. **If you choose to use active measurements, you must limit your measurement rate to no more than 1 probe per second. I suggest using scamper, which is already installed on the EC2 machines.**

You can deploy code without interactive passwords by using SSH key-based authentication. You will be expected to use this technique to execute your *deployCDN* and *runCDN* scripts.

Your CDN can be terminated at any time. Make sure you use persistent storage (i.e., files on disk) to maintain information about content cached at various nodes and any information you need to map clients to servers.

Grading

Your grade in this project will be composed by the following components:

- 5 points - Implementation of a DNS server that dynamically returns IP addresses based on your mapping code.
- 5 points - Implementation of a HTTP server that efficiently fetches content from the origin on-demand and optimizes the cache hit ratio.
- 5 points - Implementation of a system that maps IPs to nearby replica servers.
- 3 points - Performance in the TopCDN competition
- 2 points - A short (no more than 2 pages) report describing the design decisions you made, how your evaluated their effectiveness and what you would do with more time. Include this in the README file.

The final competition will be held on May 1st. We will test each CDN against two baselines: origin and random server selection. Those that do better than random and origin will get 1 point. The top 1/3 scores will earn onother 2 points (for a total of 3), and the next 1/3 will earn an additional point (for a total of 2). Students may use slip days and turn in the assignment late; however late assignments will not be eligible for TopCDN points.

Extra Credit

The top ranked team in the TopCDN project on December 10th will earn 3 bonus points! :)

Registering Your Project

Before turning in your project, you and your partner(s) must register your group. To register yourself in a group, execute the following script:

```
$ /course/cs5700f16/bin/register project5 [team name]
```

This will either report back success or will give you an error message. If you have trouble registering, please contact the course staff. **You and your partner(s) must all run this script with the same [team name]**. This is how we know you are part of the same group.

Submitting Your Milestone

There is a milestone for this project. At two weeks in, you need to turn in working DNS servers and caching HTTP clients. The remainder of your time will be spent on the CDN logic that links them together. This is due on **November 25th, 2016 at 11:59:59pm**:

- A Makefile that compiles your code.
- An httpserver executable
- A dnsserver executable
- A plain-text (no Word or PDF) README file. In this file, you should briefly describe your high-level approach, what specific performance enhancing techniques you implemented, and any challenges you faced.

Your README, Makefile, source code, etc. should all be placed in a directory. You submit your project by running the turn-in script as follows:

```
$ /course/cs5700f16/bin/turnin project5-milestone [project directory]
```

[project directory] is the name of the directory with your submission. The script will print out every file that you are submitting, so make sure that it prints out all of the files you wish to submit!

Submitting Your Final Project

To turn-in your final project, you should submit your (thoroughly documented) code along with three other files:

- A Makefile that compiles your code.
- An httpserver executable
- A dnsserver executable
- A deployCDN script
- A runCDN script
- A stopCDN script
- A plain-text (no Word or PDF) README file. In this file, you should briefly describe your high-level approach, what specific performance enhancing techniques you implemented, and any challenges you faced.

Your README, Makefile, source code, etc. should all be placed in a directory. You submit your project by running the turn-in script as follows:

```
$ /course/cs5700f16/bin/turnin project5 [project directory]
```

[project directory] is the name of the directory with your submission. The script will print out every file that you are submitting, so make sure that it prints out all of the files you wish to submit!

Do **not** turn in files larger than 10MB, as these will be rejected.

Only one group member needs to submit your project. Your group may submit as many times as you wish; only the time of the last submission will determine whether your assignment is late.

David Choffnes, Assistant Professor, College of Computer & Information Science, Northeastern University.
Last updated November 18, 2016.