Apache web server performance evaluation on different AWS instance types

**Setup**

Test the apache web server performance on t2.micro:

* Create a new EC2 instance : <http://aws.amazon.com/ec2/>
* Connect to the new EC2 instance t2.xlarge with SSH
* Install Apache on both EC2 instance
* $ sudo yum update
* $ sudo yum install httpd24
* $sudo /sbin/chkconfig --level 235 httpd on
* Obtain the ip address of original instance
* $ ifconfig
* Run the apache benchmark test on the new instance by the ip address of original one
* $ ab -c 100 -n 10000 <http://172.31.39.221/index.html>
* Monitor CPU and memory utilization on the server
* $top

Test the apache web server performance on t2.xlarge:

* Connect to the original EC2 instance t2.micro with SSH
* Obtain the ip address of new instance
* Enabling web access to the new EC2 instance
* Select security group on EC2 webpage
* Click on the Inbound Tab
* Add a new custom TCP rule with a port range of 80
* Click add new rule
* Run the apache benchmark test on the original instance by the ip address of new one
* $ ab -c 100 -n 10000 <http://172.31.27.82/index.html>
* Monitor CPU and memory utilization on the server
* $top

**Experiment Data**

|  |  |  |
| --- | --- | --- |
| Concurrent request | Pages/sec | Mean time/request |
| 100 | 5630.73 | 17.760ms |
| 200 | 2977.47 | 67.171ms |
| 300 | 2207.51 | 135.9ms |
| 400 | 1484.36 | 269.476ms |
| 500 | 745.69 | 670.522ms |
| 600 | 2772.72 | 216.394ms |
| 700 | 2749.95 | 249.058ms |
| 800 | 2668.58 | 299.785ms |
| 900 | 1392 | 646.518ms |
| 1000 | 1282.65 | 779.636ms |

Table1. Apache web server performance on t2.micro

|  |  |  |
| --- | --- | --- |
| Concurrent request | Pages/sec | Mean time/request |
| 100 | 6826.39 | 14.65ms |
| 200 | 5668.12 | 36.032ms |
| 300 | 5442.66 | 55.12ms |
| 400 | 4790.20 | 83.504ms |
| 500 | 4698.34 | 85.677ms |
| 600 | 5242.99 | 114.439ms |
| 700 | 5217.79 | 134.156ms |
| 800 | 5263.32 | 151.955ms |
| 900 | 5107.44 | 176.214ms |
| 1000 | 4935.81 | 202.601ms |

Table2. Apache web server performance on t2.xlarge

Figure 1. Pages per second on different AWS server

Figure 2. Mean Time per request on different AWS instance

**Experiment Results**

For all of the experiments, our group keep the number of requests to perform as constant in order to compare the number of multiple requests to make a time and mean time per request on different AWS instance types t2.micro and t2.xlarge. As we can see, AWS instance t2.xlarge can have a better performance than t2.micro in any tests.

Figure 1 illustrate the trend of pages per second on different AWS instance as the value of concurrent request increases. As is shown on the figure, we can conclude that Apache on t2.xlarge has a better performance to server more request per second to the machine running the ab tool and as the number of concurrent request becomes larger, the number of pages per second decreases slowly.

Figure 2 provides some data regarding mean time per request on the different AWS instance. It is clearly from the figure that t2.xlarge spent less mean time requesting than t2.micro. In addition, as the number of concurrent request becomes larger, the number of mean time per request increases gradually.

**Bottleneck**

* Initialization problem – need run a test several times to normalize the value
* Overload with request – test too much results in the outcome inaccurately

Hence, we need faster CPU and more memory to fix all these problems.

**Recommendation**

If we don’t consider the cost of instance, I will recommend to use t2.xlarge since it really has a better performance on Apache benchmark test. However, if there are not too many request in your web and 1GB memory size is enough for your web application, t2.micro is a better option due to the price is much cheaper than t2.xlarge.

MySQL Performance evaluation on different AWS instance types

**Setup**

* Create a new EC2 instance : <http://aws.amazon.com/ec2/>
* Connect to the new EC2 instance t2.xlarge with SSH
* Install MySQL 5.7 on both EC2 instance
* $ sudo yum update
* $ sudo yum install mysql57-devel mysql57-server
* Install Sysbench on both EC2 instance
* $ sudo yum install –y libtool
* $ wget https://github.com/akopytov/sysbench/archive/master.zip
* $ unzip master.zip
* $ cd sysbench-master
* $ ./autogen.sh
* $ ./configure
* $ make
* $ sudo make install
* Create a new database
* $ create database sysbench;
* Create a new user
* $ create user 'wustl\_sysbench'@'localhost' identified by 'wustl\_pass';
* Grant Access
* $ grant all on \*.\* to wustl\_sysbench@'localhost' with grant option;
* Prepare a oltp-read-only test table
* $ sysbench --db-driver=mysql --mysql-user=wustl\_sysbench --mysql-password=wusl\_pass --mysql-db=sysbench --table\_size=1000000 --tables=1 --time=120 --threads=8 --rand-type=uniform /usr/local/share/sysbench/oltp\_read\_only.lua prepare
* Read-Only Test
* $ sysbench --db-driver=mysql --mysql-user=wustl\_sysbench --mysql-password=wusl\_pass --mysql-db=sysbench --table\_size=1000000 --tables=1 --time=120 --threads=8 --rand-type=uniform /usr/local/share/sysbench/oltp\_read\_only.lua run
* Cleanup the created table
* $ sysbench --db-driver=mysql --mysql-user=wustl\_sysbench --mysql-password=wusl\_pass --mysql-db=sysbench --table\_size=1000000 --tables=1 --time=120 --threads=8 --rand-type=uniform /usr/local/share/sysbench/oltp\_read\_only.lua cleanup
* Prepare a oltp-read-write test table
* $ sysbench --db-driver=mysql --mysql-user=wustl\_sysbench --mysql-password=wusl\_pass --mysql-db=sysbench --table\_size=1000000 --tables=1 --time=120 --threads=8 --rand-type=uniform /usr/local/share/sysbench/oltp\_read\_only.lua prepare
* Read-Write Test
* $ sysbench --db-driver=mysql --mysql-user=wustl\_sysbench --mysql-password=wusl\_pass --mysql-db=sysbench --table\_size=1000000 --tables=1 --time=120 --threads=8 --rand-type=uniform /usr/local/share/sysbench/oltp\_read\_only.lua run
* Cleanup the created table
* $ sysbench --db-driver=mysql --mysql-user=wustl\_sysbench --mysql-password=wusl\_pass --mysql-db=sysbench --table\_size=1000000 --tables=1 --time=120 --threads=8 --rand-type=uniform /usr/local/share/sysbench/oltp\_read\_only.lua cleanup
* Prepare a oltp-write-only test table
* $ sysbench --db-driver=mysql --mysql-user=wustl\_sysbench --mysql-password=wusl\_pass --mysql-db=sysbench --table\_size=1000000 --tables=1 --time=120 --threads=8 --rand-type=uniform /usr/local/share/sysbench/oltp\_read\_only.lua prepare
* Write-Only Test
* $ sysbench --db-driver=mysql --mysql-user=wustl\_sysbench --mysql-password=wusl\_pass --mysql-db=sysbench --table\_size=1000000 --tables=1 --time=120 --threads=8 --rand-type=uniform /usr/local/share/sysbench/oltp\_read\_only.lua run
* Cleanup the created table
* $ sysbench --db-driver=mysql --mysql-user=wustl\_sysbench --mysql-password=wusl\_pass --mysql-db=sysbench --table\_size=1000000 --tables=1 --time=120 --threads=8 --rand-type=uniform /usr/local/share/sysbench/oltp\_read\_only.lua cleanup

**Experiment Data**

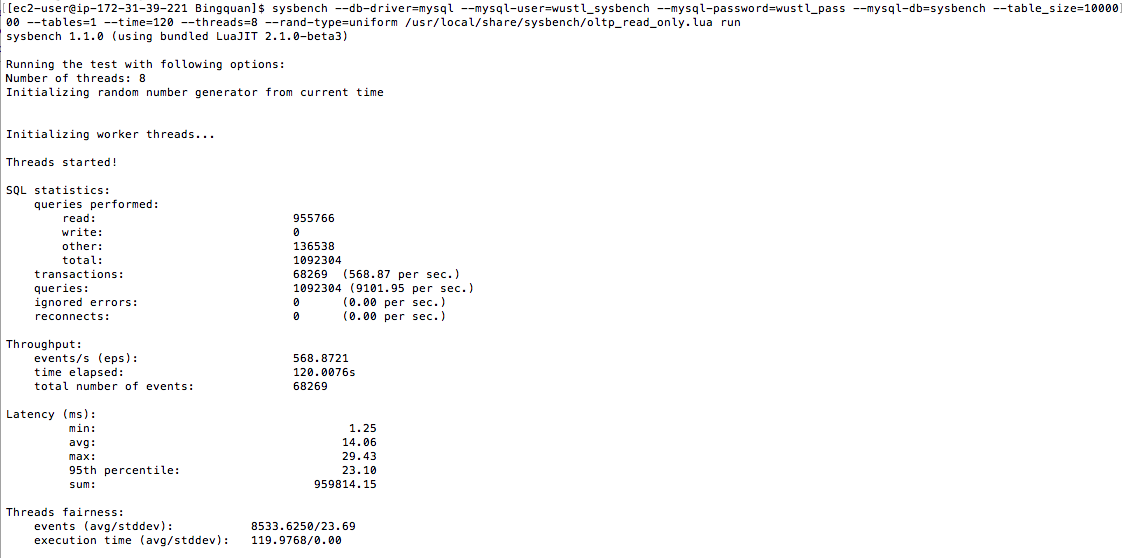
* Oltp-Read-Only Test by different EC2 instances

Figure1.1 Oltp-Read-Only Test by AWS instance t2.micro

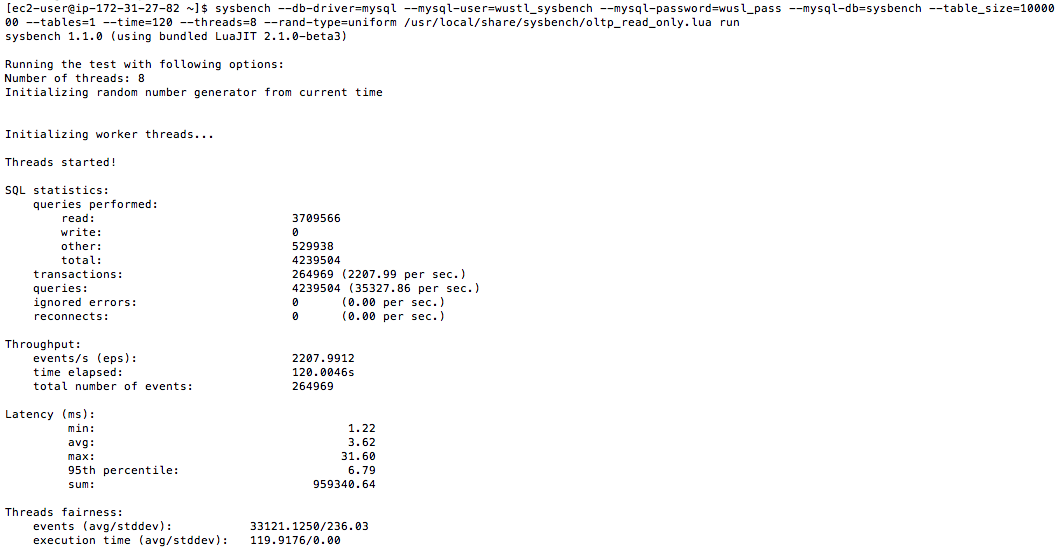
* 

Figure1.2 Oltp-Read-Only Test by AWS instance t2.xlarge

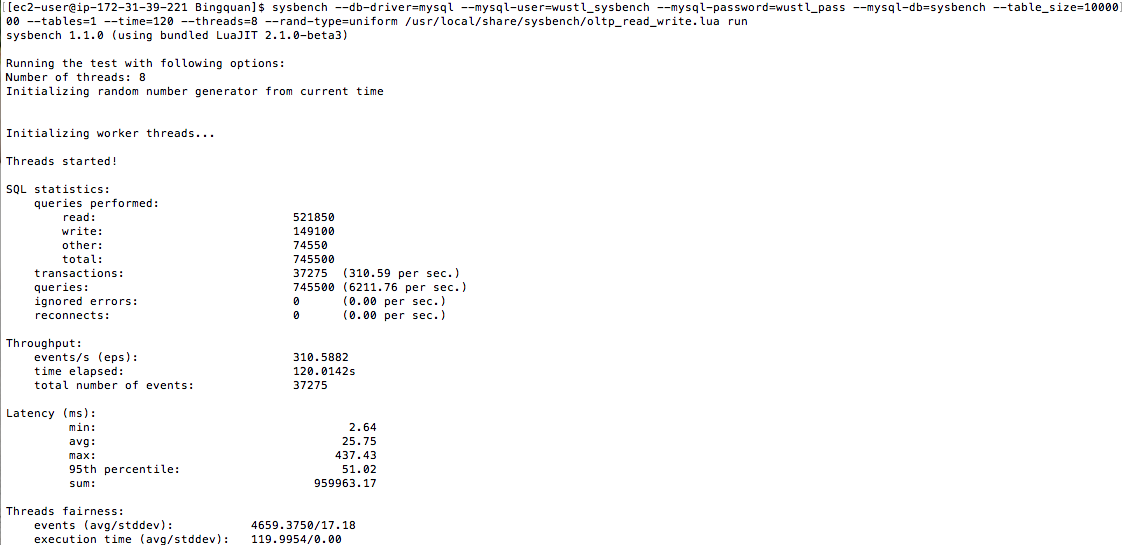
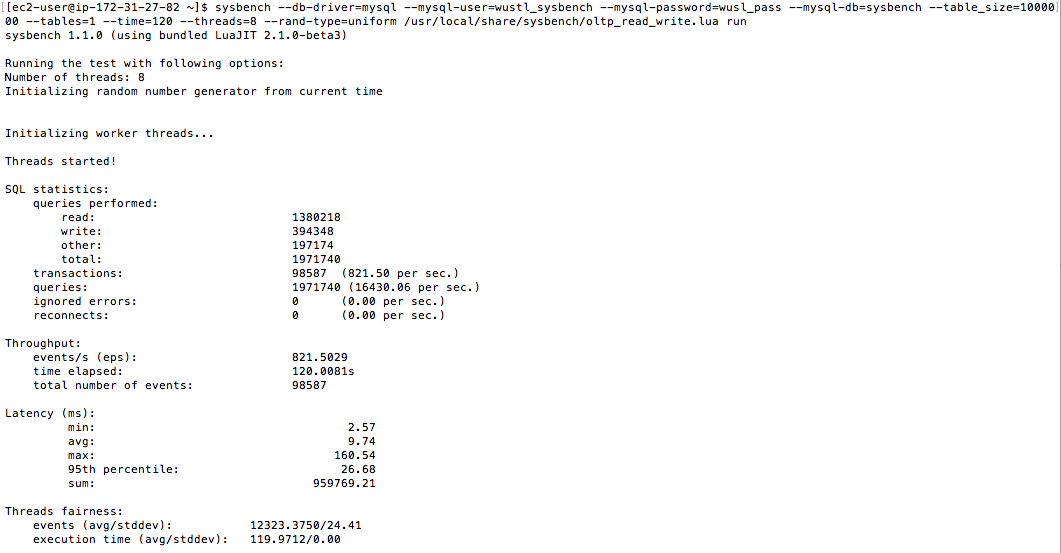
* Oltp-Read-Write Test by different EC2 instances
* Figure2.1 Oltp-Read-Write Test by AWS instance t2.micro
* 

Figure2.2 Oltp-Read-Write Test by AWS instance t2.xlarge

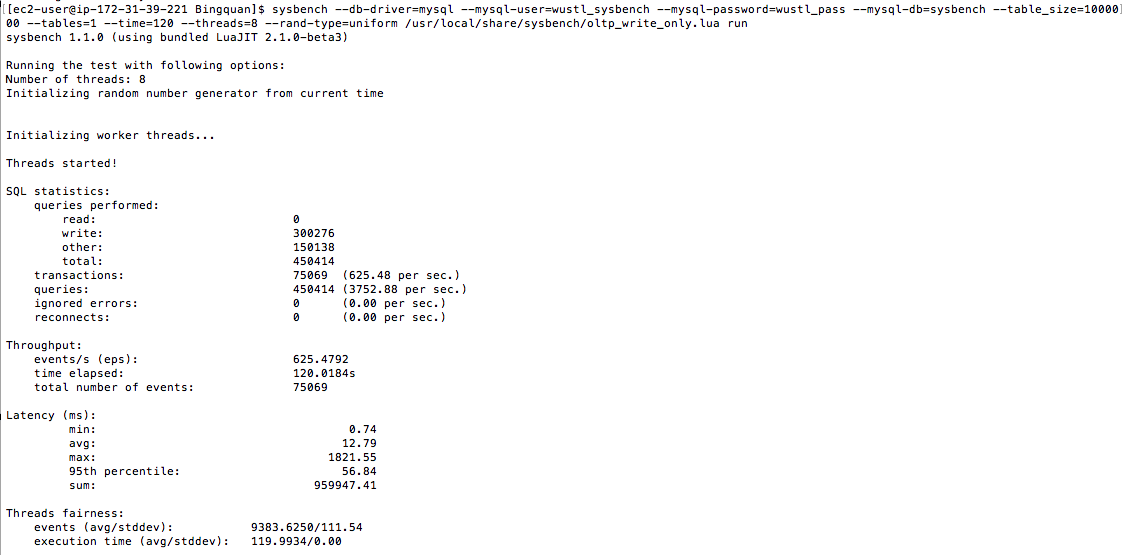
* Oltp-Write-Only Test by different EC2 instances

Figure3.1 Oltp-Write-Only Test by AWS instance t2.micro

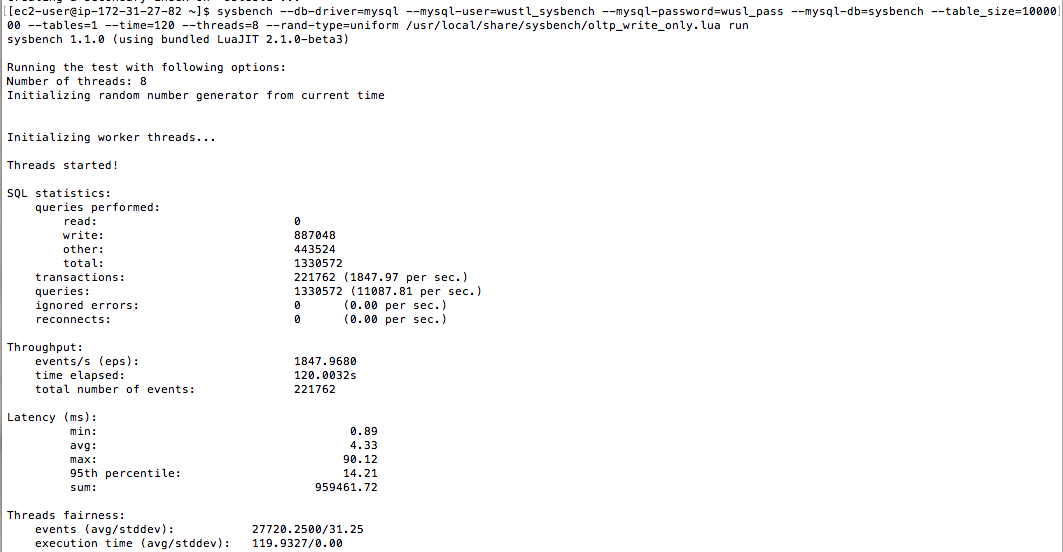
* 

Figure3.2 Oltp-Write-Only Test by AWS instance t2.xlarge

**Experiment Results**

For all three experiments, our group keep the value of table size, time, threads and rand type as constant in order to compare MySQL performance on different AWS instance types t2.micro and t2.xlarge. As we can see, AWS instance t2.xlarge can have a better performance than t2.micro in any tests.

|  |  |  |  |
| --- | --- | --- | --- |
|  | T2.micro | T2.xlarge | Ratio(xlarge/xmicro) |
| Read queries | 955766 | 3709566 | 3.88 |
| Read queries/sec | 7964.72 | 30913.05 | 3.88 |
| Transaction | 68269 | 264969 | 3.88 |
| Transaction/sec | 568.87 | 22077.99 | 3.88 |

Table1.

Table 1 illustrates the changing portion of read queries and transactions from t2.micro to t2.xlarge. It can be concluded that t2.xlarge performed more read queries and transactions than t2.micro in 120 seconds.

|  |  |  |  |
| --- | --- | --- | --- |
|  | T2.micro | T2.xlarge | Ratio(xlarge/xmicro) |
| Read queries | 521850 | 1380218 | 2.64 |
| Read queries/sec | 4348.75 | 11501.81 | 2.64 |
| Write queries | 149100 | 394348 | 2.64 |
| Write queries/sec | 1242.5 | 3286.23 | 2.64 |
| Transaction | 37275 | 98587 | 2.64 |
| Transaction/sec | 310.59 | 821.50 | 2.64 |

Table2.

Table 2 shows that the difference between t2.micro and t2.xlarge on oltp-read-write test. It is clearly from the table that t2.xlarge can perform more read, write queries and transactions on this test.

|  |  |  |  |
| --- | --- | --- | --- |
|  | T2.micro | T2.xlarge | Ratio(xlarge/xmicro) |
| Write queries | 300276 | 887048 | 2.95 |
| Write queries/sec | 2502.3 | 7392.07 | 2.95 |
| Transaction | 75069 | 221762 | 2.95 |
| Transaction/sec | 625.48 | 1847.97 | 2.95 |

Table3.

Table 3 provides some data regarding MySQL performance of t2.micro and t2.xlarge on oltp-write-only test. As is shown in the table, we can conclude that t2.xlarge query and transact more times than t2.micro in 120 seconds.

**Bottleneck**

The bottleneck of the system is the size of memory and faster CPU to perform more queries and transactions. Comparing with 16GB t2.xlarge, t2.micro only has 1 GB memory. If our database have an tremendous size of data, t2.micro might be too small to store our data. In addition, from all figures in experiment data, we can see clearly that the average latency of t2.micro is much longer than t2.xlarge’s average latency. In other words, comparing with t2.micro, t2.xlarge can perform more queries or transactions in limited time.

**Recommendation**

The drawback of t2.xlarge is the cost of instance. The price of t2.micro instances starts at $0.0116 per hour ($8.47 per month) in the US east region. The price of t2.xlarge instances starts at $0.0928 per hour ($67.74 per month) in the US east region. The price of t2.xlarge is almost 8 times of t2.micro.

If we don’t consider the cost of instance, I will recommend to use t2.xlarge since it has a larger memory size and better vCPU. However, if there are not too many read or write queries and transactions in your web and 1GB memory size is enough for your web application, t2.micro is a better option due to the price is much cheaper than t2.xlarge.