Mapserver Load Testing Results

Isaac Thiessen, 2019-11-15

EDITORS NOTE:

This report was originally provided to NFIS with confidential information, aspects of the report have been modified since the report was initially published.

I have included some of the scripts that I used to generate the test data but the process is slightly different from what is included in this paper, however the files themselves are documented and will be included. Also most of the text files contained sensitive information about our systems, so they have been omitted for security reasons.

Table of Contents

Summary

This study measures the performance of two implementations of Mapserver to see if it is worth the time and effort to switch to a Dockerized Mapserver.

This study has found that the CampToCamp Mapserver outperforms the compiled Mapserver by a large margin. CampToCamp is able to handle more requests per minute while also using far less computing power.

Requirements

Find the performance impact of running Docker + CampToCamp Mapserver versus our current infrastructure and compile Mapserver with the newest version of every library while documenting the whole process.

Test Procedure

Jmeter runs 3000 total connections over 100 individual "users" in ~100 seconds. Although the end result is the same, some of the specifics of this process have changed. Feel free to check out the scripts directory for full information on the process.

Procedure:

1. On client: Run scripts/run_jmeter_test.sh
2. SSH into server, runs "dstat" to track memory and cpu usage for 110 seconds saving data to a file in /tmp/
3. Exectute jMeter testplan for specific host (this makes the 3000 requests)
4. Download the output from dstat to the client for analysis
5. Run scripts/analyze_report.py on the newly gathered data. This generates a nice graph of the system usage over the duration of the jMeter test.
6. Create a directory in "reports/h<host_number>" for the test and put the following files in it: jmeter output, dstat output, analyze_dstat.py output and graph, jmeter graph

Hosts

These tests were performed on 2 hosts. One that matches our current infrastructure and one with the CampToCamp Docker Mapserver.

Both servers have:

- Same specs (1GB RAM, 2GB swap, 2 CPUs)
- Hosted on same hypervisor (Virtualbox)
- Same data (exactly enough to perform tests on all 3 cgi files)
- Same mapfiles
- Same operating system: CentOS 7, kernel: 3.10.0-957.21.3.el7.x86_64
- Built with Vagrant, with Ansible applied the "nfis.apache" and "nfis.common" roles

Host 1:

This host is a mirror of the current infrastructure but with newer versions of the libraries, and Mapserver.

To summarize the lifecycle of a Mapserver request:

- Apache 2.4 takes request
- Apache executes cgi file
- cgi file sets mapfile and executes Mapserver with the specified query parameters from request
- Mapserver outputs response to stdout (? I think) which is read by Apache
- Apache responds to user.

It took me approximately a week to set up this server, most of the time was spent compiling Mapserver and its dependencies and dealing with the issues involved with that. Its worth mentioning I didn't start this project with a lot of experience compiling C libraries so that may have contributed to the length of time that it took to set up this server.

Host 2:

This host is an example of the proposed architecture, it involves running Apache on the host (which in production would serve the files for the website and everything else) and running my implementation of the Docker CampToCamp Mapserver image.

In this implementation, the Mapserver is running locally and is not accessible outside of Host 2. All requests to it are internal.

The Docker image does not contain any data, all data is mapped into the image through volumes.

To summarize the lifecycle of a Mapserver request:

- Apache 2.4 takes request, if its for the location /mapserver/cgi-bin/ it is reverse proxied to the Docker Mapserver
- The Docker Mapserver handles the request, and sends the response to Apache
- Apache responds to the user.

It took me approximately one hour to configure this server.

Mapserver versions on each host:

This section is the output of the command “mapserv -v” for each host of interest.

Here I will point out that CampToCamp has version 7.5dev, which is not yet released on Mapservers website. The most recent release is 7.4.2.

Host 1 (compiled Mapserver):

```
MapServer version 7.4.2 OUTPUT=PNG OUTPUT=JPEG SUPPORTS=PROJ SUPPORTS=AGG
SUPPORTS=FREETYPE SUPPORTS=ICONV SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER SUPPORTS=SOS_SERVER
SUPPORTS=GEOS SUPPORTS=PBF INPUT=JPEG INPUT=POSTGIS INPUT=OGR INPUT=GDAL
INPUT=SHAPEFILE
```

CampToCamp:

```
MapServer version 7.5-dev OUTPUT=PNG OUTPUT=JPEG OUTPUT=KML SUPPORTS=PROJ
SUPPORTS=AGG SUPPORTS=FREETYPE SUPPORTS=CAIRO SUPPORTS=SVG_SYMBOLS SUPPORTS=RSVG
SUPPORTS=ICONV SUPPORTS=FRIBIDI SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER SUPPORTS=SOS_SERVER
SUPPORTS=FASTCGI SUPPORTS=GEOS SUPPORTS=POINT_Z_M SUPPORTS=PBF INPUT=JPEG
INPUT=POSTGIS INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE
```

Xpres 1 (for comparison):

```
MapServer version 6.4.1 OUTPUT=PNG OUTPUT=JPEG SUPPORTS=PROJ SUPPORTS=AGG
SUPPORTS=FREETYPE SUPPORTS=ICONV SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT
SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER SUPPORTS=SOS_SERVER
SUPPORTS=GEOS INPUT=JPEG INPUT=POSTGIS INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE
```

Data

Here I will contain the data of the results. Its really important to remember when looking at these numbers is this is a worst-case-scenario type test. Although I have no idea what the traffic is like on our production servers, I would be willing to bet that this is significantly more requests per minute than we usually get.

On all graphs, the time is measured in seconds. I used Python and Matplotlib to render the graphs.

This information has been computed based on data collected from jMeter and dstat.

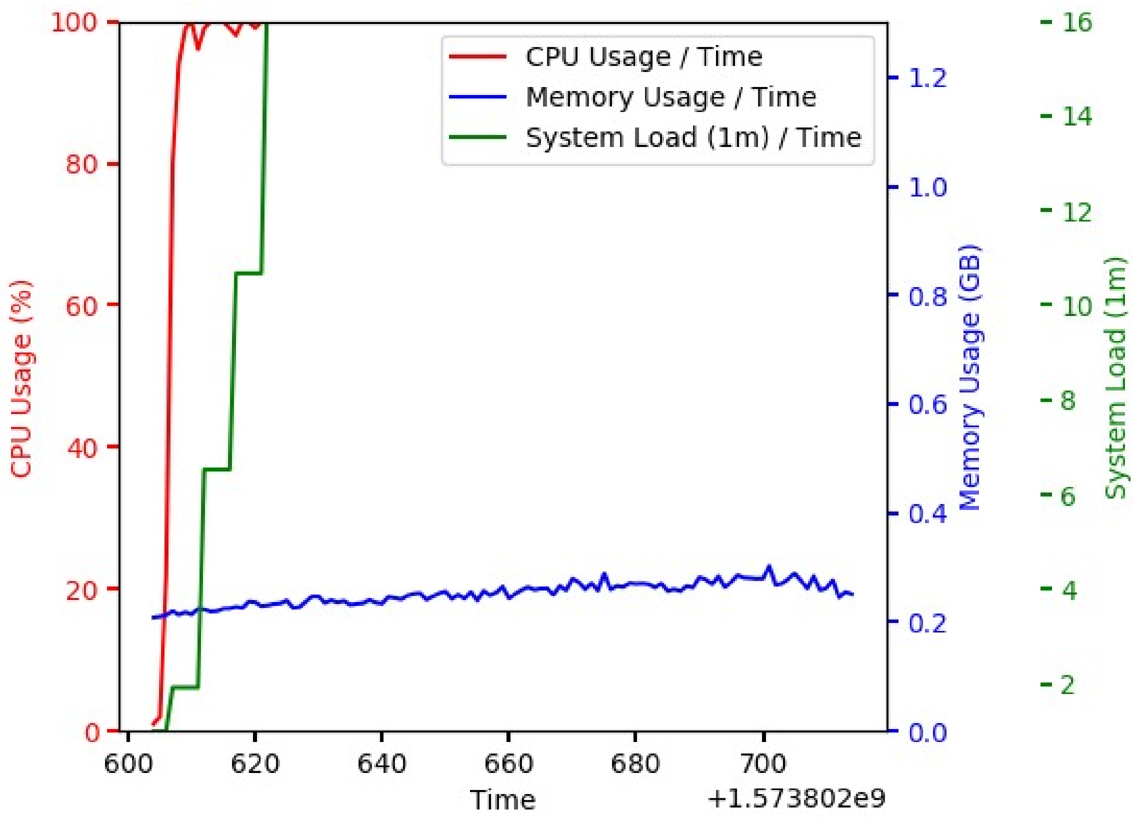
Field	Unit	Data source	Description
Throughput	requests/ minute	Jmeter csv	Average number of requests the server handled in a minute
Average Latency	ms	Jmeter csv	Average length of time for the request to complete (calculated in Libreoffice calc with the AVERAGE function)
CPU Min/Max Values	% of CPU used	Dstat csv	The minimum and maximum CPU usage during the test
CPU Average Value	% of CPU used	Dstat csv	The average CPU usage during the test
Memory Min/Max Values	GB memory used	Dstat csv	The minimum and maximum memory usage during the test
Memory Average Value	GB memory used	Dstat csv	The average memory usage during the test

I have also included graphs of Dstat info, this also includes the 1m load of the CPU during the test. Normally CPU load is measured in 1m, 5m, 15m. However this test only lasts 100 seconds so I only included the 1m value, for more information on what the CPU load is see this link:

https://www.tecmint.com/understand-linux-load-averages-and-monitor-performance/

Host1: Apache + Mapserver

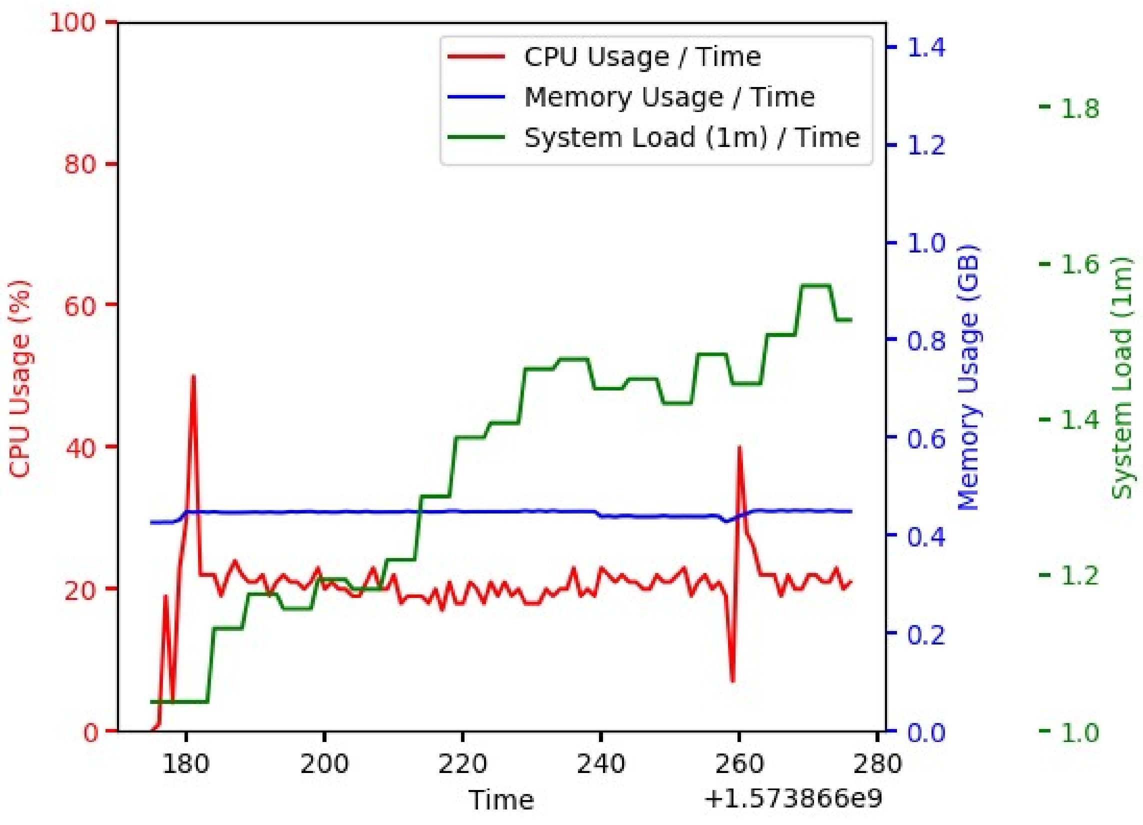
More info in under the “Host 1” header. This is a replica of the current system with newer packages.



Field	Unit	Value
Throughput	requests/minute	1589
Average Latency	ms	505.213
CPU Min/Max Values	% of CPU used	1% / 100%
CPU Average Value	% of CPU used	97.20%
Memory Min/Max Values	GB memory used	0.21GB / 0.30 GB
Memory Average Value	GB memory used	0.25 GB

Host2: Apache + CampToCamp

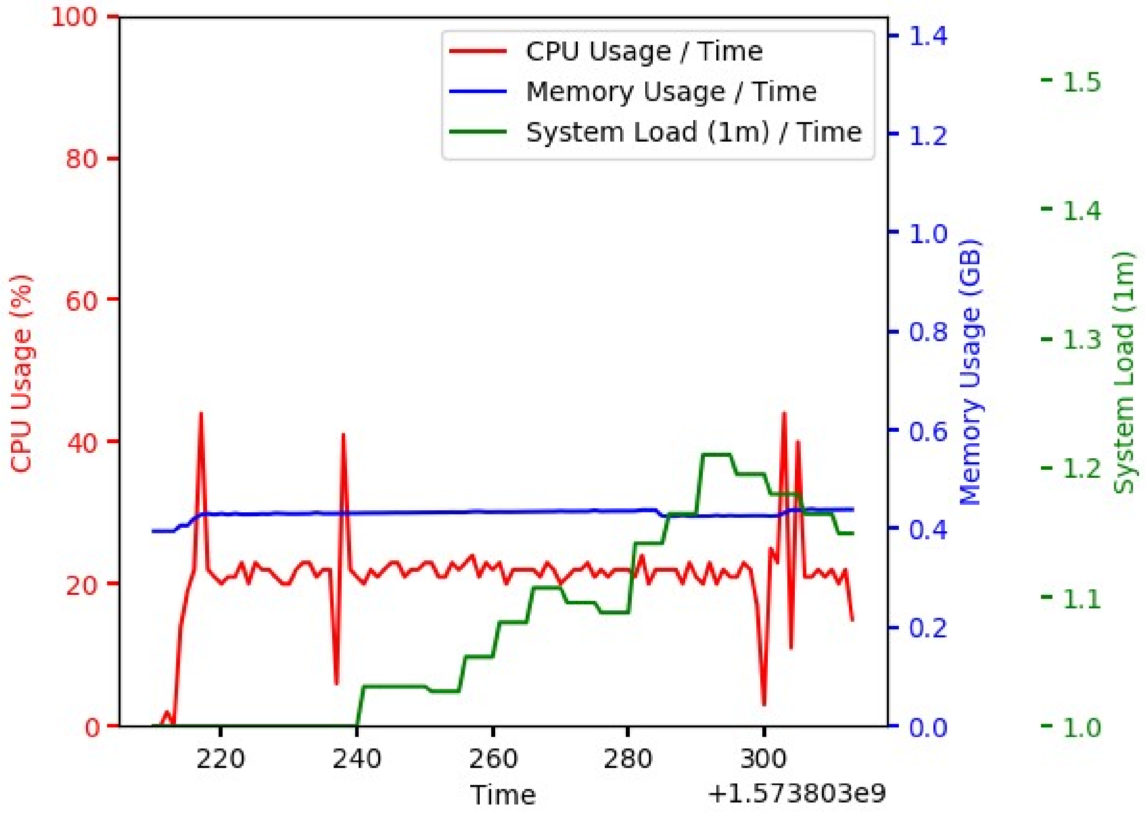
More info in under the “Host 2” header. This is the proposed infrastructure with Docker.



Field	Unit	Value
Throughput	requests/minute	1802
Average Latency	ms	35.401
CPU Min/Max Values	% of CPU used	0% / 38%
CPU Average Value	% of CPU used	18.87%
Memory Min/Max Values	GB memory used	0.42 GB / 0.45 GB
Memory Average Value	GB memory used	0.44 GB

Host2: CampToCamp only (no Apache)

I included this just out of curiosity, this test skips Apache all together and maps port 80 of the Mapserver Docker image to port 80 of the hosts, so all the requests are made directly to the Mapserver. This simulates the performance of the Mapserver as if it were on its own dedicated host. Interestingly enough it turns out it doesn't make much of a difference.

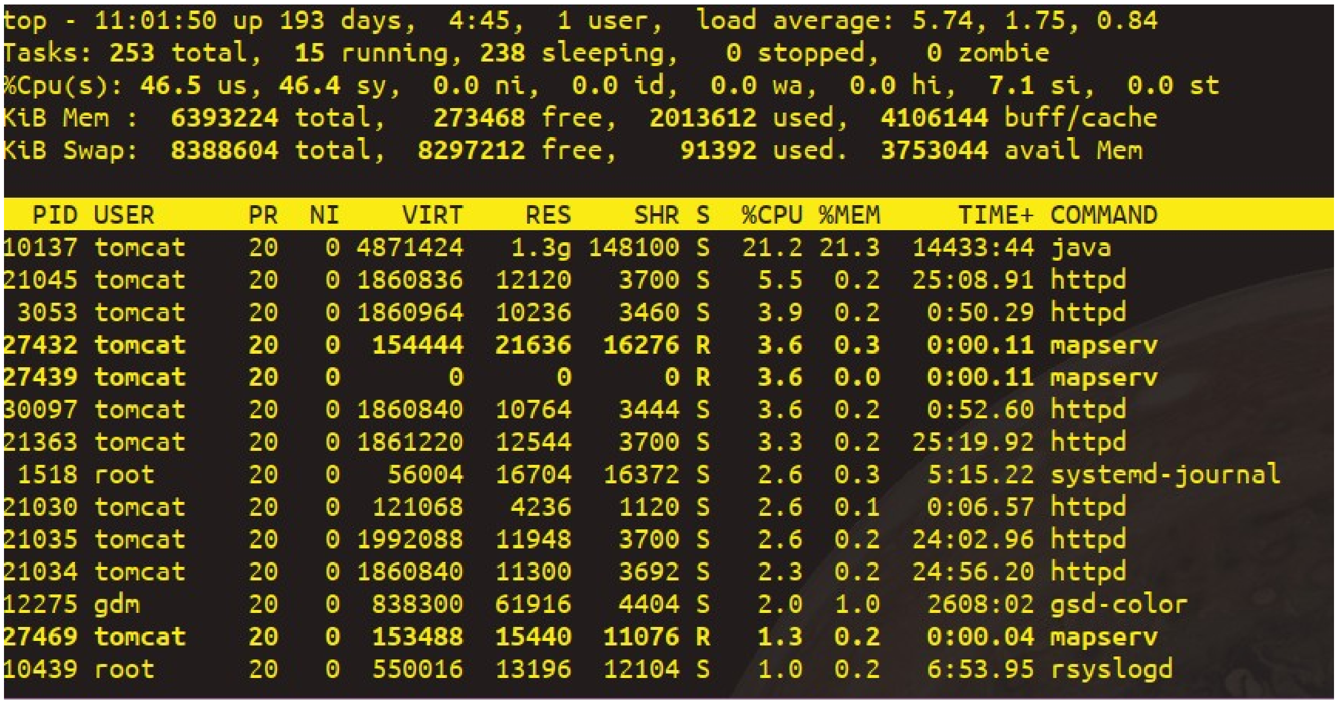


Field	Unit	Value
Throughput	requests/minute	1802
Average Latency	ms	38.829
CPU Min/Max Values	% of CPU used	0% / 44%
CPU Average Value	% of CPU used	21.09 %
Memory Min/Max Values	GB memory used	0.39 GB / 0.44 GB
Memory Average Value	GB memory used	0.43 GB

Xpres3:

This test was done for comparison to the “Host1 Apache + Mapserver” I thought something was wrong because of the exceedingly high CPU usage. However, it turns out Xpres3 behaved the same way under the same test.

I chose Xpres3 because it is less used than the other Xpres machines. I cannot provide full information of the CPU usage during the test due to the fact that I cant install dstat on Xpres. Instead I just watched the output of “top” while running the jMeter test. Here is a screenshot from during the load test:



```
top - 11:01:50 up 193 days,  4:45,  1 user,  load average: 5.74, 1.75, 0.84
Tasks: 253 total,  15 running, 238 sleeping,   0 stopped,   0 zombie
%Cpu(s): 46.5 us, 46.4 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  7.1 si,  0.0 st
KiB Mem :  6393224 total,   273468 free,  2013612 used,  4106144 buff/cache
KiB Swap:  8388604 total,  8297212 free,    91392 used.  3753044 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
10137 tomcat    20   0 4871424   1.3g 148100 S  21.2 21.3  14433:44 java
21045 tomcat    20   0 1860836  12120   3700 S   5.5  0.2  25:08.91 httpd
 3053 tomcat    20   0 1860964  10236   3460 S   3.9  0.2   0:50.29 httpd
27432 tomcat    20   0  154444  21636  16276 R   3.6  0.3   0:00.11 mapserv
27439 tomcat    20   0       0      0      0 R   3.6  0.0   0:00.11 mapserv
30097 tomcat    20   0 1860840  10764   3444 S   3.6  0.2   0:52.60 httpd
21363 tomcat    20   0 1861220  12544   3700 S   3.3  0.2  25:19.92 httpd
 1518 root      20   0   56004  16704  16372 S   2.6  0.3   5:15.22 systemd-journal
21030 tomcat    20   0  121068   4236   1120 S   2.6  0.1   0:06.57 httpd
21035 tomcat    20   0 1992088  11948   3700 S   2.6  0.2  24:02.96 httpd
21034 tomcat    20   0 1860840  11300   3692 S   2.3  0.2  24:56.20 httpd
12275 gdm       20   0  838300  61916   4404 S   2.0  1.0   2608:02 gsd-color
27469 tomcat    20   0  153488  15440  11076 R   1.3  0.2   0:00.04 mapserv
10439 root      20   0  550016  13196  12104 S   1.0  0.2   6:53.95 rsyslogd
```

The “id” attribute is the one of interest (3rd line), it represents the percentage of the CPU that is idle at this time. It stayed close to 0 for the duration of the test.

Results:

I found the results to be very unexpected. I would have thought that the current infrastructure would have performed marginally better due to not having to run Docker, The CampToCamp image, and Apache on the CampToCamp image.

Performance: Clients perspective

The CampToCamp Mapserver was able to outperform the newer Mapserver by serving an additional 213 requests per minute (see throughput) while also having a significantly lower latency per request (35 ms response on CampToCamp instead of 505 ms response on host 1). Part of this could be due to CampToCamp including the FastCGI library.

Performance: Resources perspective

Memory

This aspect of the test went exactly as expected. Both hosts show fairly steady memory usage throughout the course of the load testing. Mapserver is not very memory intensive, The Docker CampToCamp server was given 100MB of memory to work with (although I've never seen it go above 70MB usage even in load testing). Host 1 sits around 0.25 GB of RAM, and Host 2 sits around 0.44 GB of RAM. This is ultimately not all that big of a deal, our current Xpres servers have ~6 GB of RAM and I don't think a few extra hundred megabytes will make all that much of a difference in 2019.

CPU

I found the CPU results to be so unexpected that I thought I had done something wrong. I recompiled Mapserver several times, even including some of the optional libraries that weren't part of the original build plan. However every time I ran the tests the CPU usage was maxed out for the duration of the load test. Running the test against Xpres3 confirmed that this is isn't unusual for this kind of load with this implementation of Mapserver.

Host 1 sat at 100% CPU usage for the duration of the test, were as the highest recorded CPU usage in the CampToCamp test is 38%. CampToCamp averaged out to 18.87% usage for the duration of the test.

But how!?

Honestly I can only speculate as to why CampToCamp is so much less computationally expensive than traditionally compiled Mapserver.

Here are some of my theories that likely play a role:

- FastCGI: CampToCamp has FastCGI support compiled in, were we don't have that in our current infrastructure. I cannot say how much this could impact performance because I was unable to compile mapserver with FastCGI. FastCGI utilizes caching, which may explain the decreased response time and the increased throughput. Here is an article describing the performance benefits of FastCGI: https://fastcgi-archives.github.io/Understanding_FastCGI_Application_Performance_FastCGI.html
- Inefficient CGI scripts on existing infrastructure
- Newer version of Mapserver on CampToCamp, it is possible that version 7.5 is just way more efficient than version 7.4.2. Although its unlikely it would be this much more efficient.

Conclusion

Well performance aside I think its really important to note that it took me a week to compile Mapserver and its dependencies, but it took me an hour to set up the Docker Mapserver. I could rant about the many advantages of Docker outside of performance but I think the numbers in this study speak for themselves. The CampToCamp image is able to handle the same amount of requests in less time per request, while simultaneously only using ¼ of the CPU that the compiled Mapserver used. At the cost of only ~200 MB of memory. I believe this to be largely influenced by the FastCGI library compiled into the CampToCamp implementation of Mapserver.