Drupal 8 Commerce Performance Benchmarks

February 15, 2019 – Shawn McCabe, CTO

acromedia.com

solutions@acromedia.com

Table of Contents

Overview

Setup

Limitations

Methodology

Testing

Results

- 1 Server
  - T3.medium
  - C5.large
  - C5.2xlarge
  - C5.9xlarge
  - C5.18xlarge
- 2 Servers
  - T3.medium
  - C5.large
  - C5.2xlarge
  - C5.9xlarge
  - c5.18xlarge
- 3 Servers
  - T3.medium
  - C5.large
  - C5.2xlarge
  - C5.9xlarge
  - C5.18xlarge
- 4 Servers
  - T3.medium

C5.large
C5.2xlarge
C5.9xlarge
C5.18xlarge
5 Servers
T3.medium
C5.large
C5.2xlarge
C5.9xlarge
C5.18xlarge
Conclusions
Recommendations
Followup
Appendix A: Charts
Appendix B: Complete Data

Overview

Testing how Drupal 8 Commerce handles increasing traffic loads, how it scales and what hardware is required. All these tests are performed against AWS hardware using the testing configurations listed below and 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 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.
- 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, as well as small tweaks could probably be done for small additional amounts of performance.

Methodology

We took some data from actual client sites to see roughly how many page views happened compared to every order, so 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 pages per order. An order takes roughly 11-20 page views, with catalog and product pages, adding to cart and a checkout flow. 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" people are probably made up of many more than 8 people, 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, as 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, but occasionally there would be slow building bottlenecks that wouldn't appear until later, hence the longer tests. It is possible longer tests would uncover more flaws, but any tests I ran stabilized after 5-7 minutes.

I then went through a series of 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 and 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.

Caching 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 of 30 seconds removed this, which I think gives more accurate results. When 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 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

*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 that database at such 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 or 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 as the database server starts to IO bottleneck with the heavy cache updates. 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 - nginx
- 1 database server - mysql
- 1 caching server - 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 write 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 - nginx
- 1 database server - mysql
- 1 caching server - redis
- 1 caching server - varnish

Adding varnish has no effect on checkout pages that are unique, but has a great effect on both speed and load for product and catalog pages that can be cached, reducing them to usually <5ms and completely removing that load from all servers except the varnish server, where the load is minimal.

The varnish server is fairly large, but is underutilized, the web servers continue to be the bottleneck, it saved time to not have to constantly resize all the servers. You could likely run a much smaller 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 as it has 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 - nginx
- 1 database server - mysql
- 1 caching server - 5 node redis cluster
- 1 caching server - varnish

Redis on AWS became a bottleneck at this point, I switched to a cluster setup since redis doesn't multi-thread well and adding more hardware didn't help. The 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 and had to 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.

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. I anticipated Drupal itself being a large bottleneck. The was the case when running the caching through the database, but once the caching is moved outside of the database, I had no further database scaling issues. The next bottlenecks became Redis and the web server. bottlenecked on Redis or web server performance first.

Varnish has long been a primary tool for getting good performance speed. However, its use has been limited in commerce because

of many parts of commerce are difficult to cache. With all the improvements in the caching architecture, I was able to cache everything before the cart, which 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 server are all very small or even some share the same machine. The improvements from splitting out caching were significant enough to warrant this even on small setups, although make sure these are being utilized correctly as custom code and many contrib modules can limit caching.

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

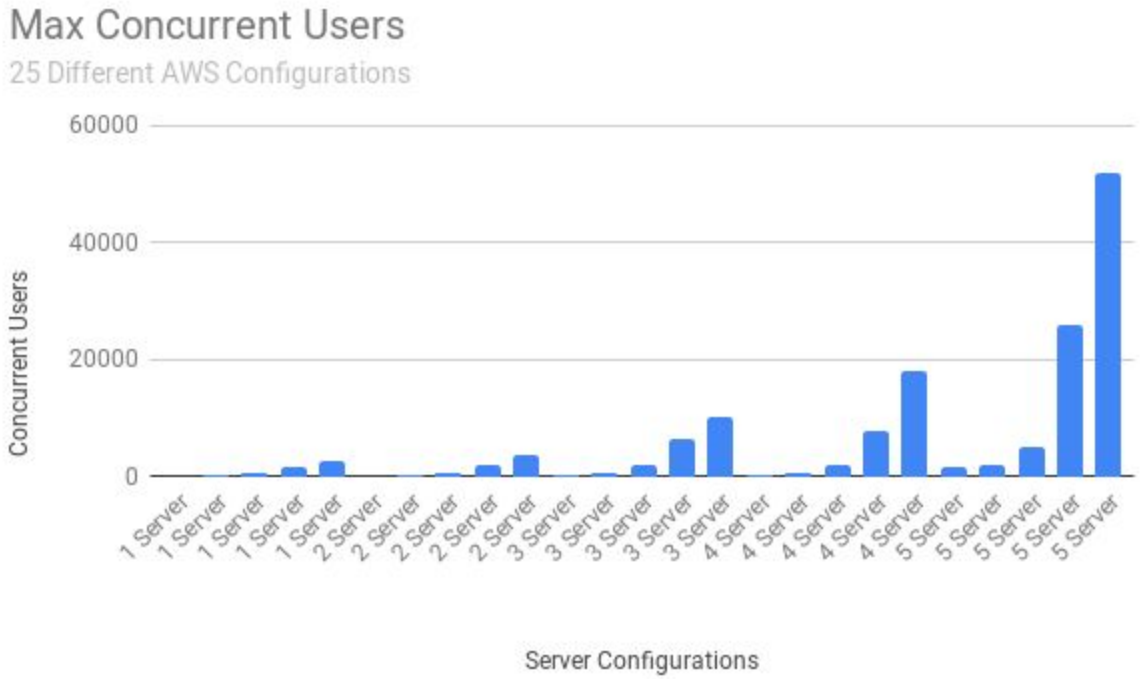
I would also recommend using a clustered Redis setup, as it became more performant at later stages and also should be more cost effective even on smaller setups as a single Redis instance doesn't make good use of multiple threads or cores.

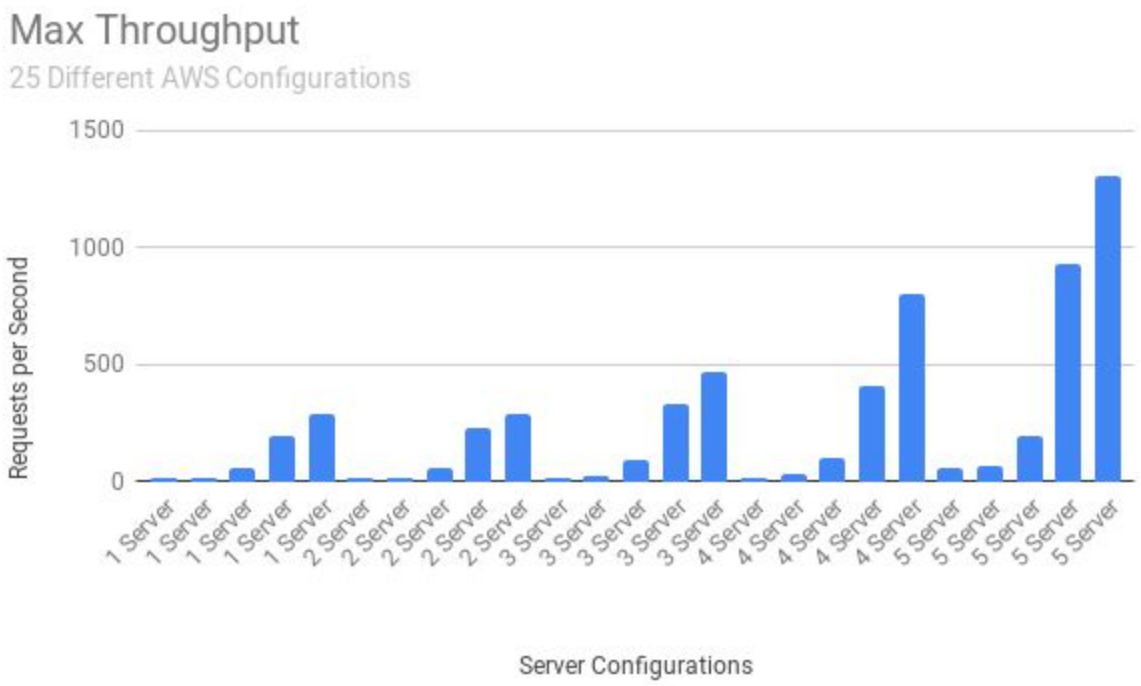
Followup

I would like to do 3 additional sets of benchmarks to clarify some tentative findings from this report

- Run tests comparing multiple EC2 and RDS database setups to confirm RDS speed findings and also record MySQL cluster data.
- Run more tests comparing multiple Redis and Memcache setups to provided more clarity on what setups are faster or cheaper.
- Run tests comparing response time at optimal loads instead of maximum loads, to see which options perform best when not under heavy load.
- Setup more specific cost setups to track costs 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