Project 4: CDN

Due: 11:59 PM, Dec 12, 2019

Contents

1 Background **2**

2 Overview **2**

3 Components **3**

3.1 Driver 4

3.2 Client 4

3.3 DNS Server 4

3.4 Edge Servers 4

4 Task **4**

4.1 Potential Optimizations 5

4.1.1 Geolocation 5

4.1.2 Ping Optimizations 5

5 Getting Started **6**

6 Create a VM instance on GCP **6**

6.1 Redeeming the Coupon Code 6

6.2 Using Cloud Console to Create a New Project 6

6.3 Create an incoming DNS firewall rule 7

6.4 Creating a new VM 7

6.5 SSH into the VM 8

6.6 Persistently Running Code 8

6.7 Shutdown / Delete the VM after Grading 9

7 Leaderboard **9**

8 Grading **9**

1 Background

Content Distribution Networks (CDNs) are widely used to distribute massive amounts of data to users all around the world. For instance, the popular CDN Akamai transports web traffic at nearly 50 terabits per second.[1]

At it's core, a CDN is intended to allow users to quickly access geographically distributed web content. To that end, Akamai's services include load balancing, caching, and several other optimizations to maximize the speed at which users can access data across the internet. In this project you are going to build one critical component of a CDN, a DNS-based redirection service. However, given the small class size and the minimal dataset we'll be using, you don't have to worry about most of these features, because there will not be heavy traffic, and the entire content will fit in several service replicas.

You will, however, be in charge of optimizing performance for the user. By the end of this project you will have:

a. Implemented a DNS server using industry-grade DNS tools

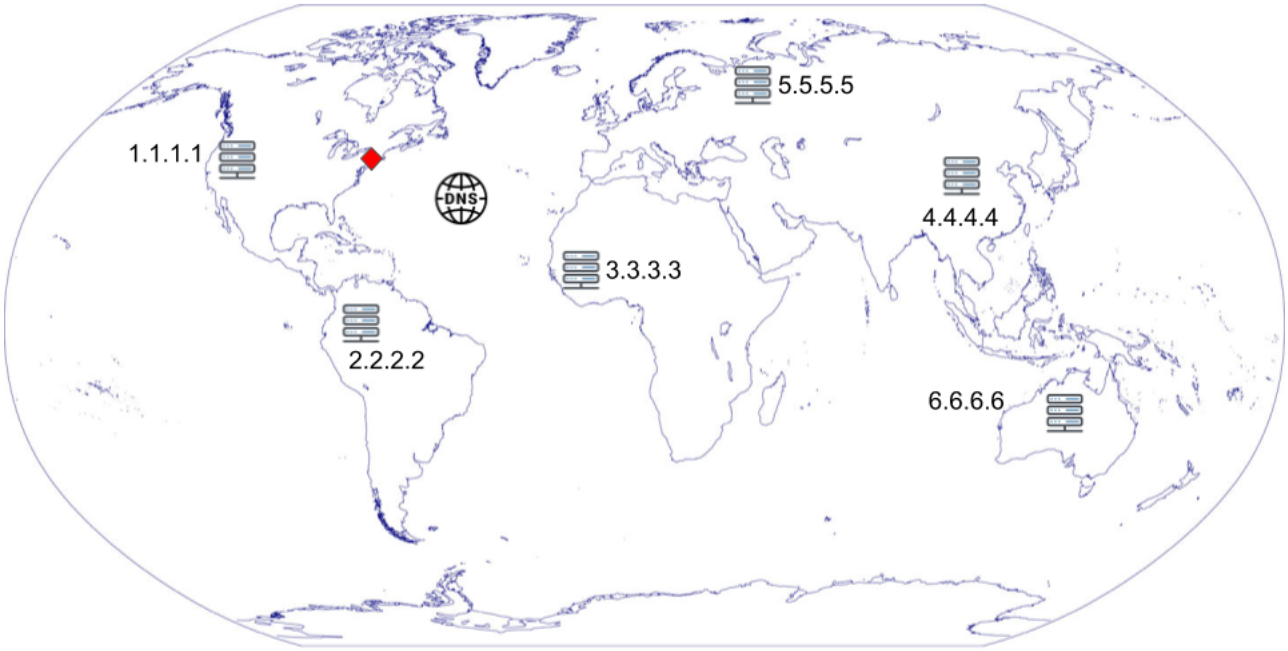
b. Learned the fundamentals of the Google Cloud Platform

c. Familiarized yourself with the architecture of actual production-level CDN systems which dominate the world wide web

2 Overview

In this assignment you will write a DNS server and run it on a VM instance on Google Cloud Platform (GCP). Its job will be to resolve DNS queries to the edge server that is optimal for the client. Consider the diagram below:

[1] https://www.akamai.com/us/en/about/facts-figures.jsp



Suppose an web user located at the red diamond wants to go to `example.com`. The various servers around the world are known as *edge servers*, and in our configuration each edge server has a copy of `example.com` which the client can request. However, the user does not know which of these servers to issue the `HTTP GET` request to for optimal performance. One of the ways a CDN does this is through DNS redirection: the CDN is responsible for resolving `example.com` into an IP address, and it uses information from the DNS request to choose an appropriate server. This is what you will be implementing.

Represented by the globe in the ocean on the figure above, the DNS server should receive a DNS query, for example:

```
www.example.com: type A, class IN
    Name: www.example.com
    Type: A (Host address)
    Class: IN (0x0001)
```

In response, your DNS server should send a valid DNS response which tells the client which edge server is optimal to request the site data from. We will come back to this example when we explain optimizations you can choose to make to improve performance in your DNS server.

3 Components

We have implemented most portions of the CDN for you, and in this section we will detail what we have done and what you need to do. In particular, we are running a set of globally distributed server replicas and clients, and we have written a driver program that instructs a particular client to request an object from the servers. The client will use your DNS server to choose the best server,

and make the request to that server. The client reports the time it took to fetch the object. We will use the driver to test how well your DNS server chooses the best server for each client.

3.1 Driver

We have built a driver for you, which can be found in your starter repository (see the Getting Started section). This simulates an application which can be used to test your DNS server. Concretely, it may be invoked either as a standalone program or as a REPL. For specific usage, simply execute `./driver.py` and type `h` for detailed instructions. In either the standalone or REPL case, when you issue a request with the driver it relies on a client to perform the bulk of the processing.

3.2 Client

We have spun up several clients located around the world on the GCP. These clients take as input requests from the driver and then perform the following steps:

- Send a DNS query using `dig` to the provided DNS server IP/port.
- Parse the response and then issue a GET request for the web content to the resolved edge server.
- Return the time taken to get the data from the edge server and the web content.

We provide a list of clients which you may query in `client-list.csv`. The list also contains the approximate latitude and longitude of each client. We will test your solution with a different set of clients, but you can assume that your DNS server will have access to a file with the IP addresses, latitude, and longitude of the clients.

3.3 DNS Server

You must implement this portion of the system. It should listen for DNS queries from clients and respond with the address of the edge server which the client should request the site data from. More on this below.

3.4 Edge Servers

We have also spun up several HTTP edge servers which contain the same web content. They are are also running around the world on GCP machines. When content is requested, they will return the actual web page. We provide a list of edge servers which you may resolve requests to in `server-list.csv`. This file also contains latitudes and longitudes of the servers, which you may need to optimize (more on this below).

4 Task

Your goal is to effectively implement a DNS server which instructs clients to use the optimal edge server. When activated, it should be able to receive DNS queries from the geographically dispersed

clients and then return the best server for each client. The best in this case is the server replica that would result in the smallest latency. The tricky part is on how to do this. You will be graded on the average latency to request content from each client to each server. Since the clients and servers are located all around the world, latencies can range from <10ms to >200ms.

A real CDN has to do this for potentially any client, and combines several methods. It can use the geographic location of the requesting client and/or of the requesting DNS resolver the client is using; it can use databases of latency measurements between different IP subnets; or it can use active measurements from different servers to the client. To make your life simpler, the IP address issuing the DNS request will always be the same as the IP of the client which will fetch the request. We also have a limited number of clients and servers. You should NOT try to pre-compute the best server for each client, and MUST do a dynamic determination for each client request.

4.1 Potential Optimizations

4.1.1 Geolocation

Geolocation is a simple way of optimizing data requests. Simply, an end-user should fetch data from the edge server which is geographically closest to them, as it will decrease latency (remember what we discussed in the physical layer about propagation, queuing, transmission, and processing delays?) There are databases on the Internet that can do IP-geolocation, and normally you would query them. `https://db-ip.com/` is one of them. However, all of the IPs we are using belong to Google, and the databases we tested are not very accurate, as Google moves the IP addresses pretty arbitrarily among its geographic regions. Instead of using these databases, we are providing you with files that have the latitude and longitude of the clients and servers. You are to treat these files as IP geo-location databases: your code should read the location of each client as requests come (you can store the file as a dictionary in memory), but NOT pre-compute the answers to every client.

In the world diagram from Section 2, you would probably return 1.1.1.1 as the closest server to the red diamond. You can compute the distance between two pairs of latitude and longitude using the Haversine formula.[2]

4.1.2 Ping Optimizations

Geolocation will not always be optimal, for several reasons. First, the geo-location databases may be inaccurate. Second, Internet paths are often not optimal in terms of latencies. This can be due to BGP policies, or the physical connectivity having to go to another country before coming back to a close one, for example. Third, there can be transitent problems in any section of the network, such as excessive queueing, making a path that was optimal not be anymore. For these reasons, you may want to use actual measurements to determine the optimal server. We have implemented a ping facility that allows your DNS server to execute real-time measurements if you want. Your DNS server can, at any time, issue a request to one of the server IP addresses (which you know), asking the server to ping a specific IP address. The server will run a ping command, and return the output. The DNS server can do this upon receiving a request from client A: ask all (or a subset) of the servers to ping A's IP address, and choose the server with the lowest latency. Of course, this process adds to the latency seen by the client, so you must be careful for the gains here to be

[2] `https://en.wikipedia.org/wiki/Haversine_formula`

competitive. For example, you might not need to wait for all servers to respond. Again, if you are using this method, you should NOT assume that you know all possible clients, and thus you should NOT pre-compute answers based on all pairs of clients and servers.

To ping, you may issue the following request:

`http://<server IP addr>/ping.php?ip=<client IP address>`

E.g.

`http://34.97.58.4/ping.php?ip=128.148.32.12`

5 Getting Started

- Accept the project at `https://classroom.github.com/g/ph8qb_-t`
- Try out the driver using our demo DNS server at IP 35.212.123.161 Port 53.
- Create a Google Cloud VM (see section 6)
- Now you're setup to write and optimize your own DNS server, which you host on GCP. **We recommend using Python for this project, as it has many DNS libraries you can use.**

6 Create a VM instance on GCP

You will create your DNS server using free credits on Google Cloud.

6.1 Redeeming the Coupon Code

- Go to the following link: Google Cloud Coupon
- You will be asked for your name and email address. The email must match either `brown.edu` or `cs.brown.edu`.
- This will give you $50, which you are free to use until 9/5/2020. There can be one redemption per email, but please only redeem with one email per student, as we have exactly one coupon per student in the class.

6.2 Using Cloud Console to Create a New Project

Refer to the **Creating a project** section of the website.

6.3 Create an incoming DNS firewall rule

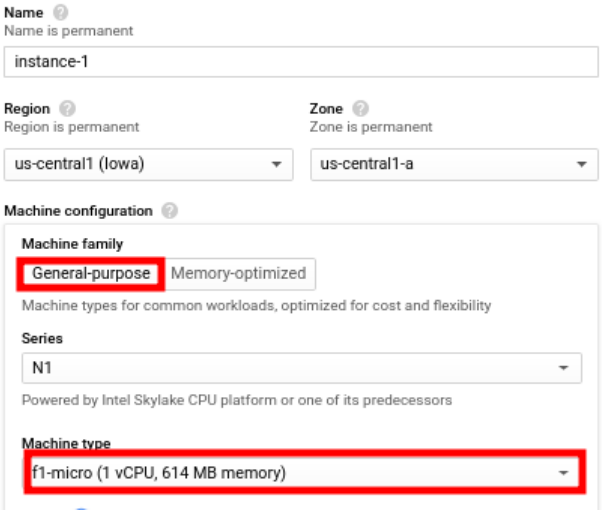
All VMs in the Google Compute Platform by default block all external traffic, so we have to explicitly open the port your DNS server will use. You can choose whichever port you want, the default for DNS is port 53.

- Navigate to the Firewall Rules Page
- Click on Create firewall rule
- Create a name, such as 'incoming-dns-from-us'
- Choose ingress for direction; 'All instances in the network' for Targets
- Choose '34.0.0.0/7' as the source IP ranges. This will encompass all of our clients. If you want you can add Brown addresses as well, to test. If things seem broken you can even temporarily use 0.0.0.0/0, but this will allow anyone on the Internet to bombard your DNS server with attack queries, and you don't want to be part of a DNS reflection attack.
- For Protocols and Ports, allow port 53 for both TCP and UDP, or any other port you want to use for your DNS server.
- Click Create.

6.4 Creating a new VM

Adapted from the python notebook made by the CS147 Staff.

- Navigate here to go to the **VM Instances** page.
- Click the **Create instance** button.
- Under **Machine configuration**, select General-purpose as Machine family and f1-micro as Machine type.

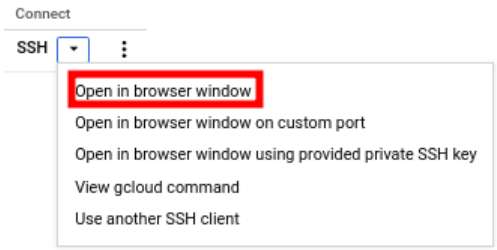


- Click on the **Create** button. It may take a few minutes to create the instance.

All other fields can be left with their default values. You should see an **external IP Address** assigned to your VM instance after the creation. For more information, refer to the **Create an instance from an image** section of the website.

6.5 SSH into the VM

You can use the web client to ssh into the VM instance you just created. On the **SSH** drop-down list in Compute Engine → VM Instance view, choose Open in browser window.



6.6 Persistently Running Code

Once you have a working DNS server, you will want it to be running persistently on the VM. If you run it in the terminal then quit your SSH session, it will turn off your server. To make it persistent, you may use any of the following utilities (in order of simplicity):

- screen or tmux these terminal multiplexing tools will let you make a terminal session then detach that session, which will keep the terminal running when you disconnect.

- nohup will let you run your server and ignore the SIGHUP signal sent when you disconnect.
- systemd will require making your DNS server into a Linux service.

6.7 Shutdown / Delete the VM after Grading

You should shut down or delete the VM instance to avoid unnecessary charge. Click on the three dots next to SSH and select Stop / Delete.

7 Leaderboard

For fun, we have implemented an online leaderboard, which updates hourly. This will automatically test your DNS server against our random version, and it will show the performance (in terms of average latency) of the current submissions. We may assign bonus points for the best performing groups. Details on the leaderboard will be coming in the next few days.

8 Grading

Both the CDN servers and clients are distributed around the world. We will primarily grade you on the performance of your DNS resolutions when used to fetch content. We will use the driver to request a subset of the documents in the servers from all clients, and we will look at the average response time as reported by the clients. You should do better than a DNS server that returns a random edge server for each request, and you should do better than a DNS server that always returns the same edge server for all requests. The basic functionality will be using IP geolocation to return answers, and extended functionality will be anything that performs better than geolocation.

We will also inspect your code to make sure you are doing things dynamically, and are not pre-computing responses. We will use a different set of clients for grading the the ones you currently have. Your DNS server may assume that a file called `client-list.csv` will be given.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS168 document by filling out the anonymous feedback form:

`https://piazza.com/brown/fall2019/csci1680`.