CSE 503S Performance Evaluation Report

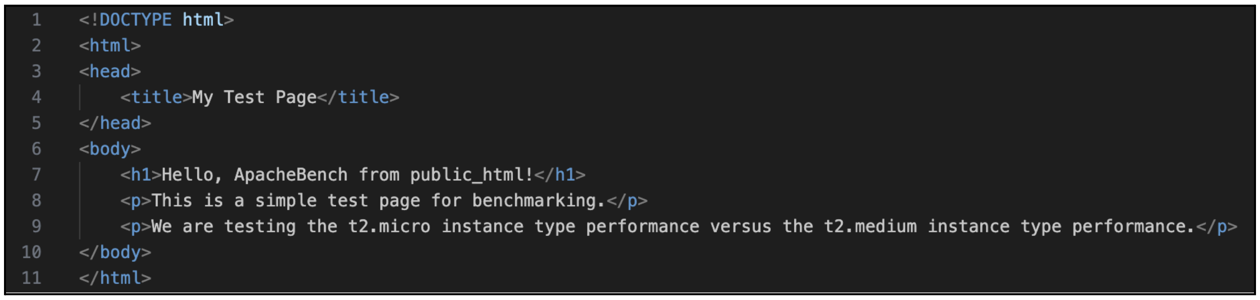
A. Viviano

December 6, 2024

In this report, two experiments will be described in order to analyze different aspects of two different AWS instance types. The first AWS instance type was t2.micro; the same instance type used for all of CSE 503S. The second AWS instance type was t2.medium. All data generated and used for every plot seen below can be seen in the associated Github repository for this assignment.

Experiment 1

For Experiment 1, Apache was installed on both servers, and benchmark tests were done using ApacheBench. The tests by ApacheBench were done on the same simple file in both instances: index.html. The content in index.html can be seen in *Figure 1* below.



```html
<!DOCTYPE html>
<html>
<head>
    <title>My Test Page</title>
</head>
<body>
    <h1>Hello, ApacheBench from public_html!</h1>
    <p>This is a simple test page for benchmarking.</p>
    <p>We are testing the t2.micro instance type performance versus the t2.medium instance type performance.</p>
</body>
</html>
```

Figure 1: Tested File: index.html Content

To test how the different instance types performed when transferring the HTML data, this ApacheBench command format was used on both instance types:

```
ab -n 500 -c 100 http://ec2-107-20-79-252.compute-1.amazonaws.com/~ec2-user/
```

In both cases, the data was saved to their respective files ("out_data_t2.micro.csv" and "out_data_t2.medium.csv"), and this data was then opened in Google Sheets, and used to plot the following graphs (*Figure 2* and *Figure 3*).

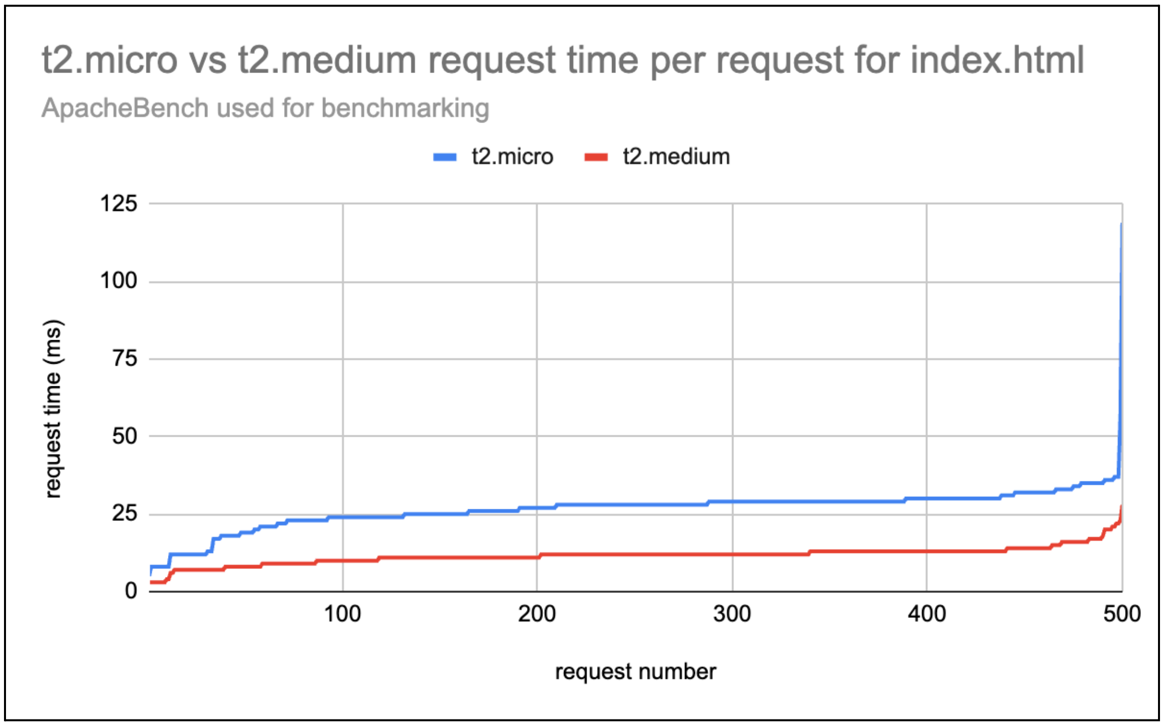


Figure 2: t2.micro vs. t2.medium request times (ms)

As can be seen in Figure 2, for each request, the t2.medium instance consistently took less time to complete a request. This is likely due to several factors that differentiate the two instances. One of these factors is that the t2.medium instance has 2 vCPUs while the t2.micro instance only has 1 vCPU. The number of vCPUs affects the burst performance of each instance. When an instance has better burst performance, it is better equipped to handle sudden increases in workload. They also acquire more CPU credits at a higher rate which allows them to work at that increased performance above the baseline for a longer period of time (less throttling). Additionally, t2.medium instances also have 4 GiB of RAM while t2.micro only has 1 GiB. This means that the t2.memium instance is less likely to run out of RAM or have to use slower disk space memory. Since the t2.micro instance has much less RAM, it is more likely to have to resort to using disk space when processing more requests since it will run out of RAM faster. All of these factors can easily contribute to the fact that the t2.medium instance processes requests much quicker than the t2.micro instance.

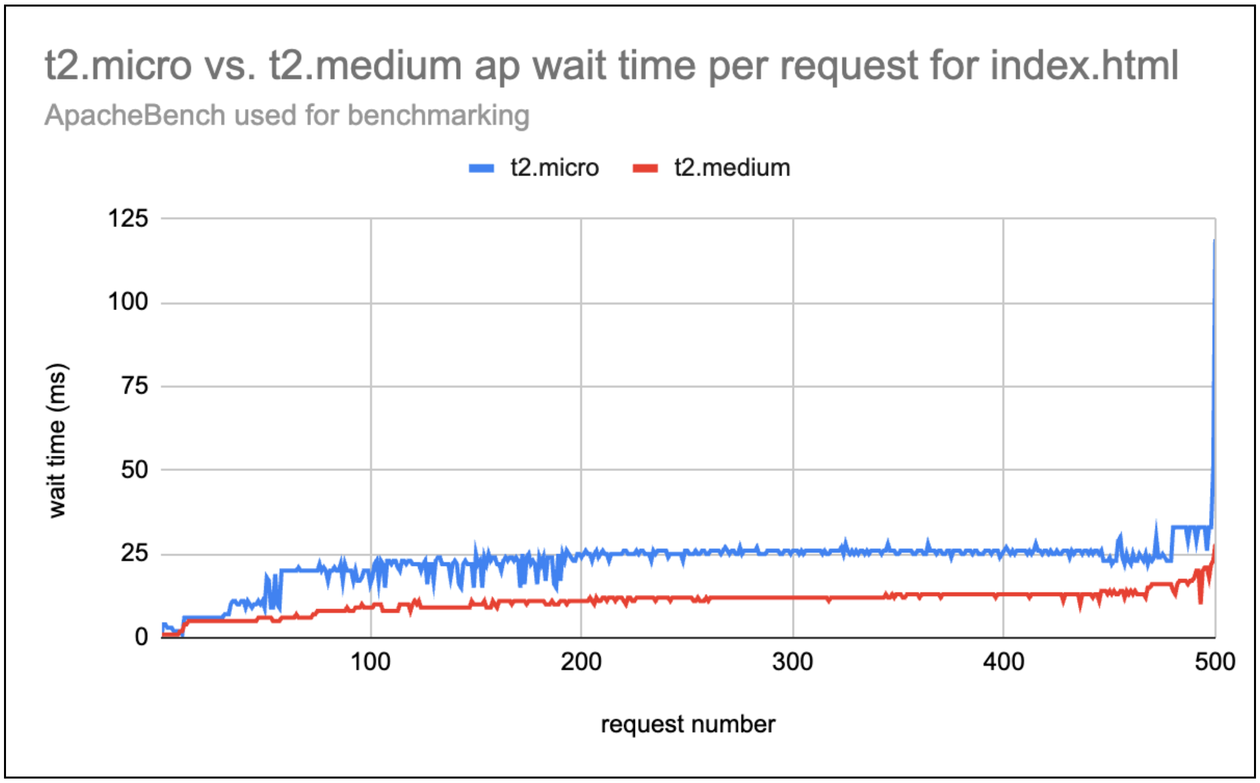
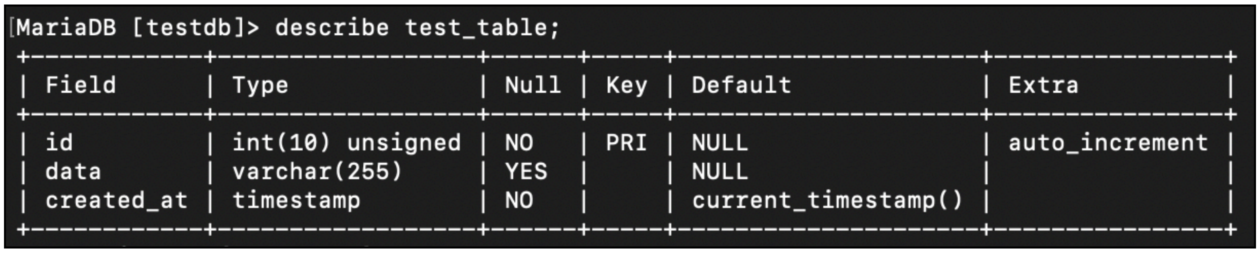


Figure 3: t2.micro vs. t2.medium wait times (ms)

Similarly to the request number versus request time comparison, the t2.medium instance experiences much less wait time per request than the t2.micro instance. Additionally, it should be noted that there is also much less variation in the plot for t2.medium when compared to the plot of t2.micro. As described in the previous data, t2.medium can handle larger increases in requests because of its better burst performance (higher vCPU) as well as its larger amount of RAM. The higher vCPU means that the t2.medium instance is able to handle the quickly incoming requests for longer periods of time without waiting, and the larger amount of RAM means that it resorts to using the slower disk memory much less often than the t2.micro instance needs to. This allows for the m2.medium plot to perform much more consistently, having much less frequent and small spikes than those seen in the t2.micro plot.

Experiment 2

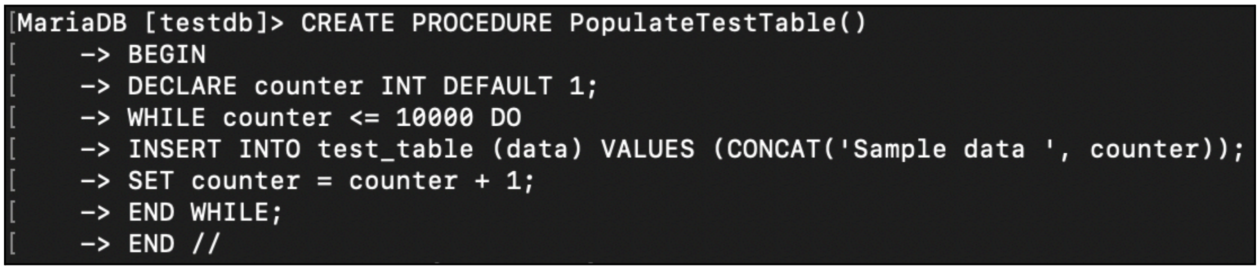
The second experiment consisted of testing the performance of mySQL on each of the AWS instance types. Naturally, mySQL was downloaded to complete this on both instances. After that, a new database "testdb" was created along with a new table "test_table" for each of the instances. A description of test_table can be seen below in *Figure 3*.



```
[MariaDB [testdb]> describe test_table;
+------------+------------------+------+-----+---------------------+----------------+
| Field      | Type             | Null | Key | Default             | Extra          |
+------------+------------------+------+-----+---------------------+----------------+
| id         | int(10) unsigned | NO   | PRI | NULL                | auto_increment |
| data       | varchar(255)     | YES  |     | NULL                |                |
| created_at | timestamp        | NO   |     | current_timestamp() |                |
+------------+------------------+------+-----+---------------------+----------------+
```

Figure 3: test_table Description

Again for both instances, each table was populated with 10000 sample data values in the data column. Each sample data consisted of the phrase 'sample data ' concatenated with a number representing the number of sample data that it was. This was completed using a custom stored procedure called "PopulateTestTable()". The code to create this stored procedure can be seen below in *Figure 4*.



```
[MariaDB [testdb]> CREATE PROCEDURE PopulateTestTable()
[    -> BEGIN
[    -> DECLARE counter INT DEFAULT 1;
[    -> WHILE counter <= 10000 DO
[    -> INSERT INTO test_table (data) VALUES (CONCAT('Sample data ', counter));
[    -> SET counter = counter + 1;
[    -> END WHILE;
[    -> END //
```

Figure 4: PopulateTestTable Stored Procedure Description

After the tables in each instance were populated, the benchmarking tool built into mySQL, "mysqlslap" was used to generate data based on queries to the tables in each of the instances. The format of the command below was used for generating data by using mysqlslap:

```
mysqlslap --user=root --password=password --host=localhost --concurrency=50--iterations=10
--create-schema=testdb --query="INSERT INTO test_table (data) VALUES ('Test data')"
--csv=benchmark_results.txt >> benchmark_results.txt
```

This command generated averages of several types of data, some of which included the type of query, the average query response time, and the number of concurrent connection threads. In order to get several data points, this command was used a total of 10 times to repeatedly get data for the average query response time for inserts of 'Test data' into the data column of test_table. Similarly, a modified version of the above command was used 10 times to get the average query response time for deletes from test_table. The results for each of these benchmark tests were saved to their respective files ("benchmark_results_t2.micro.txt" and "benchmark_results_t2.medium.txt"). This data was then used to plot box plots comparing the data for inserts and deletes between each instance type. The results can be seen in the figures below (*Figure 5* and *Figure 6*).

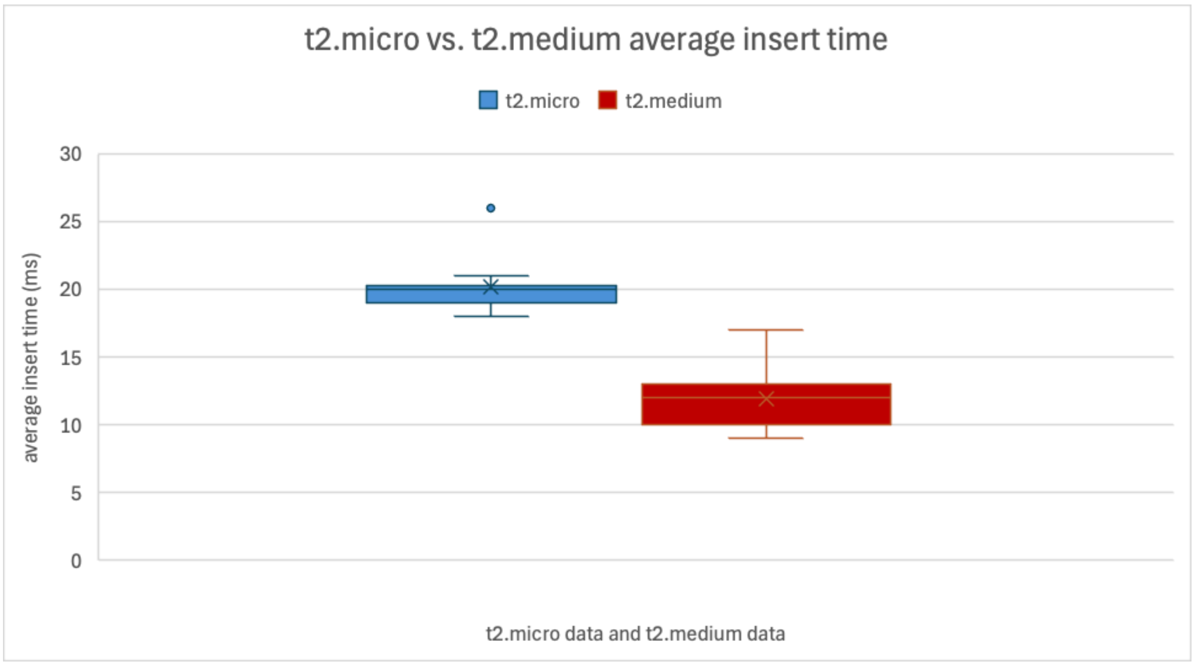


Figure 5: t2.micro vs. t2.medium average insert times (ms)

For many of the same reasons as described in Experiment 1, the t2.medium instance performs inserts into the mySQL table much faster than the t2.micro instance. The larger amount of vCPU for the t2.medium instance allows for it to process several insert queries much faster without slowing down since it has a higher burst performance and RAM than the t2.micro instance. It should be noted that the t2.micro instance has a high outlier while the t2.medium instance does not. This is consistent with the explanation that t2.micro is less likely to be able to handle sudden increases in workload for longer periods of time. Because of this, even when there are many query requests at once, the t2.micro instance cannot work above the baseline for long and throttles (drops back down to its baseline performance), causing longer wait times. Additionally, with less RAM, it is more likely to resort to using disk space causing longer wait times. Because of this, during times of high workload (the fast input of several insert queries), the t2.micro instance performance is much more sporadic and unreliable than the t2.medium instance, therefore, making it more likely to produce high outliers in the performance data.

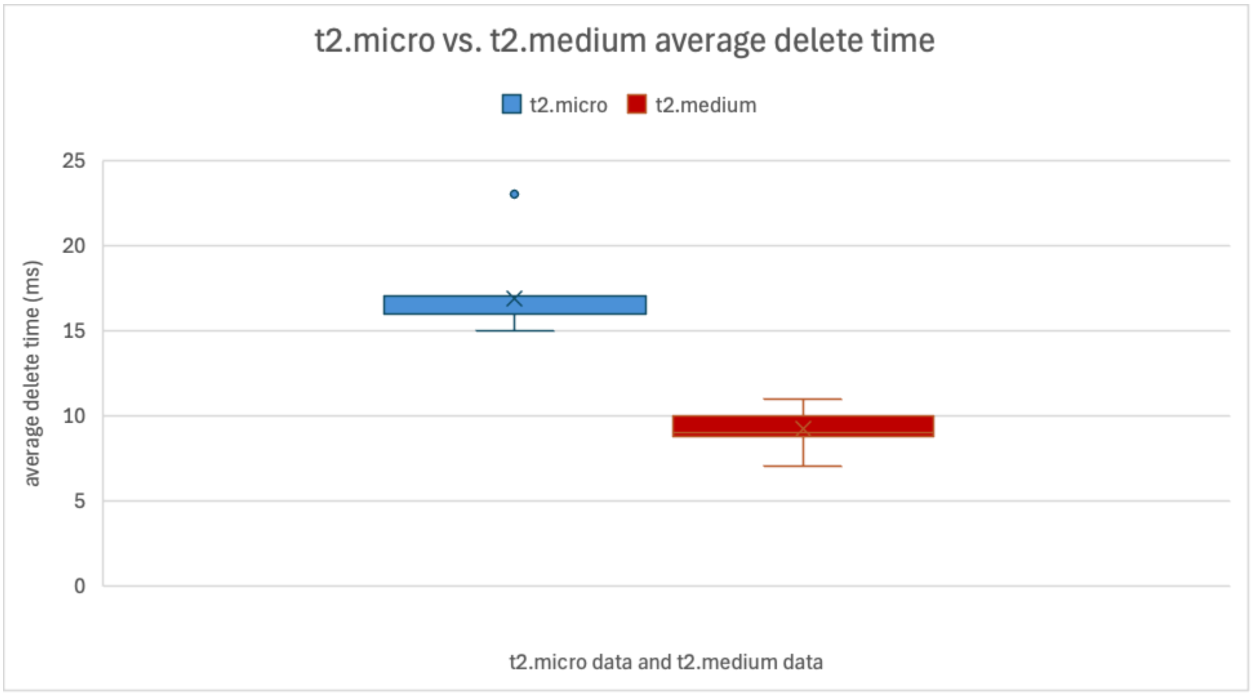


Figure 6: t2.micro vs. t2.medium average delete times (ms)

As can be seen in the figure above, the t2.medium instance performs delete queries to the mySQL table at a faster rate than the t2.micro instance. Once again, the higher vCPU and RAM amounts for the t2.medium instance allow for the delete queries to be processed more efficiently (faster, and with shorter wait times). The general inconsistency can again be seen in t2.micro based on the fact that it once again has a high outlier, while the t2.medium instance, also once again, does not have any outliers. Although it does not highlight the difference between the two instances, it can also be noted that the average delete times for both instances are shorter and have a smaller spread than their average insert times. This can likely be attributed to the fact that the insert queries in this experiment require 'checks' while the delete queries do not. For example, when inserting into test_table, every inserted string into the 'data' column requires a check that it also has a corresponding incremented ID, as well as a timestamp in their respective 'id' and 'created_at' columns. These checks are more likely to require the instances to use a larger amount of vCPU and RAM to be processed, and since the checks are not required for the delete queries (based on the setup of the tests in this experiment), the delete queries tend to be performed faster than the insert queries. Although the delete query does not require as much performance as the insert query, it can still be seen that the t2.micro instance had much longer times and higher variability than that of the t2.medium instance.

Conclusion

Overall, it can be concluded that the t2.medium AWS instance performs much better than the t2.micro AWS instance. Although both experience similar issues, the t2.micro instance experiences those issues much more often for longer periods of time. Its limited vCPU capacity causes it to throttle more often, and its smaller amount of RAM causes it to more often rely on slower memory on the disk. This causes slower performance when transferring HTML data as well as processing queries in mySQL on the t2.micro instance. In contrast, because of the larger vCPU and RAM capacity, the t2.medium instance is more likely to be able to handle peak workloads (like larger datasets or increased amounts of queries in mySQL) with much less throttling. Because of all these factors, the t2.micro instance would not be recommended unless either fast performance doesn't matter or it does not need to handle large datasets or high traffic. However, if the intention is to handle large datasets or high traffic, the t2.medium instance would be a much better choice.