Comp 4300 - Project Report

Group (sole): Ryan Loeppky

GitHub: Rloeppky/Comp-4300-Project (github.com)

YouTube Video Demo: https://youtu.be/lj1URdC1pd0

**Summary:**

**Problem**: Which is the faster method to retrieve the address for a website? Recursive or non-recursive or a unique mothed? Expanded: What is the most effective cache system to reduce the number of contacts between endpoints? Caches tested: FIFO, LRU, and a Unique method.

**Solution**: Create a simulation that simulates different types of cache methods for both recursive, non-recursive and a unique recursive methods to test and compare the results. Results will comprise of averages on time taken, number of connections required, and rate of cache hit rates to complete a DNS request.

**DNS pathing Options**:

**Recursive**: The Front DNS calls the Second DNS, which calls the Third DNS, which finally calls the Origin DNS.

**Non-Recursive**: The Front DNS calls the Second DNS to 'get' Third DNS connection. The Front DNS then calls the Third DNS to 'get' the Origin DNS connection, which the Front DNS finally calls the Origin DNS.

**Unique Recursive**: The Front DNS calls the Second DNS, which calls the Third DNS, which finally calls the Origin DNS. (Front and Second contain a cached immutable set of the x least recently used Domain Names and calls Origin directly if one was called)

**DNS caching Options**:

**None**: A set of tests exist for the DNS Pathing options that

**FIFO**: A First-In-First-Out approach of caching DomainName to their IPs.

**LRU**: A Least recently Used approach of caching DomainNames to their IPs.

**Unique**: Front, Second, and Third DNS contain an immutable cache of the most accessed DomainNames that cache their known IPs.

**Note**: For simplicity sake, DomainName and IPs are the same, and DomainName is a range of 1-1024. Where, 1-8 on average comes up 70% of the time, 9-25 on average comes up 15% of the time, 26-88 comes up 10% of the time and 89-1024 comes up 5% of the time.

**Important Data Values**:

Average Connections: Lower number of connections means a lower number of hands need to respond to a request.

Average Time: Lower number time taken means a shorter waiting period for any user wanting to get to a given website.

Percent Front Cache Hit: A higher percentage means that the Front DNS cache contains the domainName and responds directly to the request. It has a connection of 1 and a time range of 200 + (0-40). Note: Front DNS has a delay time of 200ms, and a vary time of 0-(200/5)ms.

Percent Second Cache Hit: A higher percentage means that the Second DNS cache contains the domainName and returns IP (DomainName) with a cache hit success back to Front DNS. It shortens the connections to 3 and has a time in the range of 900 + (0-180). Note: Second DNS has a delay time of 500ms, and a vary time of 0-(500/5)ms. Also that upon return to Front, delay and vary is added again. 200+200+500 + (0-40) + (0-40) + (0-100) == 900 + (0-180).

Percent Third Cache Hit: A higher percentage means that the Third DNS same as Second with longer ms delays.

**Results**:

Note: Data results and graphs are stored in the execl file on GitHub, in the root folder, called Datagraphs.xlsx

There are a total of twelve tests ((Recur, Non-Recur, Unique) X (None, FIFO, LRU, Unique)). Each test results were from a pool of 100 executions.

**Average Connections Winner**: Unique Recursive: Unique cache, completed in the smallest number of average connections over the span of 100 tests. This is likely due to the cache being immutable and based on the most likely requests. Also, unique recursive would shorten the number of required connections on the least likely requests.

**Average Time Winner**: Unique Recursive: Unique cache, completed in the smallest average time, which is no large surprise as it also had the smallest number of average connections. These two are directly correlated and are tightly bound to each other based on results.

**Percent Front Hit Winner**: Unique Recursive: Unique cache completed 100 tests with an 80% hit rate for the front cache. The Unique cache ended up being the fastest on average as well. Keeping a immutable cache of the most likely DomainNames can and in this case be the most efficient.

**Percent Second Hit Winner**: Non & Recursive: LRU cache completed 100 tests with an 55% hit rate for the second cache. The LRU cache ended up being the fastest on average as well for the second DNS. This is likely due to the spread of options for the second batch of most likely DomainNames.

**Percent Third Hit Winner**: Non-Recursive: Unique cache completed 100 tests with an 72% hit rate for the Third cache. The Unique cache ended up being the fastest on average as well. Keeping a immutable cache of the third batch of most likely DomainNames ended up being the most efficient.

**Technical components:**

My project directly works off the ERN tech-stack.

- Express for the API endpoints and backend,
- React for the visual components on the front end,
- and Node.js for the typescript that weaves the project together.
- Other techs used would include GitHub as my repository, and other install dependencies in my package.json files that get installed by npm.
- Npm is the primary tool that is used to manage and maintain dependencies.

**Instructions:**

Note: You may need npm installed on you machine to ensure that npm install and npm start work as intended. Node.js may also be a prerequisite, but I am not sure. Personally, I have npm 10.2.4 installed on my machine. I don't believe there are any other prerequisites needed to run this code, though I have had my laptop for some time and am not sure.

- First, clone my repository onto your machine following GitHub's instructions: Cloning a repository - GitHub Docs
- Second, in either a command line or in a terminal in an editor of your choice use the commands to start the Front-End-Client: (Make sure to be in the root directory of my project, again use cd to reach the root folder if needed)
  'cd .\Front-End-Client\'
  and then run:
  'npm install'
  and lastly run:
  'npm start'
  to start the Front-End-Client.
- Thirdly, in either a command line or in a terminal in an editor of your choice use the commands to start the Back-End-Servers: (Make sure to be in the root directory of my project, again use cd to reach the root folder if needed)
  'cd .\Back-End-Servers\'
  and then run:
  'npm install'
  and lastly run:
  'npm start'
  to start the Back-End-Servers.

I personally recommend using VS Code to open the project and run it as it is what I am using.

All dependencies have been added to package.json and should install with ‘npm install’