**Performance Evaluation Report**

CSE 503 Rapid Prototype Development and Creative Programming

Devanshu Save – 451662

Shengmin Huang – 451881

The experiments selected as a part of these evaluation report are –

1. Apache web server performance evaluation on different AWS instance types
2. MySQL Performance evaluation on different AWS instance types

We have used two AWS instances for the running the experiments. The configurations are tabulated below [1]

|  |  |  |
| --- | --- | --- |
| **Instance type**  **System parameters** | **t2.micro** | **t2.xlarge** |
|  |  |  |
| vCPUs | 1 | 4 |
| Baseline Performance | 10% | 90% |
| Memory (GiB) | 1 | 16 |

**Experiment #1 - Apache web server performance evaluation on different AWS instance types**

* **Environmental Setup**:

The webserver benchmarking tool we are using is ApacheBench, which can be run from linux command line ab. Since the ApacheBench is installed automatically with apache, no complicated setup is needed for the environment if your apache is correctly installed.

Site: /~Lars/md3/ (Dummy page on my instance)

Web Server: Apache/2.4.25

1. Running test on t2.micro instance

**Applied command line:**

This command will run for 1000 requests in total with 10 current requests, and output the data log to out.data file.

**Result:**

Document Path: /~Lars/md3/

Document Length: 781 bytes

Concurrency Level: 10

Time taken for tests: 6.684 seconds

Complete requests: 1000

Failed requests: 0

Total transferred: 1055000 bytes

HTML transferred: 781000 bytes

Requests per second: 149.62 [#/sec] (mean)

Time per request: 66.836 [ms] (mean)

Time per request: 6.684 [ms] (mean, across all concurrent requests)

Transfer rate: 154.15 [Kbytes/sec] received

Connection Times (ms)

min mean [+/-sd] median max

Connect: 32 33 2.6 33 90

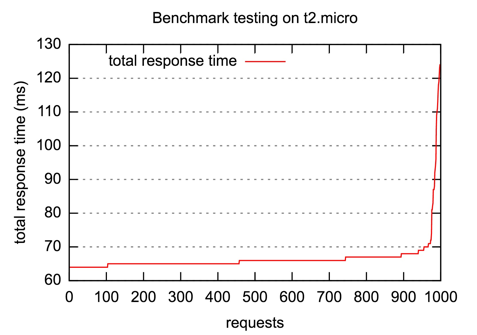
Processing: 31 34 5.2 33 91

Waiting: 31 34 5.2 33 91

Total: 63 67 5.9 66 124

**Observation:**

It can be observed that Apache can serve around 149 requests per second and the mean time to send one request is 6.684 ms. Visualization of “requests v. total response time”:



Next, we increase the current request to 50 and remain the other parameters to have the command line as follow:

And the result is:

Document Path: /~Lars/md3/

Document Length: 781 bytes

Concurrency Level: 50

Time taken for tests: 2.974 seconds

Complete requests: 1000

Failed requests: 0

Total transferred: 1055000 bytes

HTML transferred: 781000 bytes

Requests per second: 336.25 [#/sec] (mean)

Time per request: 148.698 [ms] (mean)

Time per request: 2.974 [ms] (mean, across all concurrent requests)

Transfer rate: 346.43 [Kbytes/sec] received

Connection Times (ms)

min mean [+/-sd] median max

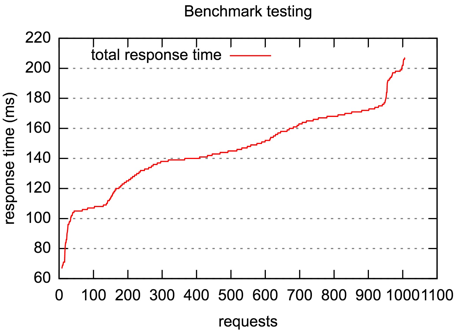
Connect: 32 35 6.4 34 104

Processing: 35 111 24.9 111 172

Waiting: 35 111 25.0 110 172

Total: 67 146 25.4 145 207

The number of request/second is now 336 and the mean time to send one request is 2.974 ms. These numbers are half of the amount when the current request number was set as 10. And the new visualization of requests vs total response time is as shown below:



In our expectation, the new numbers should be 5 times smaller than the original numbers instead of merely 2 times since the current requests are 5 times larger. We conclude the reason to be the type of AWS instance. The current instance type is t2.micro, which has limited memory of 1 GiB. Such limited memory may affect the processing of server, so we decided to change the type to t2.xlarge, with 16 GiB of memory, to see whether it will work as our expectation.

Running test on t2.xlarge instance

**Applied command line:**

This command runs for 1000 requests in total with 10 current requests.

**Results:**

Document Path: /~Lars/md3/

Document Length: 781 bytes

Concurrency Level: 10

Time taken for tests: 6.612 seconds

Complete requests: 1000

Failed requests: 0

Total transferred: 1055000 bytes

HTML transferred: 781000 bytes

Requests per second: 151.24 [#/sec] (mean)

Time per request: 66.122 [ms] (mean)

Time per request: 6.612 [ms] (mean, across all concurrent requests)

Transfer rate: 155.81 [Kbytes/sec] received

Connection Times (ms)

min mean [+/-sd] median max

Connect: 32 33 3.7 33 95

Processing: 32 33 3.4 33 96

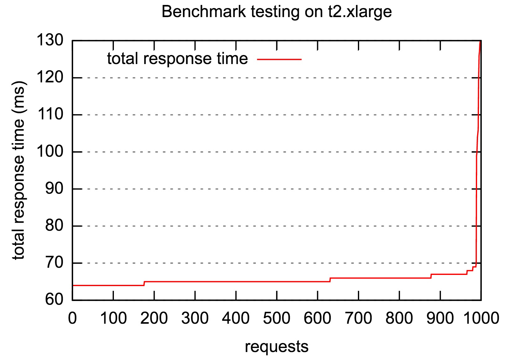
Waiting: 31 33 3.4 32 96

Total: 63 66 5.1 65 130

Observation:

It can be observed that Apache can serve around 151 requests per second and the mean time to send one request is 6.612 ms.

Visualization of “requests v. total response time”:



From the visualization and observation, we can see that there the test result on a different instance type is pretty much the same as before. And what will be the case if we change some of the parameters?

Applied command:

this command increases the current requests to 50, and the result is:

Document Path: /~Lars/md3/

Document Length: 781 bytes

Concurrency Level: 50

Time taken for tests: 3.041 seconds

Complete requests: 1000

Failed requests: 0

Total transferred: 1055000 bytes

HTML transferred: 781000 bytes

Requests per second: 328.82 [#/sec] (mean)

Time per request: 152.057 [ms] (mean)

Time per request: 3.041 [ms] (mean, across all concurrent requests)

Transfer rate: 338.78 [Kbytes/sec] received

Connection Times (ms)

min mean [+/-sd] median max

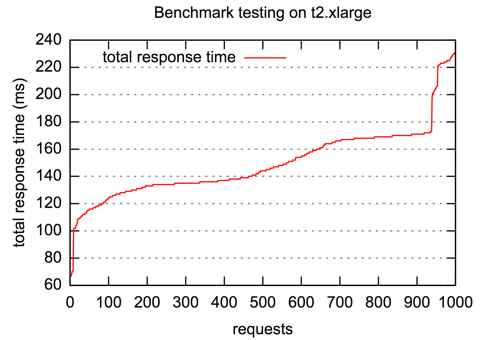
Connect: 32 34 5.6 33 94

Processing: 33 115 25.1 111 197

Waiting: 33 115 25.1 111 197

Total: 66 150 26.2 144 231

With the 328 requests/second and the 3.041 ms of mean time to send one request, which are also very close to the results from the instance type with much smaller memory. The visualization of it states the same case as well:



Even after the number of total request is increased, the two instance types have the similar results.

Putting all results together we have:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AWS Instance Type** | **Total Request #** | **Current Request #** | **Requests per Second** | **Time per Request (ms)** |
| t2.micro | 1000 | 10 | 149.62 | 6.684 |
| t2.xlarge | 1000 | 10 | 151.24 | 6.612 |
| t2.micro | 1000 | 50 | 336.25 | 2.947 |
| t2.xlarge | 1000 | 50 | 328.82 | 3.041 |
| t2.micro | 10000 | 50 | 631.30 | 1.584 |
| t2.xlarge | 10000 | 50 | 644.64 | 1.551 |

By comparing the results from evaluation different AWS instance types, we find that the there is almost no major difference. There could be to two possible reasons of it. First, the application we are testing is very easy to process, therefore the difference of the processing results from different AWS instances is too little to be observed. However, if that is the case, it is hard to explain why after the current request number was enlarged five times yet the results did not change as expect. This problem leads to the second possibility, when the server was dealing with the requests, it met a bottleneck.

**Experiment #2 - MySQL Performance evaluation on different AWS instance types**

In this experiment, we are evaluating the number of operations/queries that a mysql database can handle on two different instances of AWS i.e. t2.micro and t2.xlarge

The parameters that we are looking to evaluate are the number of requests (or queries) that the instance can support for various multithreaded configurations.

* **Environment Setup**

The tests were performed using Sysbench, which is one of the most common benchmarking tools. It can be used for CPU performance test, OLTP performance test and file I/O tests. Here, we use the Sysbench for the OLTP performance tests.We make

The following steps were followed to install Sysbench

header file was not located while running the command. This was resolved by running command to install dependencies.

A database was created with a table having 1000000 records

Now, we run 2 separate tests – one with read-only operations and the other with both read-write operations.

* **Results**

1. **Read-only operations**

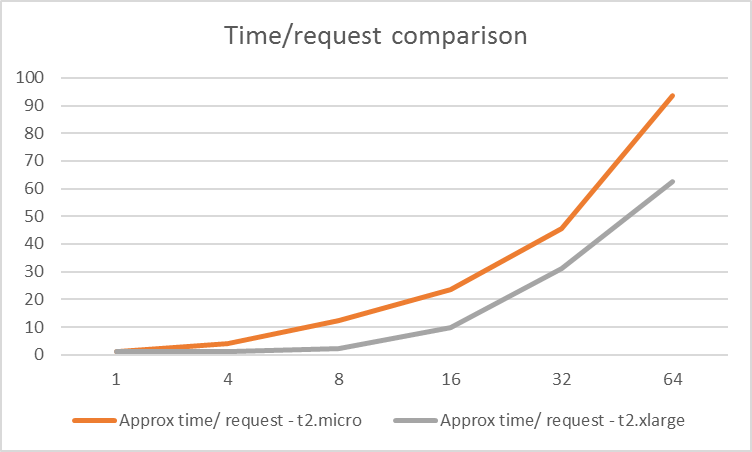
Running test to execute Read-only operations for multiple configurations for number of threads [2]



Next, we evaluate the performance of a t2.xlarge instance. The environment setup was done to change the existing instance to the new one. The tests were run and the results obtained are plotted next.



The approx. time per request increases as we increase the number of threads for both the instances. The results clearly show that more number of queries are serviced in a fixed time interval by t2.xlarge as compared to t2.micro. The comparison of time required (in ms) per request vs the number of threads used in the setup for both the instances is shown below.



1. **Read + Write operations**

The following command was executed to trigger a test for read-write operations on the AWS instance.

The results for the above command are tabulated next:

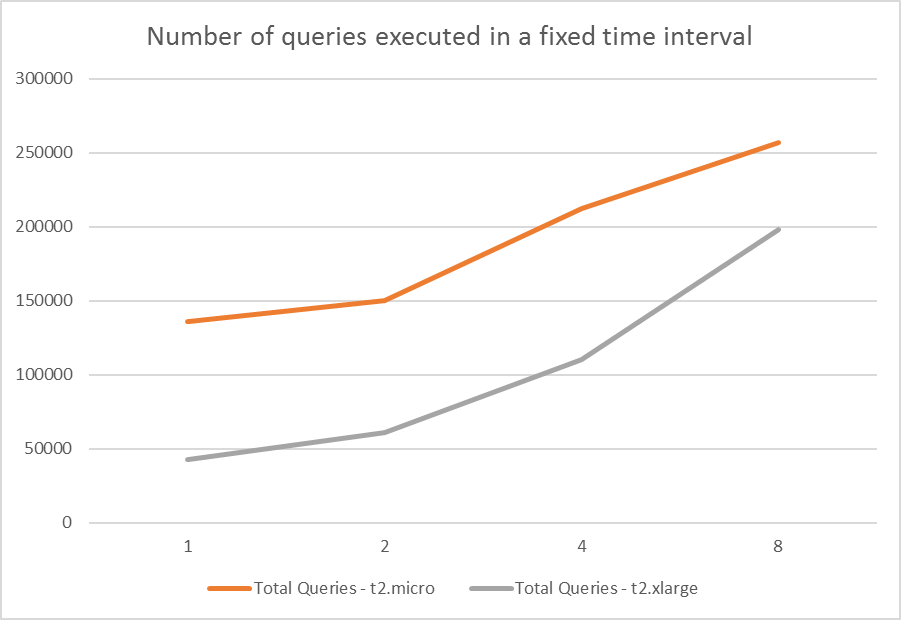


The same set of tests were run on a t2.xlarge instance and the results are tabulated below.



Now, we plot the number of queries executed vs number of threads in 20s for each of the two instances. This graph proves that the t2.micro instance executes more number of queries than the t2.xlarge instance. From previous graph (having read-only operations), we know that, t2.xlarge outperforms t2.micro.

Based on the above two observations, we can conclude that the Write operation is executed slowly on t2.xlarge as compared to t2.micro instance of AWS.



It is clear from the graph that, as the number of threads increase, the number of queries that can be executed in time , increase too. However, it is not possible to keep increasing the number of threads. We discuss this in the bottlenecks section.

While testing for the read-write operations using the following query,

, we ran into Duplicate primary key issues while running this test on 16, 32 and 64 threads for both the t2.micro and t2.xlarge instances. Hence, the tests were executed on 1, 2, 4, 8 threads’ configurations.

* **Bottlenecks:**

In case of read-only operations, the database gave an error when attempting to create 256 or more threads. The error received was “FATAL: error 1040: Too many connections”. Thus, increasing the number of threads to execute more number of operations per second results in a system bottleneck as the number of threads increase. Similarly, increasing the number of threads caused ‘Duplicate Entry for Primary Key’ Error while performing multi-threaded Read-write operations. This is another potential system bottleneck. These two factors need to be considered while determining multithreaded configurations for the AWS instance to be used for your application.

**CONCLUSION**

The evaluations of the above two experiments give us a clear picture regarding the capabilities of each instance based on the selected parameters. Hence, the decision of selecting a configuration depends on the application that the server or database needs to handle. An analysis of expected number of users, amount of memory required, estimated number of database requests that the instance should be able to handle, etc. should be done in the design phase before recommending any instance. As we have seen from the results, the t2.xlarge instance executes more number of read-only queries as compared to the t2.micro instance. However, if the requirement of the application is well and truly satisfied with a t2.micro instance, then a recommendation of t2.xlarge instance would be futile resulting in extra costs.

During this course, the applications built had very low system requirements in terms of number of users, page requests to the server, database queries, etc. The performance of t2.micro instance is comparable to the high end t2.xlarge instance for such requirements. Hence, we recommend the t2.micro instance for all similar applications. Another factor that needs to be considered is scaling. If the application is expected to be scaled in future, then cost of migration to a new instance should be evaluated. If such a cost is cheap and the process is simple, then, the recommendation would be for the instance that satisfies the current requirements. However, if the cost of migration is high, then the best approach (cheaper/easier) should be taken.

References

[1] <https://aws.amazon.com/blogs/aws/new-t2-xlarge-and-t2-2xlarge-instances/>

[2] <https://wiki.mikejung.biz/Sysbench>

[3] <http://lowrank.net/gnuplot/datafile-e.html>

[4] <http://www.bradlanders.com/2013/04/15/apache-bench-and-gnuplot-youre-probably-doing-it-wrong>

[5] <http://infoheap.com/ab-apache-bench-load-testing/>