Drupal 8 Commerce Performance Benchmarks

February 15, 2019 – Shawn McCabe, CTO

acromedia.com

solutions@acromedia.com

Table of Contents

Overview

This document compiles the results of testing how Drupal 8 Commerce handles increasing traffic loads and how it scales. All these tests are performed against AWS hardware using the testing configurations listed below. It should be reproducible by anyone.

Setup

Testing Software: jMeter
Scripts: https://gitlab.com/acromedia/performance
Hardware: AWS
Drupal and Server Config: Drupal 8 Tuning, Drupal 7 Tuning
Test Site: Copy of https://commerceplus.acromedia.com/

Limitations

- Traffic was simulated and not actual traffic. Simulated traffic was made as realistic as possible, but is ultimately not real traffic.
- Not all permutations of server configurations were tested. For example, adding Varnish to some of the smaller 1 or 2 server options would probably be beneficial, but was not explicitly tested.
- Tests were conducted within the same network, so data transmission time was negligible.
- Not all tests pushed the servers the exact same amount. Each version endeavored to push the servers to the point where performance started to degrade, but this is not exactly the same degradation for each test as degradation happened in different ways in each configuration. Small tweaks could probably be made for additional small performance gains.

Methodology

We took some data from actual client sites to see roughly how many page views happened compared to every order. We did this so that we could get a rough simulation of actual traffic. Turns out, this ranged anywhere between 35 for very streamlined sites with few products to over 400 for large sites with poor conversion. Taking an average of what we considered to be standard catalog sites, we ended up with about 180 page views per order. An order takes roughly 11-20 page views including catalog and product pages, adding to cart pages and checkout flow pages. Obviously, this could be more direct, but users don't go through the process in the minimum steps as they browse around. With that in mind, I setup 2 user types, one for general browsing and 1 for purchasing, and ran them at an 8:1 ratio – 8 "just browsing" for every 1 buyer. Those 8 "just browsing" users are probably made up of many more than 8 users, but the ratio is what is important.

This gets the right ratio of non-buy to buying pages, but doesn't quite accurately simulate the number of users because purchasing users likely have more page views than non-purchasing users. Comparing users to purchases results in about a 12:1 ratio on average, so we modify the concurrent user results to match this ratio, adjusting up by 5/13ths.

Testing

I would run the tests for 10 minutes, taking about 90 seconds to spool up the purchasing users so they would be spread out amongst the different steps. Usually, the data wouldn't change between about minute 3 onward. However, occasionally there would be a slow building bottlenecks that wouldn't appear until later, hence the longer tests. It is possible that even longer tests would uncover more flaws, but any tests I ran stabilized after 5-7 minutes.

I then went through a series of server setups, from a small everything-on-one server setup to a large multi-server setup consisting of 5 different servers. I did have to go back and re-run previous instances as I went along, as I would tweak settings to simulate better user traffic or fix small bugs and bottlenecks.

Results

All server setups are done using AWS services, primarily EC2 instances of various sizes. You should be able to take the supplied code

and setup and achieve similar results. I switched from t3 servers to c5 servers early on as it became apparent cpu was usually the primary bottleneck, not memory.

I tested using Amazon RDS as the database backend instead of a plain EC2 instance, but got 2.5x worse performance. Some googling revealed other people having poor performance with RDS, but I am leery of a flaw in the tests causing this as well since it is such a big difference.

No RDS results are included below.

For caching, I used ElasticCache Redis, which performed slightly better than Memcache but was essentially equal. I did not test any stand-alone EC2 Redis instances.

1 Server

- Running NGINX and MySQL
- No caching servers, all caching handled via MySQL

I adjusted the script slightly to allow a small amount of ramp up in the browsing flow since the sudden slam of users would cause a temporary spike. Spreading it out for 30 seconds removed this, which I believe gives more accurate results. If using Varnish, this isn't necessary as the Varnish server absorbs the spikes easily.

T3.medium

AWS On Demand Cost: $30.46 / month USD | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
1 Server t3.medium	130	8163	0.00%	317.27 MS	6 MS	5627 MS	2169.72 MS	13.59 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.large

AWS On Demand Cost: $62.22 / month USD | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
1 Server c5.large	156	9909	0.00%	312.69 MS	5 MS	4032 MS	2054.3 MS	16.49 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.2xlarge

AWS On Demand Cost: $248.88 / month USD | View Full Results »

At this size the 1 server option started to scale poorly. Imbalances in usage from both PHP and MySQL caused the server to be underutilized, but it still suffered from occasional errors when both PHP and MySQL would temporarily spike. I had to set the test lower than I expected and the single server still had a small error rate.

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
1 Server c5.2xlarge	520	34909	0.03%	177.07 MS	3 MS	3112 MS	1375.97 MS	58.15 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.9xlarge

AWS On Demand Cost: $1119.96 / month USD | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
1 Server c5.9xlarge	1820	115379	0.45%	278.96 MS	0 MS	3515 MS	2437.99 MS	193.15 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.18xlarge

AWS On Demand Cost: $2239.92 / month USD | View Full Results »

The response time is excellent, but running the web server and the database at such a high load seems to result in a higher error rate. Pushing the server any farther just caused further error rate spikes that couldn't be resolved with PHP and MySQL tuning.

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
1 Server c5.18xlarge	2600	173732	0.34%	159.33 MS	0 MS	1279 MS	808.99 MS	289.3 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

2 Servers

- 1 web server running NGINX
- 1 database server running MySQL

T3.medium

AWS On Demand Cost: $60.92 / month USD | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
2 Server Web: t3.medium Database: t3.medium	130	8495	0.00%	299.56 MS	9 MS	8523 MS	2336.8 MS	14.26 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.large

AWS On Demand Cost: $92.68 / month USD | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
2 Server Web: c5.large Database: t3.medium	182	10936	0.00%	511.79 MS	9 MS	8593 MS	3480.26 MS	18.26 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.2xlarge

AWS On Demand Cost: $279.34 / month USD | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
2 Server Web: c5.2xlarge Database: t3.medium	520	34760	0.07%	213.86 MS	0 MS	5039 MS	1160.97 MS	58 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.9xlarge

AWS On Demand Cost: $1505.73 / month USD | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
2 Server Web: c5.9xlarge Database: c5.2xlarge	2080	136976	0.29%	184.58 MS	2 MS	1643 MS	979.99 MS	228.54 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

c5.18xlarge

At this point, not having a caching server becomes a significant bottleneck. The heavy cache updates cause the database server to start I/O bottlenecking. The average speed stays OK, but there is a roughly 1% error rate and some edge case time spikes.

AWS On Demand Cost: $3011.45 / month USD | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
2 Server Web: c5.18xlarge Database: c5.4xlarge	3900	171552	1.75%	1432.11 MS	0 MS	28676 MS	11270 MS	286.06 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

3 Servers

- 1 web server running NGINX
- 1 database server running MySQL
- 1 caching server running Redis

The database usage drops dramatically once the caching is moved to a separate server. Probably somewhere around ⅔ of all database usage comes from writing and reading cache tables.

T3.medium

AWS On Demand Cost: Caching and Database oversized, price not accurate. | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
1 Server t3.medium	130	8163	0.00%	317.27 MS	6 MS	5627 MS	2169.72 MS	13.59 TPS

1 Server c5.large	156	9909	0.00%	312.69 MS	5 MS	4032 MS	2054.3 MS	16.49 TPS
1 Server c5.2xlarge	520	34909	0.03%	177.07 MS	3 MS	3112 MS	1375.97 MS	58.15 TPS
1 Server c5.9xlarge	1820	115379	0.45%	278.96 MS	0 MS	3515 MS	2437.99 MS	193.15 TPS
1 Server c5.18xlarge	2600	173732	0.34%	159.33 MS	0 MS	1279 MS	808.99 MS	289.3 TPS
2 Server Web: t3.medium Database: t3.medium	130	8495	0.00%	299.56 MS	9 MS	8523 MS	2336.8 MS	14.26 TPS
2 Server Web: c5.large Database: t3.medium	182	10936	0.00%	511.79 MS	9 MS	8593 MS	3480.26 MS	18.26 TPS
2 Server Web: c5.2xlarge Database: t3.medium	520	34760	0.07%	213.86 MS	0 MS	5039 MS	1160.97 MS	58 TPS
2 Server Web: c5.9xlarge Database: c5.2xlarge	2080	136976	0.29%	184.58 MS	2 MS	1643 MS	979.99 MS	228.54 TPS
2 Server Web: c5.18xlarge Database: c5.4xlarge	3900	171552	1.75%	1432.11 MS	0 MS	28676 MS	11270 MS	286.06 TPS
3 Server Web: t3.medium Database: c5.4xlarge Redis: r5.2xlarge	390	11701	0.00%	405.38 MS	10 MS	6535 MS	2963.96 MS	19.42 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.large

AWS On Demand Cost: Caching and Database oversized, price not accurate. | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
3 Server Web: c5.large Database: c5.4xlarge Redis: r5.2xlarge	520	14758	0.00%	515.18 MS	10 MS	7244 MS	3542.1 MS	24.55 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.2xlarge

AWS On Demand Cost: Caching and Database oversized, price not accurate. | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
3 Server Web: c5.2xlarge Database: c5.4xlarge Redis: r5.2xlarge	2080	57048	0.05%	435.88 MS	3 MS	3757 MS	2716 MS	94.92 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.9xlarge

AWS On Demand Cost: Caching and Database oversized, price not accurate. | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	90th Pct	Throughput
3 Server Web: c5.9xlarge Database: c5.4xlarge Redis: r5.2xlarge	6500	199639	0.12%	320.51 MS	0 MS	3245 MS	1238 MS	331.6 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.18xlarge

AWS On Demand Cost: $3705.54 / month USD | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
3 Server Web: c5.18xlarge Database: c5.4xlarge Redis: r5.2xlarge	10400	282245	0.12%	622.19 MS	0 MS	4767 MS	3806.99 MS	469.45 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

4 Servers

- 1 web server running NGINX
- 1 database server running MySQL
- 1 caching server running Redis
- 1 caching server running Varnish

Adding Varnish has no effect on checkout pages since they are unique, but, on pages that can be cached, such as product and catalog pages, it has a great effect on both speed and load. It reduced them to usually <5ms and completely removed that load from all servers except the Varnish server, where the load is minimal.

The Varnish server is fairly large, but is underutilized. The other web servers continue to be the bottleneck. It saved me time to not have to constantly resize all the servers, but you could likely run a much smaller Varnish server.

T3.medium

AWS On Demand Cost: Caching and Database oversized, price not accurate. | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
4 Server Web: t3.medium Database: c5.4xlarge Redis: r5.2xlarge Varnish: c5.2xlarge	390	11747	0.00%	369.43 MS	0 MS	5004 MS	2860.52 MS	19.56 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.large

AWS On Demand Cost: Caching and Database oversized, price not accurate. | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
4 Server Web: c5.large Database: c5.4xlarge Redis: r5.2xlarge Varnish: c5.2xlarge	780	20867	0.00%	650.25 MS	0 MS	8546 MS	4551 MS	34.58 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.2xlarge

AWS On Demand Cost: Caching and Database oversized, price not accurate. | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
4 Server Web: c5.2xlarge Database: c5.4xlarge Redis: r5.2xlarge Varnish: c5.2xlarge	1950	60937	0.00%	277.03 MS	0 MS	2872 MS	1538.99 MS	101.15 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.9xlarge

AWS On Demand Cost: Caching and Database oversized, price not accurate. | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
4 Server Web: c5.9xlarge Database: c5.4xlarge Redis: r5.2xlarge Varnish: c5.2xlarge	7800	246666	0.00%	250.78 MS	0 MS	3196 MS	2325.92 MS	409.76 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.18xlarge

At this point, the Redis server started to max out and had to be upped to 4xlarge because it had become the bottleneck.

AWS On Demand Cost: $3979.31 / month USD | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
4 Server Web: c5.18xlarge Database: c5.4xlarge Redis: r5.4xlarge Varnish: c5.2xlarge	18200	482065	0.00%	669.4 MS	0 MS	5626 MS	4497.99 MS	798.96 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

5 Servers

- 2 web servers running NGINX
- 1 database server running MySQL
- 1 caching server running a 5 node Redis cluster
- 1 caching server running Varnish

Redis on AWS became a bottleneck, so I switched to a cluster setup. Redis doesn't multi-thread well and adding more hardware didn't help, but a cluster worked nicely and was actually cheaper. I would probably run a cluster in most instances in the future.

I also compared against Memcache, which produced slightly slower but essentially the same results.

You could probably scale this same architecture to a couple more web servers and an additional Varnish server before you got into database trouble. You could then look at clustering your database setup.

T3.medium

AWS On Demand Cost: Caching and Database oversized, price not accurate. | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
5 Server Web: 2x t3.medium Database: c5.4xlarge Redis: 5x r5.large Varnish: c5.2xlarge	1560	33289	0.00%	628.5 MS	0 MS	8066 MS	6536.97 MS	55.19 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.large

AWS On Demand Cost: Caching and Database oversized, price not accurate. | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
5 Server Web: 2x c5.large Database: c5.4xlarge Redis: 5x r5.large Varnish: c5.2xlarge	2080	42660	0.00%	685.43 MS	0 MS	9187 MS	8192.94 MS	70.97 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.2xlarge

AWS On Demand Cost: Caching and Database oversized, price not accurate. | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
5 Server Web: 2x c5.2xlarge Database: c5.4xlarge Redis: 5x r5.large Varnish: c5.2xlarge	5200	117019	0.00%	544.56 MS	0 MS	5728 MS	4680.99 MS	194.39 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.9xlarge

AWS On Demand Cost: Caching and Database oversized, price not accurate. | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
5 Server Web: 2x c5.9xlarge Database: c5.4xlarge Redis: 5x r5.large Varnish: c5.2xlarge	26000	560355	0.00%	607.33 MS	0 MS	9589 MS	7261.97 MS	929.1 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

C5.18xlarge

AWS On Demand Cost: $6618.79 / month USD | View Full Results »

Layout	Concurrent Users	# Samples	Error %	Avg	Min	Max	99th Pct	Throughput
5 Server Web: 2x c5.18xlarge Database: c5.4xlarge Redis: 5x r5.large Varnish: c5.2xlarge	52000	786400	0.01%	763.5 MS	0 MS	41122 MS	10510.92 MS	1305.85 TPS

*MS = *milliseconds* ** TPS = *transactions per second*

Conclusions

Generally, scaling went much better than I expected. I ended up with top-end numbers probably 8x what I predicted they would be.

The primary differences seemed to be in both the utilization of Varnish and the lack of database bottlenecks.

Drupal has always been heavy on databases and write heavy operations like processing orders, so I anticipated Drupal itself being a large bottleneck. This was the case when running the caching through the database, but once the caching was moved outside of the database, I had no further database scaling issues. The next bottlenecks became Redis and the web server.

Varnish has long been a primary tool for getting good performance speed, but its use has been limited with the Commerce module because many parts of Commerce are difficult to cache. With all of the recent improvements in the modules caching architecture, I was able to cache everything before the cart. This provided a very significant improvement in both response time and overall capacity.

Recommendations

For any Drupal Commerce implementation, I would recommend using at minimum 4 server setup, even if those servers are all very small or share the same machine. The improvements from splitting out caching were significant enough to warrant this, even on small setups. However, make sure the servers are being utilized correctly as custom code and contributed modules can limit caching.

At this time, I would also recommend not using RDS and instead use an EC2 instance or instances with MySQL. I would like to do a full follow up test specifically around this to confirm.

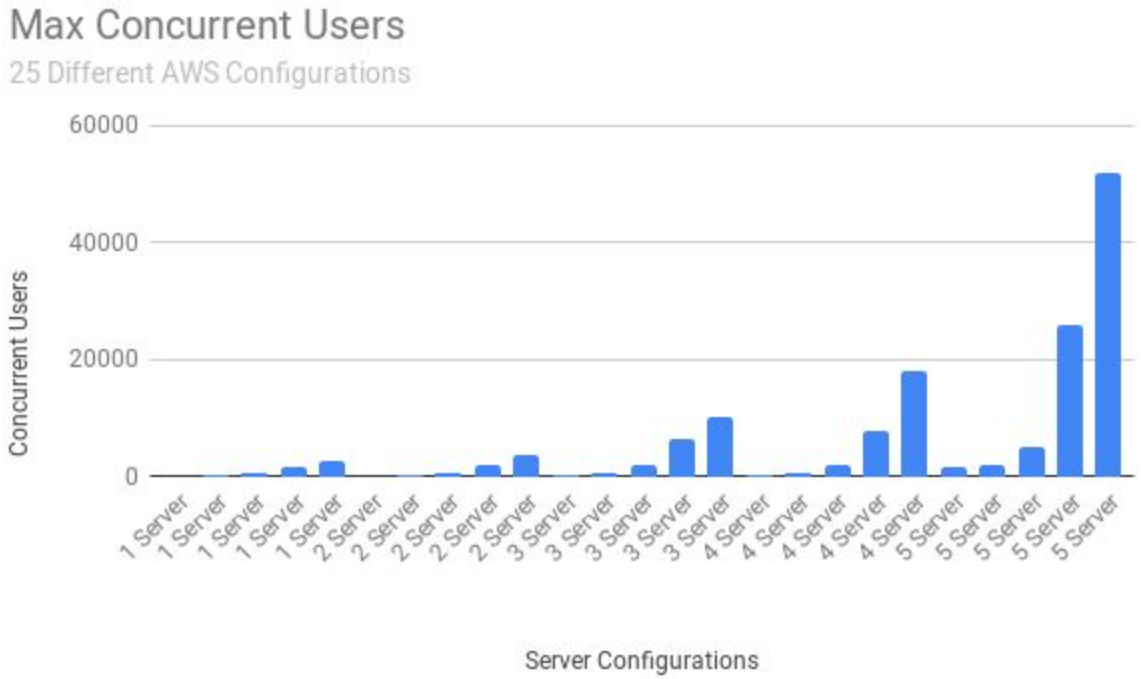
Finally, I recommend using a clustered Redis setup because a single Redis instance doesn’t make good use of multiple threads or cores. Redis crustered became more performant at later stages and should also be more cost effective, even on smaller setups.

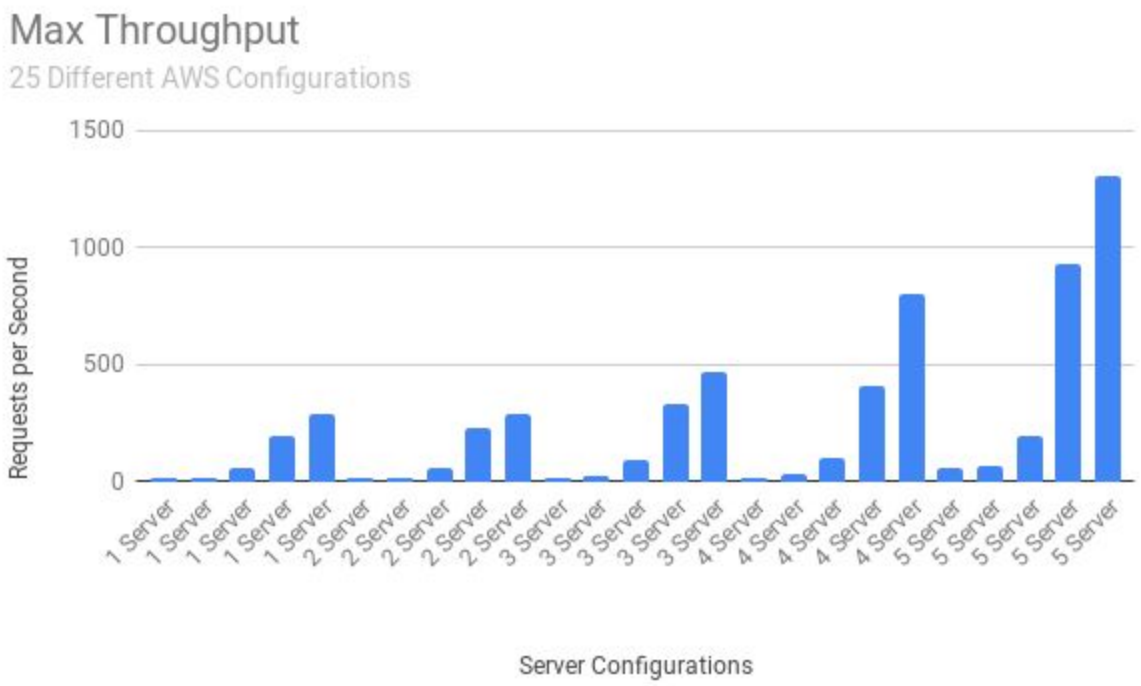
Followup

I would like to do additional sets of benchmarks to clarify some tentative findings from this report:

- Run tests comparing multiple EC2 and RDS database setups. This is to confirm RDS speed findings and also record MySQL cluster data.
- Run more tests comparing multiple Redis and Memcache setups. This is to provide more clarity on what setups are faster and/or cheaper.
- Run tests comparing response time at optimal loads instead of maximum loads. This is to see which options perform best when not under heavy load.
- Try additional server configurations to track costs/performance for all setups, not just the largest ones.

Appendix A: Charts





Appendix B: Complete Data

View Complete Data Spreadsheet »

https://docs.google.com/spreadsheets/d/1_F9wB1w9M0AAngOvAvlT-XQQilHdCHEtNl4kJNgcLeY/edit#gid=0