University of Victoria

Department of Mathematics & Statistics

The Transition from MaxMind to MySQL Database for HEPRC's Shoal Project

In partial fulfillment

of the requirements of the Math & Stats Co-op Program

Summer 2021

Work Term #1

By

Kathryn Dolby

Performed at: UVic HEPRC, Victoria

Supervisor(s): Colson Driemel, Research Assistant

Job title: Cloud Software Developer

Confidentiality notice (Please select all that apply)

Students - Please confirm the confidentiality status of the report with your employer!

☐ This report **is** confidential. Do not share for any purpose other than evaluation and record keeping.

☒ This report is **not** confidential

☒ This report **may** be shared with other co-op students

☐ This report **may not** be shared with other co-op students

Student Signature: _____

Table of Contents

1. Introduction

The main purpose of my project was to convert the database used in HEPRC's Shoal project[1] for linking IP addresses to geographic locations (country, city, coordinates, etc.). For the purposes of this project, the relevant part of Shoal takes an IP address, finds its location, then matches an active squid which is closest to that location. The database used for the location data was to be converted from a MaxMind database to MySQL (Structured Query Language). The reason for this change was a potential increase in efficiency when accessing the IP database as well as to have flexibility with the database by having the ability to make direct changes. Modifications to a MaxMind geoip database (also known as MMDB) aren't possible, because of the way the information in the database files are stored. Similar to most other databases, to be

space efficient, the file itself contains data stored in binary. This is unreadable to the human eye and can only be interpreted by MaxMind's associated Application Programming Interface[2] (API). MaxMind's API does not contain a way to modify the entries of an MMDB file, thus, transitioning to a different database is the only way to gain the freedom to make modifications.

Both the MaxMind API along with an MMDB file are referenced by Python code in the Shoal project. My end goal was to have Shoal's IP location database use MySQL instead, and to have an API that accommodates this change. It is ideal that no major changes must be made throughout the code in Shoal after the transition, so that wherever the database is accessed or the API is used, the behaviour is the same as MaxMind's API. This would allow for a smooth transition.

2. Discussion

2.1. Converting the Data

Shoal already had a complete database file in MMDB format. My first step was to extract this data and change it into a form that can be understood by MySQL, so that it could eventually be input into tables within a MySQL database. The easiest way to accomplish this was by converting the MMDB file to a CSV file, which could then be read by MySQL.

Fortunately, there exists such a conversion online [3]. The code generates all IPv4 and IPv6 addresses, checking one-by-one for the contents associated with each address in the given MMDB file, using MaxMind's API to interact with it, then outputs the address and associated data to a CSV file if it exists in the MaxMind database. I had to modify the code to work as a

regular python script and accommodate for my database file having differently named fields as well as some missing fields. I also made a few larger changes to this code.

Firstly, I converted the IP address ranges from one value to two values: the start and end of the range as integers. This made the future MySQL queries much simpler. Since the code generated an address, then checked for it, then wrote to the CSV file, it created many repeat entries which were each small IP address ranges. Originally, in the MMDB file, these many small entries were one big range of IP addresses associated to a geolocation. I modified the code to scan through each chunk of 10,000 entries, sort them, and condense them to one IP range per entry, in O(nlogn) time, before writing the new group of less than 10,000 entries to the CSV file. This saved a lot of space, bringing sixteen million entries down to nine million entries. Although, there were potential range combinations missed on the boundary entries of these chunks, approximately 3200 in the worst case. Having a constant chunk size was worth missing some combinations, because the program runs faster and does not have to store all original sixteen million entries at once in the machine's RAM.

Another change I made was in the way the code wrote the database entries to the CSV file. Instead of having one large file with all entries, I adjusted it so that there would be two smaller files, one containing only IPv4 addresses, and the other containing only IPv6 addresses. I decided to do this within the conversion step because it did not add to the runtime of the conversion process. Performing the IPv4 and IPv6 separation in a separate script would have added to the total set up time. This separation was beneficial because it would prove to be quicker to add these separate CSV files into two separate tables in my MySQL schema. These two tables would also allow for faster query times.

According to the site from which the MMDB file was downloaded[4], there were 6,097,870 data entries. After running my conversion code, there were 9,220,980 lines/entries in total for the CSV files: 3,164,119 IPv4 entries and 6,056,861 IPv6 entries. None of these entries were duplicates. The cause of these extra entries is unknown, and it would not have a large effect on the overall speed of queries. Using the command line to do the combinations on a fully converted CSV file instead of during the conversion process, the same outcome was found. The cause of this issue is unknown, but was likely caused by the way the conversion code picks IP addresses before checking if they are in the database.

At the end of this file conversion, I compared the outputs of sixteen MMDB queries (Appendix, Table 1) against the *grep* command on the CSV files to confirm that the conversion worked as expected. The next step was to integrate the CSV files of geodata with a MySQL database.

2.2. MySQL Database

Creating the general schema for my MySQL database consisted of preparing columns corresponding to each geographical data column in the CSV files. MySQL offers many different data types, and I used those that needed the least space while storing the data correctly. For the string fields, *varchar* was the best option, and for the latitude and longitude, *decimal* saved space, since I knew there were no more than five digits after the decimal. For the start and end IP integer values, the data type used for each table was different. For the IPv4 table, the integer values were small enough to use an unsigned int. However, for the IPv6 table, the integer values

were much larger and had to be put into the decimal data type with at most thirty-nine digits. The CSV files could then be easily input into their respective tables.

To save time in testing, I used only the "head" and "tail" of the IPv4 file, ten entries from the beginning and ten entries from the end. I then input the entirety of the CSV file and compared the same results from the first ten MMDB tests (Appendix, Table 1) to a basic MySQL query (Appendix, Table 2). The results matched as expected.

The majority of testing for this project came from choosing the ideal database setup. I tested two different database engines with no indexing, indexing on the start and end IP addresses, as well as with primary keys. On the InnoDB engine, the database sizes were 315.8 MB, 411.0 MB, and 345.9 MB, respectively. The MyISAM engine gave smaller databases sizes, with 234.4 MB, 295.9 MB, and 277.9 MB. I then tested all six of these databases with multiple different MySQL queries (Appendix, Tables 3 & 4). The MyISAM engine proved to be much faster for queries in general than MariaDB's default InnoDB engine.

Where x is the integer value of the IP address, the tested queries were:

Query 1: *select * from ipv4 where start_ip <= x and end_ip >= x;*

Query 2: *select * from ipv4 where x between start_ip and end_ip;*

Query 3: *select * from (select * from ipv4 where start_ip<= x) where end_ip>=x;*

Query 4: *select * from ipv4 where end_ip >= x order by end_ip asc limit 1;*

Overall, using the MyISAM engine and indexing with the "end_ip" values resulted in the quickest time with query #4. This query is the fastest because it uses only "end_ip" and does not consider "start_ip", then sorts the query results, and outputs the first result. Each query using this

method took less than one hundredth of a second. However, there is one issue with this method. The given database only contains IP addresses in the range 1.0.0.0 – 223.255.255.255 for IPv4 and 2000:: – feff:ffff:ffff:ffff:ffff:ffff:ffff:ffff for IPv6. Thus, there is a possibility that a queried IP address does not exist in the database. Even if a given IP address is not in the database, an incorrect result will be found and returned. This was fixed by checking the validity of the result after calling the query within the API created in the next step.

Direct time comparisons for all MySQL queries were made against running MMDB queries (Appendix, Table 5 with MMDB times). Shell scripts were created to run 101 randomly selected IP address queries for both the MMDB and the MySQL databases (Appendix, Table 6) to confirm that the MySQL query times were faster on average.

2.3. The API

Before creating the layout of the new API to be used to interact with the MySQL database, I wrote the Python code that would call a query, access the database, and output a result. This would be called when initializing an instance of the Reader class. I again compared this output with the expected data from Table 1 and documented the time it took for each run of the code with different IP addresses as input. These times are significantly slower than the direct queries, because every time the script is called, the entire database is read before being queried. This happens because the database is read when initializing an instance of the Reader class, which is created for each individual query. The original MaxMind API has the same class layout and, also, reads the database for each Reader instance[5], so the issue of this overhead reading time has always been present in the Shoal project.

The format of the classes for the API were as follows, where "Reader" is a class that contains the class Geodata as an attribute:

- Reader
    - Geodata
        - City
        - Country
        - Continent
        - Location
        - Postal
        - Subdivisions
            - Subdivision

These classes were tested to ensure that the names used with MMDB would correspond to the same names with the same associated values in the new database. This allows for minimal change to the existing database reader call in the Shoal code[6]. The only change was the reference to the new API instead of MaxMind's GeoIP2. All other references to attributes remained unchanged between MaxMind and the new implementation.

2.4. Server Testing

A server for the database, MariaDB, was installed and set up. After writing the API in Python, very minimal changes must be made to the existing code in Shoal's utilities.py. The new API was imported, and one line of code was changed to access this new API. Other necessary files were copied over. The test server webpage correctly displayed the geographical data of the active squids (Appendix, Figure 1) and the nearest squid (Appendix, Figure 2).

Although, some issues became apparent on the webpage [7]. Some decimal values for longitude and latitude were too long, so this was fixed in the MMDB to CSV conversion to

round values to the nearest hundred-thousandth. Also, accented letters in the CSV were being displayed as strange symbols in the MySQL database. The database and tables needed charset and collation changes to match the Unicode used from the MaxMind database, utf8mb4 for the charset and utf8mb4_unicode_ci for the collation.

One unexpected problem occurred where a squid on the webpage displayed a city name in Germany that does not exist. This was found to be an error in the original MMDB file. Because of the new MySQL database, this was an easy fix. The city name for every similar entry was changed from the non-existent "Arching" to "Garching bei München" with only one line of MySQL for each of the two tables[8]. This showcases a major advantage to the new database. Now it is possible to change or remove existing entries, or add new entries manually as needed.

2.5. Automation

Finally, the entire process was automated. With one shell script[9], the setup for the MySQL database only required one call to run the script. Within the script, the MMDB file was downloaded and converted to CSV, the database and tables were set up according to the MySQL schema I created, the CSV files were input into the database, and any modifications to the database were made. The site from which the MMDB file is downloaded releases a new updated database monthly. The download portion of the script could be modified to download the most recent file at any given time, or update the database periodically based on need. Due to the MMDB to CSV conversion process, the script takes approximately two hours to run from download to completion.

The Shoal Ansible[10] sets up the project from scratch on a machine. To the existing ansible, I added the installation and set up of the database server, MariaDB, the Python package installations required for my new code, and the call to run the script so that no part of the database transition was required to be done manually. I tested the Shoal ansible on a clean virtual machine on an Openstack cloud, and eventually it ran smoothly from start to finish. Now, whenever a new Shoal server is set up, it will be using a MySQL database instead of MaxMind.

3. Conclusions

Using a MySQL database as opposed to a MaxMind database is beneficial in the case of Shoal and potentially other projects as well. The average query times using MaxMind were: 0.25s for an IPv4 address, 0.081s for an IPv6 address, 7.238s for 101 random sequential queries, and 2.040s for 101 random queries in parallel. MySQL proved to be quicker, for its average query times were: 0.072s for an IPv4 address, 0.073s for an IPv6 address, 6.759s for 101 random sequential queries, and 1.737s for 101 random queries in parallel. MySQL and MMDB have similar times in the fastest case, but MySQL is more consistent and maintains the short query times between 0.06 - 0.08 s, whereas MMDB can take almost one second. All these times include the creation of a "Reader" object which reads the entire database for each query that is called.

Currently, the creation of the Reader object takes much more time than the query itself. In order to improve the efficiency of the database querying process, we would need to eliminate the need for the Reader object to be created with every query. Instead, the Reader could be recreated after a certain number of queries or after a specified amount of time. For now, the

MySQL database is still advantageous because of the ease with which the entries in the database can be modified.

4. Acknowledgments

Thank you to Colson Driemel, Marcus Ebert, and Catherine Meng for helping me numerous times when I ran into errors, and for giving me direction with the project when I was unsure of how to proceed. They were also very patient when explaining ideas to me and giving tips for thorough testing.

5. References

1. HEPRC. *Shoal GitHub*, github.com/hep-gc/shoal.
2. MaxMind. *GeoIP2 Python API*, https://geoip2.readthedocs.io/en/latest/.
3. A. Ovidiu. *MaxMind mmdb to csv converter*, https://github.com/ovimihai/MaxMind-python-mmdb-to-csv-converter/blob/main/mmdb_converter.ipynb
4. dbip. *IP to City Lite database*, https://db-ip.com/db/download/ip-to-city-lite.
5. MaxMind. *GeoIP2-python database.py*, https://github.com/maxmind/GeoIP2-python/blob/4e475fb1344cc95d05eff4772994ea2924342af3/geoip2/database.py.
6. C. Driemel, et al. *Shoal-sql utilities.*py, https://github.com/hep-gc/shoal/blob/shoal-sql/shoal-server/shoal_server/utilities.py, line 53.
7. HEPRC. *Shoal Test Server*, http://206.12.154.84/.
8. K. Dolby. *Geodata Updates*, https://github.com/hep-gc/shoal/blob/master/shoal-server/shoal_server/setup-db/geodata_updates.sql.
9. K. Dolby. *Setup MySQL database script*, https://github.com/hep-gc/shoal/blob/master/shoal-server/shoal_server/setup-db/setup_db.sh.
10. HEPRC. *Ansible system for shoal*, https://github.com/hep-gc/ansible-systems/tree/master/heprc/staticvms/roles/shoal.

6. Appendix

Table 1. MaxMind Tests

Sixteen test IP addresses and their output from the MaxMind database for comparison to CSV files and MySQL queries. There are ten IPv4 addresses and six IPv6 addresses.

Test IP Address	MMDB Output
206.12.154.13	NA North America CA Canada Victoria (Harris Green) British Columbia 48.4244 -123.36
1.0.0.125	OC Oceania AU Australia South Brisbane Queensland -27.4767 153.017
76.255.255.254	NA North America US United States Chicago Illinois 41.8781 -87.6298
62.18.255.1	EU Europe IT Italy Rome Lazio 41.818 12.4148
160.167.53.236	AF Africa MA Morocco Rabat (Agdal) Rabat-Salé-Kénitra 33.9977 -6.84793
208.64.0.240	NA North America US United States Toronto Ohio 40.4642 -80.6009
59.118.72.238	AS Asia TW Taiwan Taipei Taiwan 25.033 121.565
229.181.76.76	geoip2.errors.AddressNotFoundError: The address 229.181.76.76 is not in the database.
49.115.176.86	AS Asia CN China Ewirgol Xinjiang 42.5246 87.53960000000001
69.208.83.7	NA North America US United States Kalamazoo Michigan 42.2917 -85.5872
2001:4860:4860::8888	NA North America CA Canada Montreal Quebec 45.5017 -73.5673
2600:fc4f:8b66::	NA North America US United States Chantilly Virginia 38.9097 -77.4524
af71:e5d0:45ff:229b:e4fa:1ac2:bdeb:d8a5	EU Europe CH Switzerland Murten/Morat Fribourg 46.9219 7.16595
3fff:ffff:ffff:ffff:ffff:e4b7:14a8:2003	EU Europe FR France Paris Île-de-France 48.8566 2.35222
2c0f:f020::f:3656	AF Africa ZA South Africa Pretoria Gauteng -25.8521 28.1939
0123:4567:89ab::	geoip2.errors.AddressNotFoundError: The address 0123:4567:89ab:: is not in the database.

Table 2. MySQL Tests

Ten IPv4 addresses and their MySQL query results to compare against the results of Table 1.

IP Address Queried	start_ip	end_ip	continent_code	continent
206.12.154.13	3456932352	3456932863	NA	North America
1.0.0.125	16777216	16777471	OC	Oceania
76.255.255.254	1291845384	1291845631	NA	North America

62.18.255.1	1041432000	1041432335	EU	Europe
160.167.53.236	2695298048	2695331839	AF	Africa
208.64.0.240	3493855232	3493855743		North America
59.118.72.238	997605376	997617151	AS	Asia
229.181.76.76	Empty set			
49.115.176.86	829423616	829734911	AS	Asia
69.208.83.7	1171275776	1171283967		North America

country_code	country	city	region	latititude	longitude
CA	Canada	Victoria (Harris Green)	British Columbia	48.4244	-123.36
AU	Australia	South Brisbane	Queensland	-27.4767	153.017
US	United States	Chicago	Illinois	41.8781	-87.6298
IT	Italy	Rome	Lazio	41.818	12.4148
MA	Morocco	Rabat (Agdal)	Rabat-SalÃ©-KÃ©nitra	33.9977	-6.84793
US	United States	Toronto	Ohio	40.4642	-80.6009
TW	Taiwan	Taipei	Taiwan	25.033	121.565
CN	China	Ewirgol	Xinjiang	42.5246	87.5396
US	United States	Kalamazoo	Michigan	42.2917	-85.5872

Table 3. MySQL Query Times on InnoDB

Ten IPv4 test IP addresses queried in a MySQL database using the InnoDB engine. The databases tested were either indexed, had a primary key, or neither. Times are rounded to the nearest hundredth.

Indexed or Primary Key?	Test IP Address	Query 1 (s)	Query 2 (s)	Query 3 (s)
No	206.12.154.13	2.26	1.97	1.98
No	1.0.0.125	1.95	1.91	2.07
No	76.255.255.254	1.96	1.98	2.08
No	62.18.255.1	2.02	2	2.06
No	160.167.53.236	2.02	2	2.02
No	208.64.0.240	1.91	2.02	2.02
No	59.118.72.238	1.89	2.01	2.08
No	229.181.76.76	1.94	1.97	2.01
No	49.115.176.86	1.88	1.96	2.05
No	69.208.83.7	1.83	1.93	2.08
Indexed	206.12.154.13	1.98	1.95	1.94

Indexed	1.0.0.125	0	0	0
Indexed	76.255.255.254	1.9	1.98	2.02
Indexed	62.18.255.1	1.92	1.93	2.03
Indexed	160.167.53.236	1.94	1.93	1.79
Indexed	208.64.0.240	2.48	1.89	2
Indexed	59.118.72.238	1.94	1.82	2.1
Indexed	229.181.76.76	0.01	0	0
Indexed	49.115.176.86	2.09	1.73	2.1
Indexed	69.208.83.7	2.06	1.72	2.04
Primary Key	206.12.154.13	2.89	2.13	2.15
Primary Key	1.0.0.125	0	0	0
Primary Key	76.255.255.254	0.7	0.76	0.67
Primary Key	62.18.255.1	0.27	0.26	0.25
Primary Key	160.167.53.236	1.41	1.33	1.32
Primary Key	208.64.0.240	2.49	2.16	2.23
Primary Key	59.118.72.238	0.38	0.37	0.36
Primary Key	229.181.76.76	2.32	2.3	2.27
Primary Key	49.115.176.86	0.31	0.31	0.29
Primary Key	69.208.83.7	0.48	0.46	0.46

Table 4. MySQL Query Times on MyISAM

Ten IPv4 test IP addresses queried in a MySQL database using the MyISAM engine. The databases tested were either indexed, had a primary key, or neither. Times are rounded to the nearest hundredth.

Indexed or Primary Key?	Test IP Address	Query 1 (s)	Query 2 (s)	Query 3 (s)	Query 4 (s)
No	206.12.154.13	0.64	0.55	0.6	0.56
No	1.0.0.125	0.58	0.55	0.65	0.8
No	76.255.255.254	0.6	0.55	0.63	0.73
No	62.18.255.1	0.59	0.55	0.63	0.75
No	160.167.53.236	0.61	0.55	0.6	0.65
No	208.64.0.240	0.64	0.56	0.59	0.57
No	59.118.72.238	0.58	0.56	0.64	0.75
No	229.181.76.76	0.64	0.56	0.58	0.55
No	49.115.176.86	0.58	0.55	0.64	0.74
No	69.208.83.7	0.59	0.55	0.64	0.72
Indexed	206.12.154.13	0.45	0.41	0.44	0
Indexed	1.0.0.125	0	0	0	0
Indexed	76.255.255.254	0.59	0.56	0.63	0
Indexed	62.18.255.1	0.69	0.66	0.7	0
Indexed	160.167.53.236	0.6	0.57	0.61	0.01
Indexed	208.64.0.240	0.38	0.35	0.37	0
Indexed	59.118.72.238	0.64	0.59	0.62	0

Indexed	229.181.76.76	0	0	0	0
Indexed	49.115.176.86	0.54	0.5	0.53	0
Indexed	69.208.83.7	0.59	0.55	0.63	0
Primary Key	206.12.154.13	0.62	0.55	0.59	0
Primary Key	1.0.0.125	0	0	0	0
Primary Key	76.255.255.254	0.63	0.55	0.65	0
Primary Key	62.18.255.1	0.07	0.05	0.07	0
Primary Key	160.167.53.236	0.63	0.56	0.61	0
Primary Key	208.64.0.240	0.63	0.56	0.59	0
Primary Key	59.118.72.238	0.06	0.05	0.06	0
Primary Key	229.181.76.76	0.64	0.56	0.58	0
Primary Key	49.115.176.86	0.05	0.04	0.04	0
Primary Key	69.208.83.7	0.6	0.57	0.63	0

Table 5. MaxMind Query Times

Sixteen test IP addresses and the amount of time in seconds that it took to find the associated data to each IP address using the MaxMind database. Tests where the IP address is not in the database are included.

Test IP Address	MMDB Query Time (s)
206.12.154.13	0.063
1.0.0.125	0.064
76.255.255.254	0.064
62.18.255.1	0.063
160.167.53.236	0.23
208.64.0.240	0.756
59.118.72.238	0.07
229.181.76.76	0.463
49.115.176.86	0.073
69.208.83.7	0.479
2001:4860:4860::8888	0.084
2600:fc4f:8b66::	0.079
af71:e5d0:45ff:229b:e4fa:1ac2:bdeb:d8a5	0.079
3fff:ffff:ffff:ffff:ffff:e4b7:14a8:2003	0.079
2c0f:f020::f:3656	0.075
0123:4567:89ab::	0.088

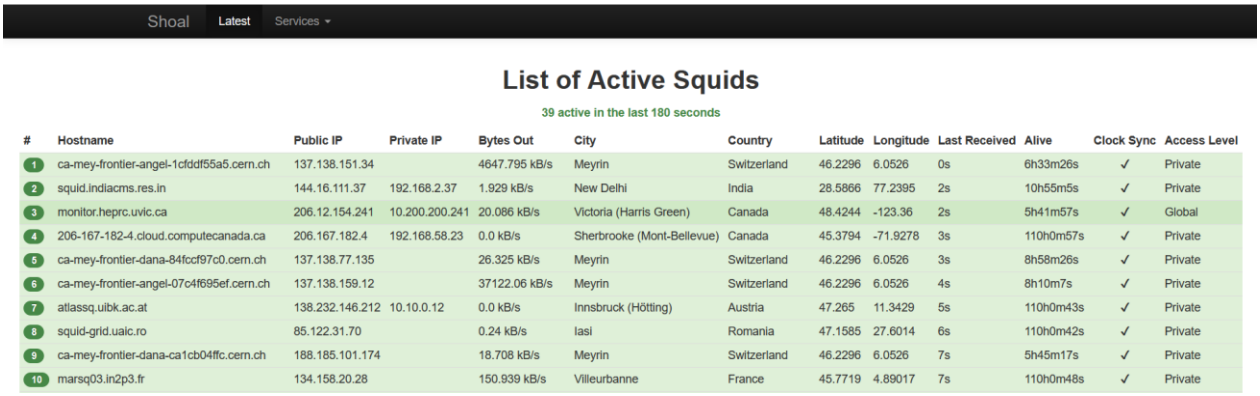
Table 6. Comparison Times between MySQL and MaxMind

Times for 101 queries ran sequentially and in parallel, rounded to the nearest thousandth.

Query Type	Sequential (s)	Parallel (s)
MySQL	0.637	0.406
MMDB via Python Script	7.238	2.04
MySQL via Python Script	6.456	1.786

Figure 1. Webpage View of Squids

A view of the first ten squids on a test server and their associated geographical data found by the new MySQL database.



Shoal Latest Services

List of Active Squids

39 active in the last 180 seconds

#	Hostname	Public IP	Private IP	Bytes Out	City	Country	Latitude	Longitude	Last Received	Alive	Clock Sync	Access Level
1	ca-mey-frontier-angel-1cfddf55a5.cern.ch	137.138.151.34		4647.795 kB/s	Meyrin	Switzerland	46.2296	6.0526	0s	6h33m26s	✓	Private
2	squid.indiacms.res.in	144.16.111.37	192.168.2.37	1.929 kB/s	New Delhi	India	28.5866	77.2395	2s	10h55m5s	✓	Private
3	monitor.heprc.uvic.ca	206.12.154.241	10.200.200.241	20.086 kB/s	Victoria (Harris Green)	Canada	48.4244	-123.36	2s	5h41m57s	✓	Global
4	206-167-182-4.cloud.computecanada.ca	206.167.182.4	192.168.58.23	0.0 kB/s	Sherbrooke (Mont-Bellevue)	Canada	45.3794	-71.9278	3s	110h0m57s	✓	Private
5	ca-mey-frontier-dana-84fccf97c0.cern.ch	137.138.77.135		26.325 kB/s	Meyrin	Switzerland	46.2296	6.0526	3s	8h58m26s	✓	Private
6	ca-mey-frontier-angel-07c4f695ef.cern.ch	137.138.159.12		37122.06 kB/s	Meyrin	Switzerland	46.2296	6.0526	4s	8h10m7s	✓	Private
7	atlassq.uibk.ac.at	138.232.146.212	10.10.0.12	0.0 kB/s	Innsbruck (Hötting)	Austria	47.265	11.3429	5s	110h0m43s	✓	Private
8	squid-grid.uaic.ro	85.122.31.70		0.24 kB/s	Iasi	Romania	47.1585	27.6014	6s	110h0m42s	✓	Private
9	ca-mey-frontier-dana-ca1cb04ffc.cern.ch	188.185.101.174		18.708 kB/s	Meyrin	Switzerland	46.2296	6.0526	7s	5h45m17s	✓	Private
10	marsq03.in2p3.fr	134.158.20.28		150.939 kB/s	Villeurbanne	France	45.7719	4.89017	7s	110h0m48s	✓	Private

Figure 2. Webpage View of Nearest Squid

The “nearest” page on a test server, correctly displaying the closest active squid to my machine’s location.