CS 4700 / CS 5700 - Network Fundamentals

Project 5: Roll Your Own CDN

This project is due at 11:59pm on May 3, 2022. The milestone is due at 11:59pm on April 19, 2022. The due dates are the same for both UG and MS/PhD students.

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 cloud servers, you will build a CDN that uses cloud sites as replica servers. Your performance will be compared to two baselines: origin (a single server in a single cloud 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 cloud 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 cloud for any purpose other than running your replica server code. **If we find you using our cloud VMs for any other purpose, your account will be deleted and you will receive a zero on this project.**

Detailed Requirements

The main focus of the project is **HTTP cache management** and **DNS redirection**.

For this project, you will submit: a DNS server, an HTTP server with cache management, code that manages the mappings between clients and replicas, and scripts to manage deploying, running and stopping your CDN.

Libraries:

Libraries that offer the full functionality of any of the required services are prohibited. For example:

- dnslib is allowed
- dnspython is not

As the focus of the project is on cache management and not serving HTTP, there is an exception for Python's http.server or an equivalent in another language. Anything more fully functioned is prohibited.
If you have a question regarding whether a library is permitted, **ask before using** it.

DNS:

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.

The DNS server only needs to respond to **A** queries for the site specified in the project description.

HTTP:

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. **Note that the origin server is running a Web server on port 8080**, not 80.

Student's HTTP servers must respond to an HTTP request for

```
/grading/beacon
```

with a **204** status code.

Scripts:

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 cloud server> ...
```

Deployment Specifics

- Origin Server: HTTP on port 8080 [ssh access is not available]: **cs5700cdnorigin.ccs.neu.edu**
- Build server: **cs5700cdnproject.ccs.neu.edu** [**Note**: This is a Khoury server, so you need your Khoury login credentials here.]
- List of servers for deploying DNS: see the pinned post on Piazza (https://piazza.com/class/kyc1vsrynkx2mv?cid=146)
- List of replicas for deploying HTTP Caches: see the pinned post on Piazza (https://piazza.com/class/kyc1vsrynkx2mv?cid=146)
- A listing of page views for the origin server can be found on Piazza (https://piazza.com/class/kyc1vsrynkx2mv?cid=146)

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 the port given for testing.

All resources are shared across all teams during testing. For simplicity, every team will be assigned a port number (tbd) to use for their testing. The single port will be used for both the DNS serverand the HTTP replicas.

Per-user account limits on each instance:

- Max 2 Simultaneous Logins (should only have 1)
- Total 20MB Disk space across all folders
- RSS limited to 20MB [**Note**: We will not enforce this at runtime but we will check your code for limited use of in-memory caching]

Submissions that use subprocess will be ineligible for TopCDN credit.

Testing Your Code

To test your code, you will be given an account on each of the cloud VMs described above. **Do NOT use any other server for testing.**

The domain name querried will be *cs5700cdn.example.com*. Your DNS servers must run on a cloud instance listed in **dns-hosts.txt**. You must run this server on a high-numbered port that will be given to you when you register. **Unless you are told different, you should assume that your HTTP server will run on the same port.**

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 Khoury servers, cloud 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:

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

time will allow you determine 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 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 cloud sites.

Measurement

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

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.**

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 (https://www.caida.org/catalog/software/scamper/), which is already installed on the cloud machines.**

Beware of relying on-line data via web APIs. These servers can rate limit and cut off your access.

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:

- 9 points - Implementation of a DNS server that dynamically returns IP addresses based on your mapping code.
- 9 points - Implementation of a HTTP server that efficiently fetches content from the origin on-demand and optimizes the cache hit ratio.
- 9 points - Implementation of a system that maps IPs to nearby replica servers.
- 5 points - Performance in the TopCDN competition
- 3 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 5th. 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 another 3 points (for a total of 4), and the next 1/3 will earn an additional 2 points (for a total of 3). Students may use slip days and turn in the assignment late; however late assignments will not be eligible for TopCDN points. **All projects must be submitted by May 4th.**

Extra Credit

The top ranked team in the TopCDN project on May 5th will earn 3 bonus points! :)

Submitting Your Milestone

The milestone is due **as stated above**. The milestone is to perform an early **'Does it run?'** evaluation of your DNS server, a caching HTTP client, and your deployment scripts, by deploying a single DNS server and a single caching HTTP client in the cloud using your deployment scripts.

For the milestone, you must submit:

- A Makefile that compiles your code

- httpserver source
- dnsserver source
- A deployCDN script
- A runCDN script
- A stopCDN script
- A plain-text (no Word or PDF) README file. In this file, you should describe:
  - In brief, your high-level approach, how you implemented your DNS and HTTP Servers, and any challenges you faced.
  - **(For groups) In detail, a description of which student worked on what parts of the code.**

For the milestone, use the following commands to submit:

```
$ /course/cs5700sp22/bin/register project5 [team name]
```

```
$ /course/cs5700sp22/bin/turnin project5-milestone [project directory]
```

Your README, Makefile, source code, etc. should all be placed in a directory (no subdirectories please). Your group may submit as many times as you wish. Only the last submission will be graded, and the time of the last submission will determine whether your assignment is late.

Submitting Your Final Project

The project is due **as stated above**. To turn-in your final project, you must submit your (thoroughly documented) code along with three other :

- A Makefile that compiles your code
- httpserver source
- dnsserver source
- A deployCDN script
- A runCDN script
- A stopCDN script
- A plain-text (no Word or PDF) README file. In this file, you should describe:
  - In brief, your high-level approach, how you implemented your DNS and HTTP Servers, and any challenges you faced.
  - **(For groups) In detail, a description of which student worked on what parts of the code.**

Your README, Makefile, source code, etc. should all be placed in a directory (no subdirectories please). Submit your project as a zip of that directory to Gradescope. **Only one group member needs to submit your project. Remember to add your teammate when submitting.** You may submit as many times as you wish; only the time of the last submission determines whether your assignment is late.
