Analyzing the Performance of Apache HTTP Server and MongoDB on AWS EC2 Instances

CSE 503S Performance Evaluation Study

Austin Tolani

Introduction

For my creative project for CSE330S, my partner and I developed a full-stack web application that allowed users to rate the food at dining halls on Washington University's campus. This web application was hosted on an Amazon T2.micro EC2 instance and used an Apache HTTP Server to serve our web application and MongoDB as a database solution. When configuring the Amazon EC2 instance, I was overwhelmed by the options that one has to choose from: different options for CPUs, memory and network performance as well as different options for the type of instance. Naturally, I was curious as to the effect of these configurations on web server and database performance.

In this study, I will explore the performance of different Amazon EC2 instances. More specifically, I will compare the performance of Apache HTTP Server and MongoDB on the different instances of the same "family" of EC2 instances (t2.micro,t2.small,t2.medium,t2.large,t2.xlarge) as well as across similarly priced instances in different "families" including general purpose instances and compute optimized instances (T2,T3,M4,C4).

Figure 1 shows the features of the instances included in this study as well as their on-demand pricing in the US East (Ohio) region as of December 2020. Amazon classifies all of the instances except for the C4 instance as "General Purpose Instances" while they classify the C4 instance as a "Compute Optimized Instance"

Figure 1: AWS Instance Types

Instance	vCPU	Memory	Network Performance	Price ($ per hour)
T2.micro	1	1	Low to Moderate	0.0116
T2.small	1	2	Low to Moderate	0.023
T2.medium	2	4	Low to Moderate	0.0464
T2.Large	2	8	Low to Moderate	0.0928
T2.Xlarge	4	16	Low to Moderate	0.1856
T3.Large	2	8	Up to 5 Gps	0.0832
M4.Large	2	8	Moderate	0.10
C4.Large	2	3.75	Moderate	0.10

Experimental Setup

A total of 8 instances were used to conduct experiments. All were mounted with the Amazon Linux 2 machine image with the default configuration. Once initialized, the same process was followed to set up each instance. First, all packages were updated and Development Tools was installed. Then, Apache HTTP Server (version 2.4) and MongoDB (version 4.4) were installed and configured to start upon reboot. For MongoDB, the configuration file was edited so that connections could be made from any IP address. Next, the respective ports for Apache and MongoDB were opened by editing the security groups of the instances in the AWS console.

Before any tests were run for either of the experiments, each instance was always stopped and then restarted. This was done to normalize the effect of CPU credits across the burstable instances (T2 and T3). Burstable instances have a baseline level of CPU usage available with the ability to "burst" above this level. The ability to burst is dependent on CPU credits that are earned every hour. Stopping and then starting a burstable instance sets CPU credit to 0.

All testing was done on a separate EC2 instance. The benchmarking tools used in this study can be resource intensive, so running them on a separate instance ensured that the benchmarking tools did not affect the resources available to what service was being tested. Additionally, when testing both Apache HTTP Server and MongoDB, the Private IPv4 address of the instance being tested was always used to avoid contention from other networks.

Apache Web Server Evaluation Setup

To measure the performance of Apache HTTP Server across instances, Apache Benchmark was used. This is a command line tool that allows you to benchmark web servers. Apache Benchmark takes two key parameters: the number of requests and the concurrency level, or the number of requests to perform at a time. Exploratory testing found that increasing the number of requests generally had a negligible effect on the outcome of Apache Benchmark. Changing the concurrency level had a significant effect on outcomes. To explore this effect further, my experiment involved running Apache Benchmark on each of the instances at a fixed number of requests (10,000) and at increasing concurrency level. 10 concurrency levels were tested for each instance, starting at no concurrency level (only one request at a time) and then increasing the concurrency level by 100 until a level of 1000 was reached. The same webpage was used for all tests.

Apache Benchmark returns many values that reflect the performance of the web server. In this experiment, the two values that I considered were the number of requests per second and the time within 90% of requests were served. Number of requests per second is a relatively standard measure of web server performance and represents the speed of a web server. The time which 90% of requests are served within is a less common measure and was chosen to reduce the effect of few unusually long requests that were always present for all instance types. Figure 2 shows example descriptive statistics for the 10,000 requests generated by Apache Benchmark with a concurrency level of 500 on a T2.Large instance.

Figure 2: Descriptive Statistics for 10,000 Requests performed by Apache Benchmark for T2.Large Instance

Mean	81.8609
Standard Error	3.37219687
Median	20
Mode	20
Standard Deviation	337.219687
Sample Variance	113717.117
Range	3404
Minimum	14
Maximum	3418
Sum	818609
Count	10000

These statistics show that most requests were performed in around 20 milliseconds however there were several very long requests (over 3000 milliseconds) that skew the mean. This pattern is present across all concurrence levels and instance types. It is not clear what causes these unusually long requests. Monitoring of system resources on the instances being tested showed that resource availability was not a factor. Additionally, analysis of the request times over the course of the benchmark showed that the unusually high response times did not correspond to the times at which the web server was being tested against a high concurrent load. My hypothesis is that these unusually large response times are a result of temporary poor network performance. Regardless of cause, the time which 90% of requests were served within is a measure that could measure web server performance while mostly ignoring the effect of the few very large response times. This measure represents the performance of the server during majority of requests.

Running 10 Apache Benchmark tests for each of the 8 instances meant that the tool had to be run a total of 80 times. Starting, waiting, and parsing the response given by the command line tool is a tedious and timely process. A Bash script was created to automate the process of calling Apache Benchmark repeatedly from the command line. This script allowed the 10 tests for each instance to be initiated with a single keystroke. Additionally, it allowed the intervals between each test to be standardized across all of the instances. The primary output of Apache Benchmark is printed to the terminal, meaning that after running the Bash script there were hundreds of lines of text in the terminal. Instead of searching through this text by hand to find the relevant values, a Python script was created to automate this process and find relevant values using Regular Expressions. The output of the Python script was then copied to an Excel spreadsheet for analysis.

MongoDB Evaluation Setup

To measure the performance of MongoDB across the instances, Yahoo! Cloud Serving Benchmark (YCSB) was used. This is an open-source program to evaluate the performance of databases. It is particularly suited towards benchmarking NoSQL databases, making it a good choice for benchmarking MongoDB. This tool is also run from the command line. YCSB is run in two stages. The first stage involves loading the initial working set into a blank database. This stage takes two key parameters: record count and threads. The second stage involves actually running read and update operations that simulate an actual workload. This stage also takes two key parameters: operation count and threads. In both stages, the number of threads represents the number of clients connecting to perform the operations. Similar to the web server performance experiment described above, exploratory testing showed that increasing the number of operations (in either stage) had a negligible effect on outcomes. Increasing the number of threads had a significant impact on performance. To explore this effect further, YCSB was run with a fixed number of inserted records/operations (10,000) and with increasing threads for both stages. For each of the 8 instances, YCSB was run for both stages with 1,2,4,8,32,64 and 128 threads respectively.

YCSB produces many results of interest. In this study, the primary value of interest was the number of operations/seconds or throughput that each instance supported. Each test consists of the two stages (each resulting in a measure of throughput) meaning that each test had a throughput value that represented inserts and read/write respectively.

Like Apache Benchmark, YCSB is run in the command line and prints its output to the terminal. Running YCSB 56 times is a timely and tedious task. Similar to the web server experiment described above, a Bash script was created to automate the process of calling YCSB repeatedly and a Python script was created to parse the output.

Results

Apache Web Server Performance Results

Figures 3 and 4 show the number of requests per second handled by Apache HTTP Server on the 8 instances tested at increasing concurrence levels. All instances seem to follow the same general pattern: the number of requests per second peaks at a concurrence level of 100 and then subsequently decreases as the concurrency level increases. In the T2 family, the T2.Large supports the largest number of requests per second at all concurrence levels except at a concurrence level of 1, where it performs the worst of all instances. Comparing the T2,T3,C4 and M4, the T2 and M4 generally support higher requests per second than the T3 and C4 at all concurrence levels.

Figure 3: Requests per Second for T2 Instances

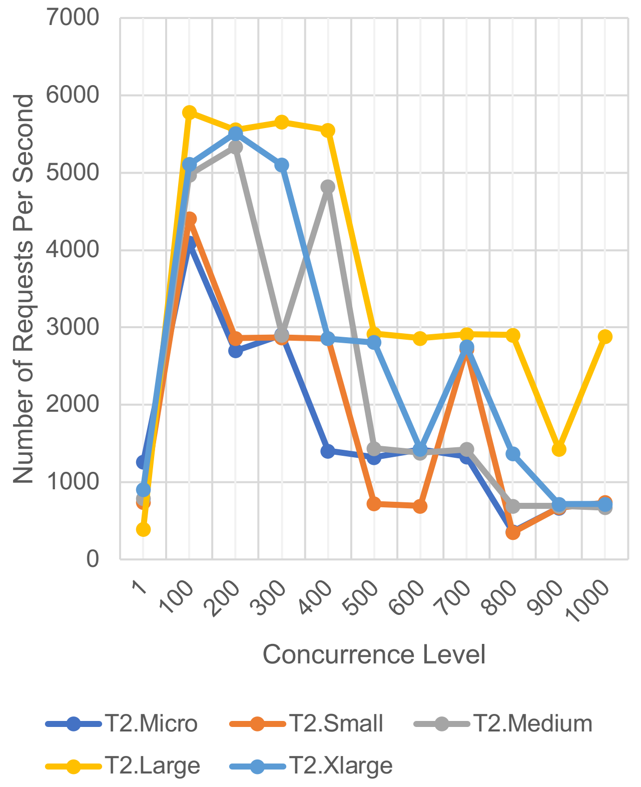


Figure 4: Requests per Second for T2, T3, M4 and C4 Instances

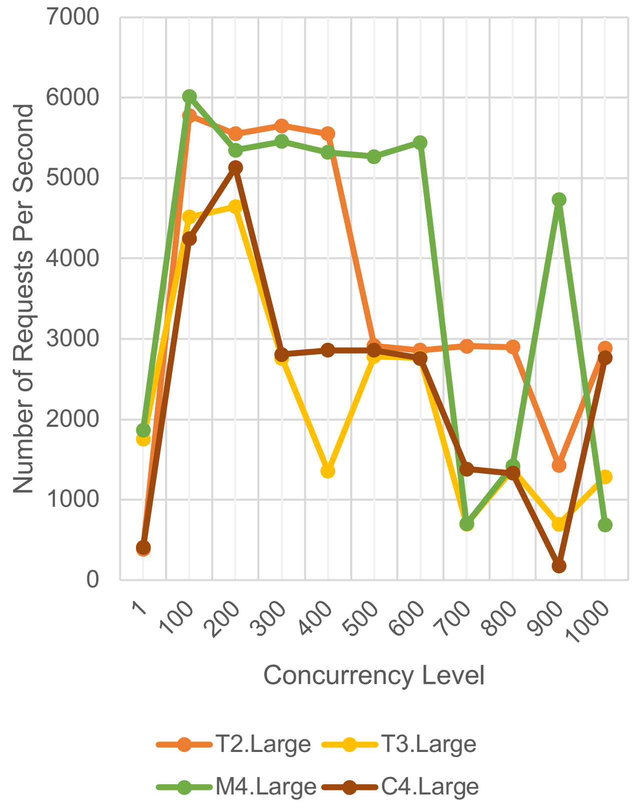


Figure 5 and Figure 6 show the time which 90% of requests were served within by Apache Http Server on the 8 instances tested at increasing concurrence levels. The T2.Large and T2.Xlarge instances perform similarly for all concurrency levels up to 800, after which the T2.Xlarge instance serves requests significantly quicker. Notably, up until a currency level of 1000 (where Apache Benchmark timed out for the instance), the T2.Micro outperforms the T2.Small and T2.Medium. For the T2,T3,C3, and M4, the T2 serves 90% of requests similarly or faster than all other instance types at all concurrence levels.

Figure 5: Time within 90% of Requests were Served For T2 Instances (ms)

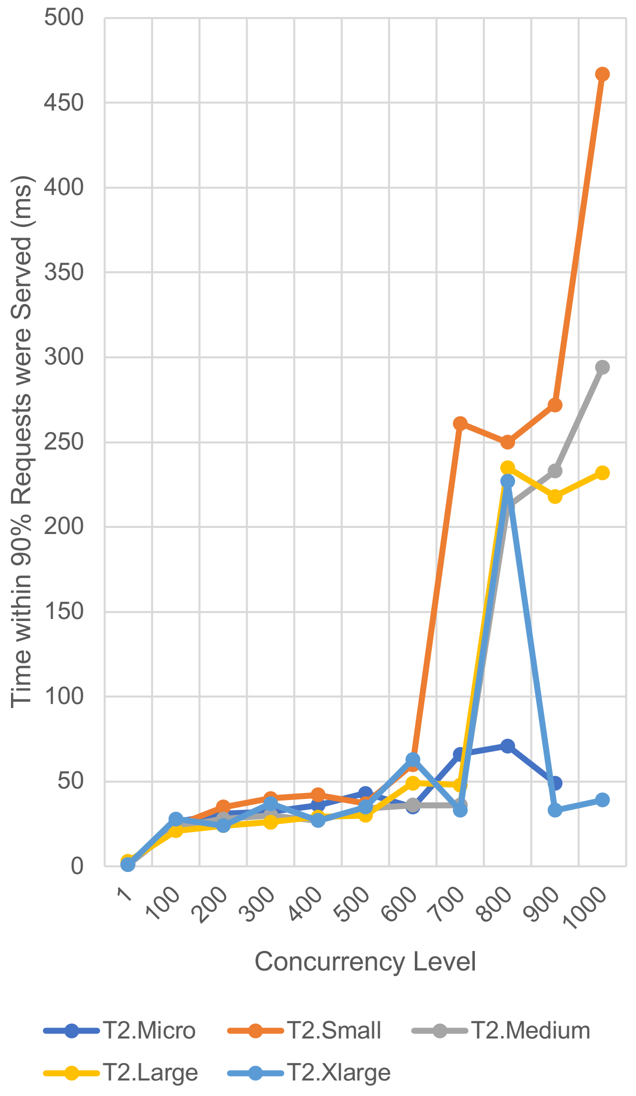


Figure 6: Time within 90% of Requests were Served For T2,T3,M4 and C4 Instances (ms)

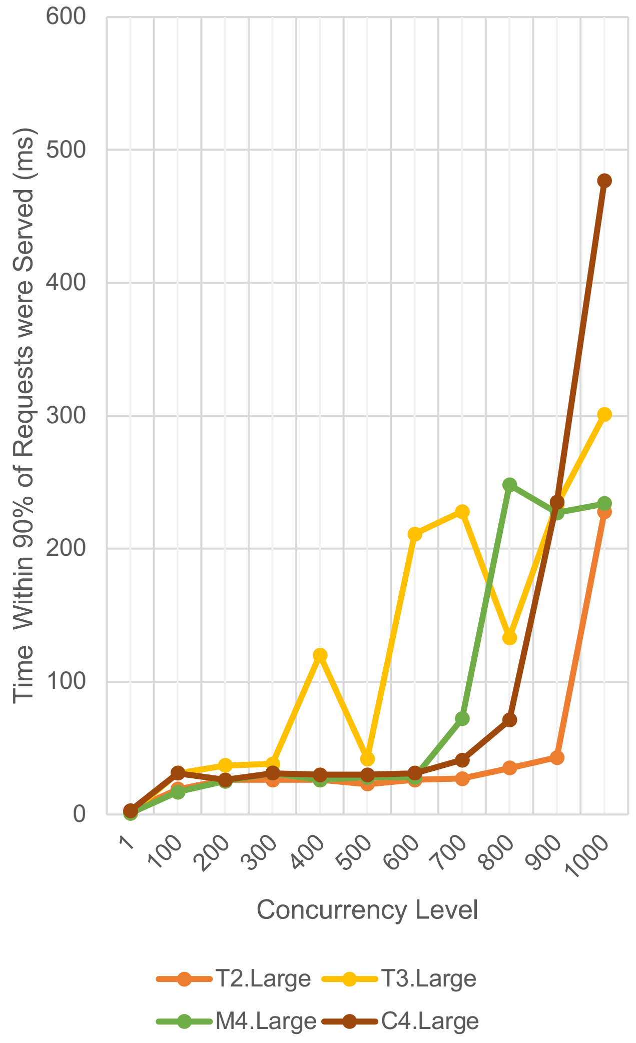


Figure 7: Operations Per Second (inserts) for T2 Instances

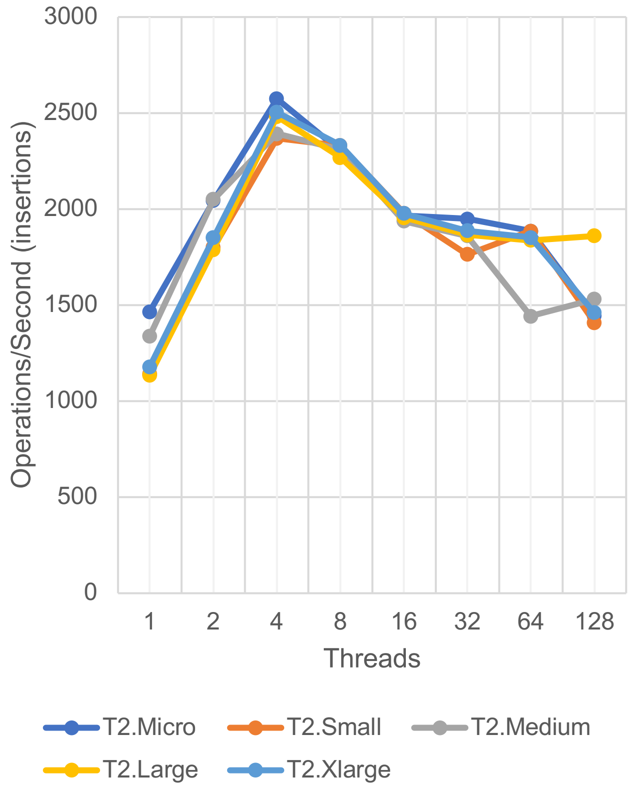
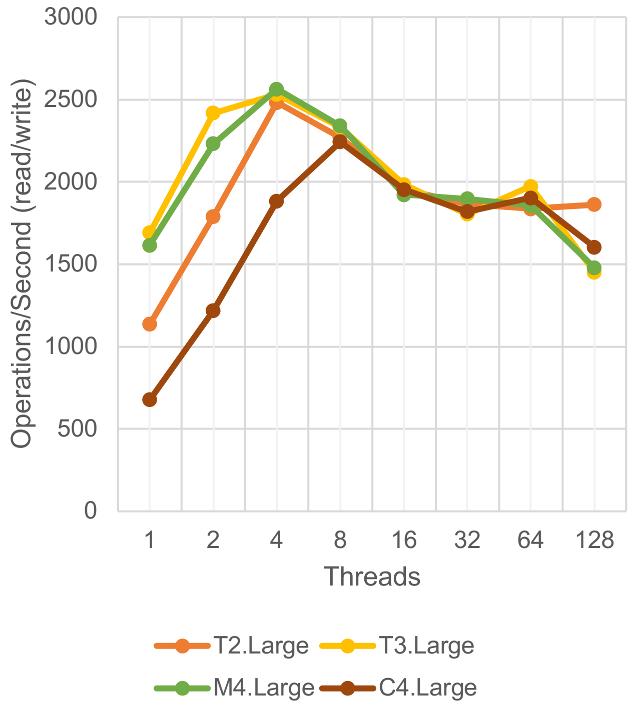


Figure 8: Operations Per Second (inserts) for T2, T3, M4 and C4 Instances



MongoDB Performance Results

Figure 7 and 8 show the throughput (operations/seconds) for insertions into MongoDB on the 8 instances. Figures 9 and 10 show the throughput (operations/second) for read/writes on the 8 instances. For both measures of throughput, the T2.Micro surprisingly performs the best against other T2 instances when the number of threads was less than 4. At higher threads, the T2.Large generally supported the highest throughput. For the T2,T3,M4 and C4 instances, the T3 instance generally performed the best overall. Notably, the T2 instance generally supported the highest throughput at high thread counts.

Figure 9: Operations Per Second (inserts) for T2 Instances

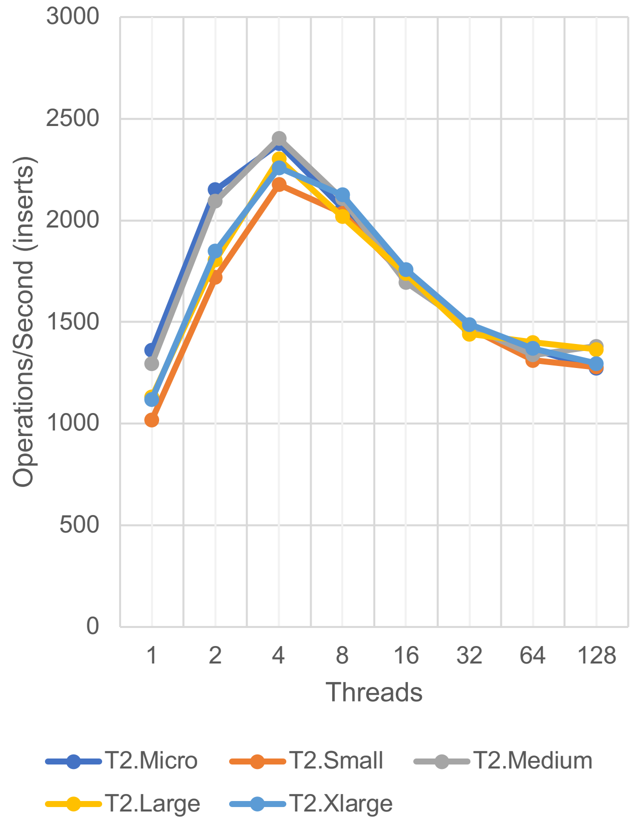
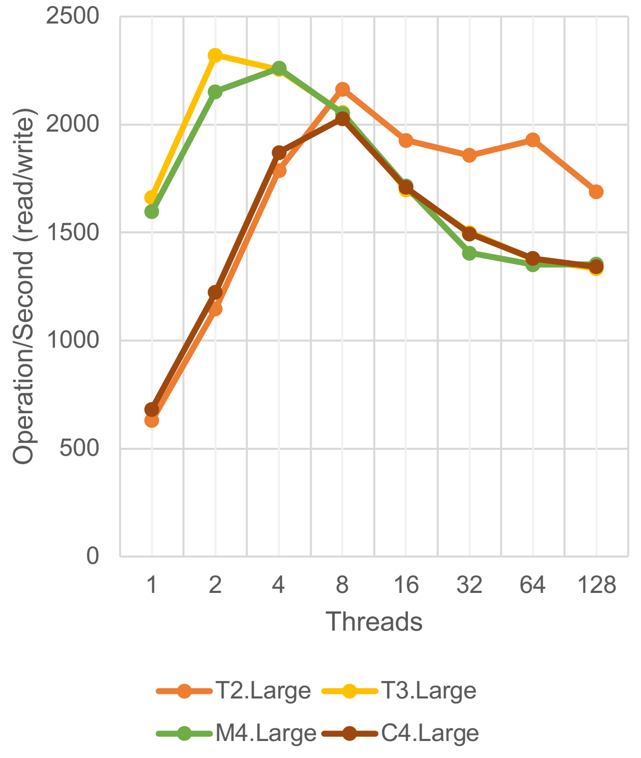


Figure 10: Operations Per Second (inserts) for T2, T3, M4 and C4 Instances



Discussion

Expectedly, determining what instance to serve your web application depends on how the instance is going to be used. Namely, how many requests or operations you expect your instance to handle at a single time.

For an Apache HTTP Server, the T2 instance performs the best in aggregate. It performs similarly to the T3 instance at all concurrence levels in the number of requests it serves and performs significantly better than the T3,C4 and M4 in the time it takes to serve most requests. Within the T2 family, the micro instance certainly presents the best value proposition. If you only need to support 600 concurrent requests or less, a micro instance will handle 90% of requests at a similar or faster time than all other instances in its family. The number of requests the micro instance serves in a second is certainly less but comparable to the medium, large and extra-large instances. This is remarkable given that the micro instance is a fraction of the cost of the other instances in its family. If you need to support more than 600 concurrent requests, or want your web server to serve generally more requests per second, the medium instance is the best value. It serves more requests per second than the micro and small instances and serves a similar amount to the large and extra-large instances at a cost of half of the large instance and a quarter of the extra-large instance respectively. If maximum requests per seconds and/or the ability to support 1000 concurrent requests is of priority, the large instance unexpectedly is the best option. It outperforms the extra-large instance in requests per second at all concurrency levels tested and is half the price.

Testing the Apache HTTP Server did not reveal any significant bottlenecks in the system. For all instances, CPU or memory utilization never came close to reaching capacity. As mentioned previously, all instances suffered from infrequent unusually long requests that appear to be independent of load or concurrency. My hypothesis is that these long requests are caused by temporary poor network performance. This suggests that the network bandwidth of the instances is a potential bottleneck however much more testing would need to occur to confirm this hypothesis.

For serving a MongoDB database, the T2 instance supports most insert and read/write operations when the number of connecting clients (threads) is roughly below 8. When the number of connecting clients is above 8, the T3 instance supports the most insert and read/write operations. Within the T2 family, while there are differences between the difference instances in the number of insert and read/write operations they support, these differences are small. All instances generally reach peak throughput at around 4 threads and reduce their throughput as the number of threads increases past 4. Notably, the smaller instances (micro, small) have higher throughput at lower threads. The opposite is true at higher threads. Again, benchmarking MongoDB did not reveal any bottlenecks. Monitoring system resources at the time of the tests revealed that MongoDB never came close to reaching CPU or memory capacity.

For serving an Apache HTTP Server and MongoDB database, the T2 instance performs the best overall. Within the T2 instance family, the micro instance certainly provides the best value, unless the number of concurrent requests to the web server or operations to the database is relatively high, at which point the medium instance provides the best value while the large instance provides the best performance. I expect that past 1,000 concurrent requests or 128 concurrent operations, the extra-large instance would start to perform best due to it's greater number of virtual CPUs. For most types of web applications that don't have thousands of users every second, the T2 micro instance is undoubtedly suitable and the best value.

Limitations

There were several limitations to this study. Most notably, when using Apache Benchmark, every instance suffered from having some unexpectedly large requests. This had the effect of making the output of Apache Benchmark inconsistent. Figure 11 shows the output of Apache Benchmark (Requests per Second) on the same instance with the same number of requests and concurrency level.

Figure 11: Requests Per Second for Identical Apache Benchmark Trials

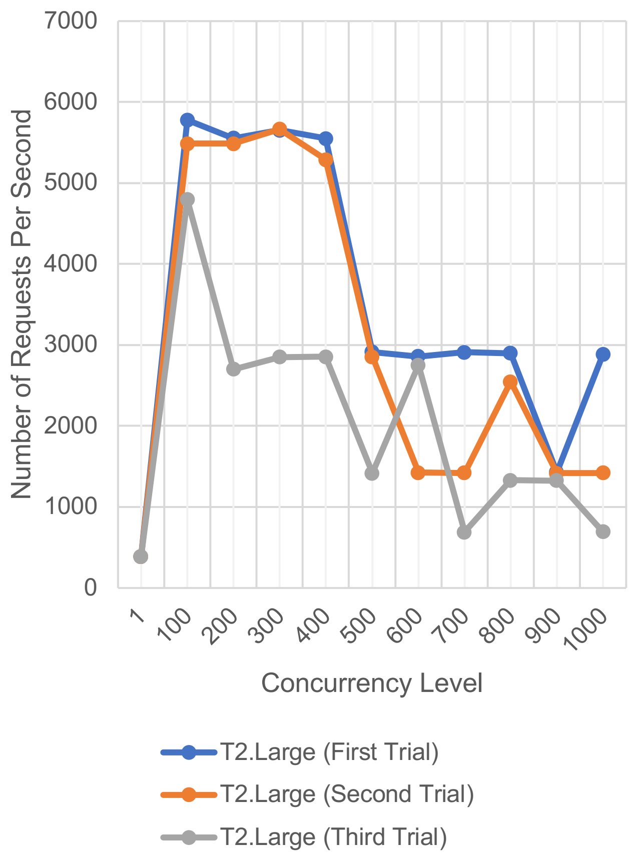


Figure 11 shows how inconsistent the output of Apache Benchmark was. The effect of this inconsistency was attempted to be mitigated by including the "90% of requests served within a certain time" value in the analysis. Nonetheless, the inconsistency of the results of this study is certainly a limitation and if I were to continue this study, I would make sure to use the average of multiple trials.

Another limitation was that I only compared the performances of instances within the same family for the T2 family and not other AWS EC2 families. It is not clear whether or not the same performance outcomes would also be true between the micro, small, medium, large and extra-large instance types of the T3, C4, and M4 instances. An extension of this study could analyze if the performance outcomes are similar for the difference instance families.

Additionally, I must note that Apache HTTP Server and MongoDB were tested independent of any interaction with other services. Typically, an instance will host a web server, backend processing, a database, a REST API, and more. The performance of MongoDB and Apache HTTP Server would certainly be different if other services were using system resources simultaneously. While it would be more difficult to simultaneously benchmark multiple services, it would also be an interesting extension to this study.

Conclusion

In conclusion, choosing the type and configuration of your AWS EC2 instance is contingent on what you will be using your instance for. This study found that for hosting an Apache Web Server and a MongoDB database, the T2 family of instances performed the best overall compared to the T3, C4 and M4 families, although a case could be made for the T3 instance in certain use cases. Within the T2 family the micro instance provides exceptional value as it is a fraction of the cost of and provides better or similar performance to other instances in its family, provided that the number of concurrent requests/operations to the web server/database is moderate. For most web applications that don't have thousands of active users, the T2 micro instance provides comparable performance to more expensive configurations and is relatively inexpensive. For web applications with more demanding workloads, more close consideration and testing needs to be done to pick an optimal hosting solution based on load and technologies used. For most developers, the T2 micro instance is an excellent choice for developing and testing web applications, particularly because it is included in AWS's "free tier".

I was initially excited to do this study because I felt that it would lead to useful knowledge that I could use the next time I need to deploy a web application. Obviously, I have learned a lot, but nonetheless I find it ironic that I am left with the same guiding principle that I had when I configured an EC2 instance for the first time: just use the free one!

Notes

All of the data generated in this study as well as the scripts created to automate the workflow can be found in this repository.