Project 6: Content Delivery Network

Roll your own CDN

This project is due at 11:59pm on Monday, April 15, 2024.

Description

By now, you have learned that most of the content we consume over the Internet 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.

High-level 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, 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

but online (i.e., active measurement) techniques are the most likely to succeed if implemented properly.

Low-level 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]
```

`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. Both parameters are required.

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]
```

`port` is the port that your HTTP server will bind to and `origin` is the name of the origin server for your CDN. Both parameters are required.

This process is responsible for delivering the requested content to clients. It may do so by fetching uncached content from 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 the path `/grading/beacon`

Scripts

Your scripts will be called as follows:

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

`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://cs5700cdnorigin.ccs.neu.edu:8080*. SSH access is not available; this is simply an HTTP server that hosts content.
- Build server: *cs5700cdnproject.ccs.neu.edu*. Use your Khoury credentials to log in.
- Server for deploying DNS: TBD
- List of replicas for deploying HTTP Caches: TBD
- A listing of page views for the origin server: TBD

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 (in registration information) to use for their testing. The single port will be used for both the DNS server and the HTTP replicas.

Per-user account limits on each instance:

- Maximum of two (2) simultaneous SSH logins. You should only have one (1); the second connection is for emergencies (i.e., to kill runaway processes).
- Total 20MB disk space across all folders.
- RSS (i.e., process RAM) limited to 20MB. Note: we may not enforce this at runtime but we will check your code for limited use of in-memory caching.

Submissions that use subprocess, multiprocessing, or other forms of process forking will be ineligible for extra 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 queried will be cs5700cdn.example.com. Your DNS servers must run on a cloud instance (replicas) listed above. You must run this server on a high-numbered

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`, 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. **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, 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

Submission

To turn-in your project, you must submit the following things:

1. A `Makefile` that compiles your code
2. Well-documented `httpserver` source code
3. Well-documented `dnsserver` source code
4. A `deployCDN` script
5. A `runCDN` script
6. A `stopCDN` script
7. A plain-text (no Word or PDF) `README.md` 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) Identify which student worked on what parts of the code.

Your `README.md`, `Makefile`, source code, etc. should all be placed in the root of a compressed archive (e.g., a .zip or .tar.gz) and then uploaded to Gradescope. Do not use the archive option on a Mac—Gradescope does not parse these archives. Instead, use the `zip` program on the command line. Alternatively, you can check **all** these items into Github, download a zip file from Github, and submit that 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.

Grading

This project is worth 15 points. You will receive 11 points if you:

- Implement a system that maps IPs to nearby replica servers.
- Implement a DNS server that dynamically returns IP addresses based on your mapping code.
- Implement a HTTP server that efficiently fetches content from the origin on-demand and optimizes the cache hit ratio.
- Thoroughly describe the design decisions you made, how your evaluated their effectiveness, and what you would do with more time in your `README.md` file.

The remaining 4 points will based on the performance of your solution in the Top CDN competition, which will be held after April 12. 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).

GENERAL

Syllabus

Schedule

PROJECTS

Project 1: Socket Basics

Project 2: FTP Client

Project 3: BGP Router

Project 4: Reliable Transport Protocol

Project 5: Web Crawler

Project 6: Content Delivery Network

Powered by Netlify, Hugo, and Doks