CSE 503S Master Performance Evaluation

Student ID: 467207 & 458715

Choice of the experiments:

- Apache web server performance evaluation on different AWS instance types
- Apache/PHP and Node.js comparison on the same AWS instance type

Experient 1: Apache web server performance evaluation on different AWS instance types

1.Experiment setup

- We choose ApacheBench (ab) as the tool for this experiment.
- We choose four types of instances: t2.micro, t2.small, t2.medium and t2.large. We choose t2 since its characters fitting the personal service as we have used this semester.

Model	vCPU*	CPU Credits / hour	Mem (GiB)	Storage	Network Performance
t2.nano	1	3	0.5	EBS-Only	Low
t2.micro	1	6	1	EBS-Only	Low to Moderate
t2.small	1	12	2	EBS-Only	Low to Moderate
t2.medium	2	24	4	EBS-Only	Low to Moderate
t2.large	2	36	8	EBS-Only	Low to Moderate
t2.xlarge	4	54	16	EBS-Only	Moderate
t2.2xlarge	8	81	32	EBS-Only	Moderate

As for the chosen types, we want them to form a control experiment. From the

sheet in the above picture (coming from EC2 official website), we can see a list of characters which may affect the performance of EC2, where vCPU and Memory are believed to be the most vital one by us. We try to eliminate the potential affect of Network Performance by setting them all to Low to Moderate when CPU Credits/ hour does not affect performance. To do the control experiment, we choose (t2.micro, t2.small) and (t2.medium, t2.large). Each group have same vCPU but twice-over Memory which helps explore Memory, and the latter group has twice the value of vCPU as the former group. We cannot find a group of two types which have same memory for us to explore vCPU. So we choose the groups to have twice the size of vCPU for exploration of vCPU. Apart from these, Network Performance character could be important too, but we do not set groups to explore it because the type which has medium value of it also contains much higher value than other types in vCPU and Memory. We make all the Network Performance to 'Low to Medium'.

The more specific configuration is as the following sheet.

	vCPU	Memory
T2.micro	1	1
T2.small	1	2
T2.medium	2	4
T2.large	2	8

Sheet 1 - Configuration

2.Experiment process

We run the command according to the reference [2] in course Wiki, so the command is **"**ab -n 200 -c 20 XXX" and "ab -n 200 -c 100 XXX", where XXX represents the address of our each EC2 instance.

The number after -n represents the number of request in total, and the number after -c means the number of concurrent request.

3.Results and discussion

Requests Per Second

We list one of the results here as the following picture.

```
Benchmarking ec2-18-217-122-83.us-east-2.compute.amazonaws.com (be patient)
Completed 100 requests
Completed 200 requests
Finished 200 requests


Server Software:        Apache/2.4.34
Server Hostname:        ec2-18-217-122-83.us-east-2.compute.amazonaws.com
Server Port:            80

Document Path:          /~FeiyangYang/
Document Length:        1951 bytes

Concurrency Level:      100
Time taken for tests:   0.261 seconds
Complete requests:      200
Failed requests:        0
Total transferred:      427000 bytes
HTML transferred:       390200 bytes
Requests per second:    765.92 [#/sec] (mean)
Time per request:       130.562 [ms] (mean)
Time per request:       1.306 [ms] (mean, across all concurrent requests)
Transfer rate:          1596.91 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    6   5.5      8      12
Processing:     3   96  42.3    119     145
Waiting:        0   89  40.5    113     137
Total:          3  102  38.9    123     145

Percentage of the requests served within a certain time (ms)
  50%    123
  66%    129
  75%    130
  80%    131
  90%    137
  95%    138
  98%    141
  99%    143
 100%    145 (longest request)
```

Picture 1- Result ScreenShot

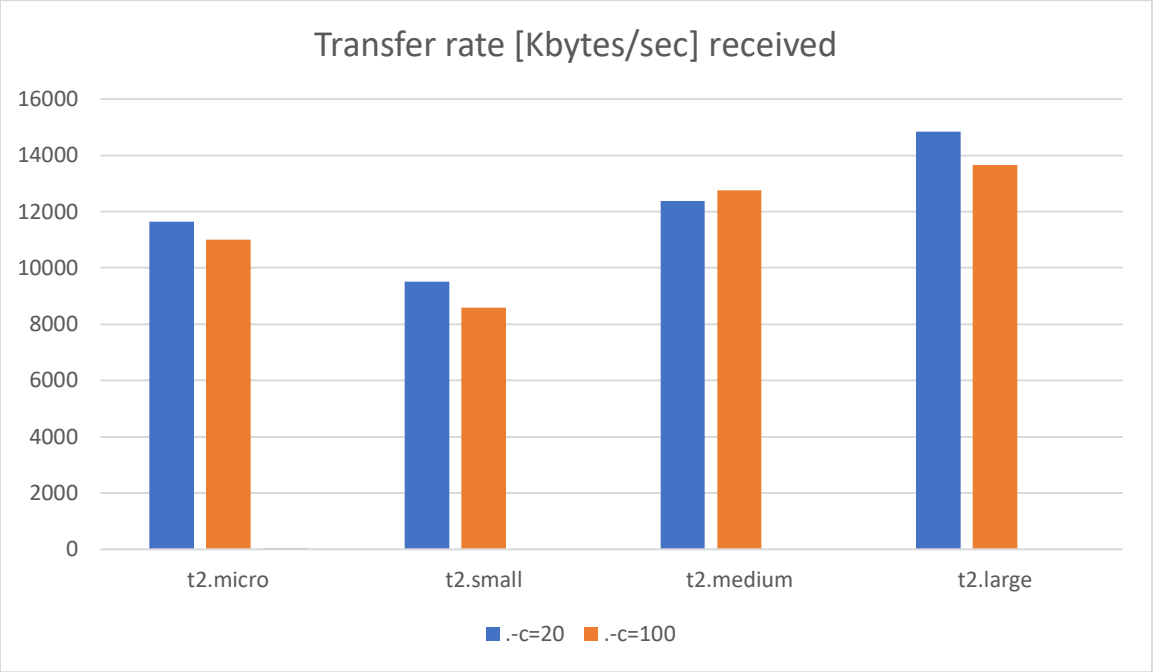
Result

Firstly, we try to analyze two elements in the result – “transfer rate [Kbytes/sec] received” and “request per second [#/sec] (mean)”. The following sheet demonstrates the raw data.

T2.micro	Requests per second [#/sec] (mean)	Transfer rate [Kbytes/sec] received
-c = 20	2304.04	11639.46
-c = 100	2178.51	11005.29
T2.small	Requests per second [#/sec] (mean)	Transfer rate [Kbytes/sec] received
-c = 20	2482.99	9507.63
-c = 100	2241.90	8584.47
T2.medium	Requests per second [#/sec] (mean)	Transfer rate [Kbytes/sec] received
-c = 20	3232.17	12376.29
-c = 100	3332.33	12759.84
T2.large	Requests per second [#/sec] (mean)	Transfer rate [Kbytes/sec] received
-c = 20	3873.87	14833.43
-c = 100	3564.17	13647.58

Sheet 2- Result Raw Data

To be more obvious, we make the following histograms.



Discussion of the result:

From the result, we can see that -c element has almost no effect on the processing speed and transfer speed of the EC2. In each type, the two results only vary in the range of 5% compared with 100 is five times larger than 20.

Then we turn into comparation of different types:

For “request per second”, as we run the two commands for the two groups, the result does not change so much -- even the biggest variance of the absolute value of the result is 750 in t2.small vs t2.medium, which only occupies 23.3% of the value of the t2.edium or 30.2% of t2.samll. Compared with the 200% devotion of both vCPU and Memory, the increase of the result is not sounding.

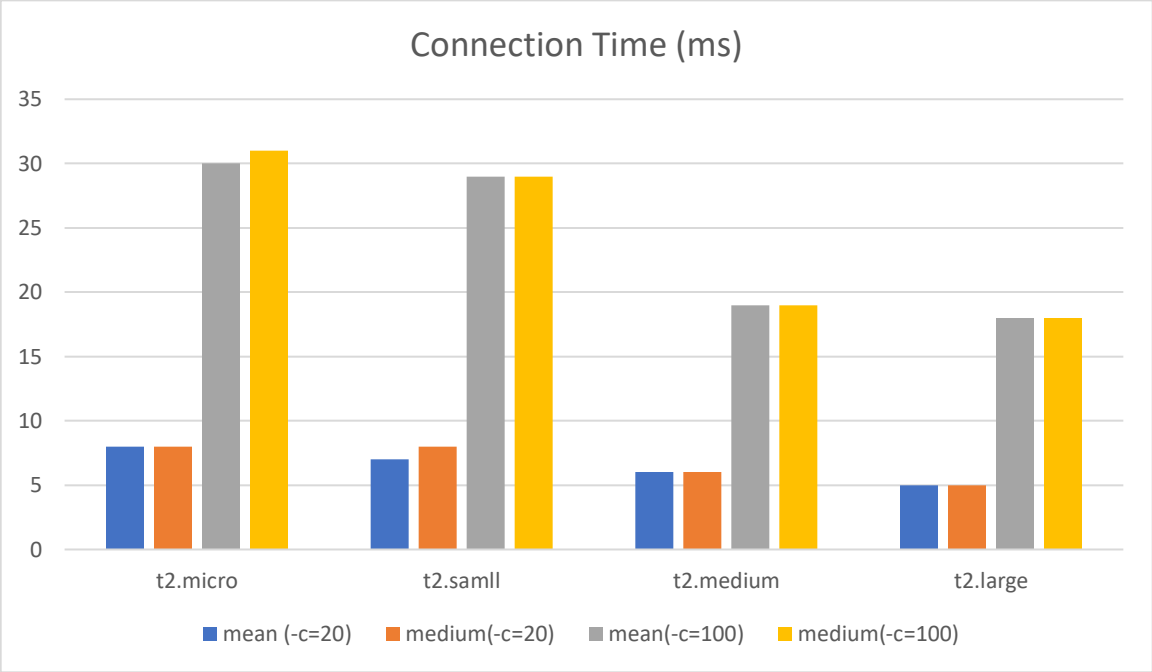
For “transfer rate”, we encounter the same issue that the result value only fluctuates in very small range compared with we set the vCPU or Memory variable quantity vary 200%. That means, we double the vCPU or Memory, or even both, but the instance varies not so much in transfer speed or the capacity of dealing with requests.

Though the above result seems disappointed, we can still find that increasing the absolute value of Memory has effect, especially in the second group. T2.medium which

has double both vCPU and Memory performances the most increasement ratio. And when we find that, when we set the same vCPU, and we increase Memory from 4 to 8, t2.large performs 20% better of t2.medium, comparing with the Memory increase from 1 to 2, t2.small only improves 8% in Request per Second but even worse in Transfer Rate.

• Connection time

Apart from the above, we find one more thing requires attention in the result – "connection time". This character has five fields: min, mean, [+/-sd], median and max. To analyze more average cases, we take mean and median into consideration here.



Discussion of the result:

From the picture, we can tell that when -c increases 5 times, the connection time also increases 3 to 4 times. And we can tell that in each group in (t2.micro,t2.small) and (t2.medium, t2.large), the result is similar, and though we double Memory in both groups, the increasement is less for t2.medium and t2.larger, which has larger vCPU

and bigger variance in Memory. As a result, we come to the result that vCPU is more vital here.

4.Discussions and recommendations

- **Discussion and potential bottlenecks**

As a result, we come into the conclusion that -c (number of concurrent request) is in positive ratio to connection time, but has little effect on request per second or transfer rate. And we find that in each group, the double of memory has only very small affection on request per second and little effect connection time. However, we do notice that the second group which has better vCPU always has better increasement in ratio when the memory doubles, which means vCPU effects. We think the small value of vCPU maybe the bottleneck and limitation of the result.

- **Recommendations**

If the customer does not take price into consideration, we recommend they choose the type with better CPU and Memory. But if we take price into consideration, though micro, small, medium to large, the price doubles each much, but the performance does not double. So we think the value for money of t2.medium is better, which means it worth the money.

Experient 2: Node.js and Apache + PHP comparison on the same AWS instance type

1.Experiment setup

In this experiment, we use Apache Benchmark tool to compare the performance of Node.js and Apache/PHP for loading the HelloWorld pages written in their own languages respectively.

Software versions

NAME	VERSION
APACHE BENCHMARK	v2.3
NODE.JS	v8.12.0

PHP	PHP/5.6.37
APACHE	2.4.34 (Amazon)
AMAZON EC2 INSTANCE TYPE	t2.micro

Experiment Process

To evaluate the ability to handle http requests, we test the HelloWorld files given in Node.js and Php official website respectively. We first deploy the HelloWorld.js file (running on port **3456**) and the HelloWorld.php file (running on port **80**) (See Figures above) on Node.js and Apache respectively **on the same Amazon instance**. Then we use Apache Benchmark tool to send requests ranging from 1000, 10000, 100000 **with a concurrency level of 10** respectively.

	Sample Testing Commands	**Concurrency Level**	**Port**
Node.js	ab -n 100000 -c 10 http://ec2-18-218-83-108.us-east-2.compute.amazonaws.com:3456/helloword.js	10	3456
Apache/PHP	$ ab -n 10000 -c 10 http://ec2-18-218-83-108.us-east-2.compute.amazonaws.com/~jinhan/helloword.php	10	80

2.Experimental results and discussion

#1 Requests Per Second

We select Requests Per Second (RPS) as the metric to compare the ability to process HTTP requests between Apache/PHP and Node.js.

Requests sent\Platform	**Apache/2.4.34 (Amazon) PHP/5.6.37**	**Node.js (v8.12.0)**
100	3277.83	5806.19
1000	3371.77	5731.55
10000	3429.72	5650.21
100000	3417.43	5942.06

Table 1. RPS Comparison

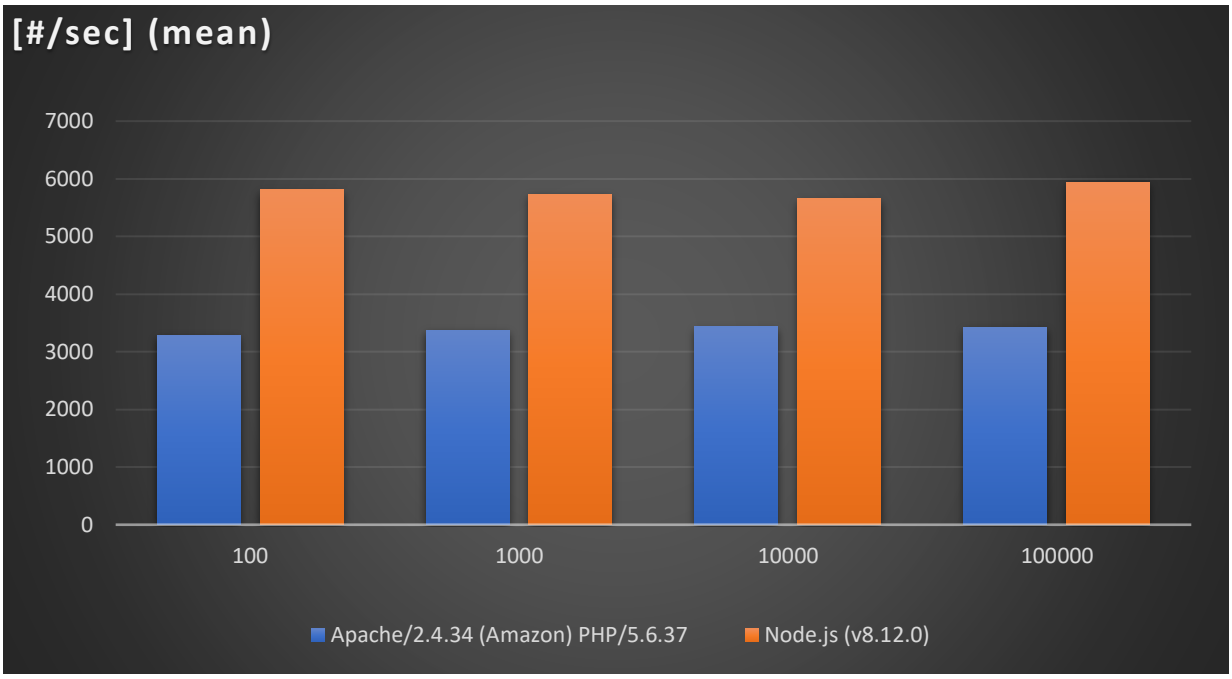


Figure 1. RPS Comparison

According to table 1 and figure 1, the average RPS for Node.js is about 70% greater than Apache/PHP. This means that Node.js has greater ability for handling simple http requests.

#2 Time taken for tests

We compare the time taken for tests to compare the overall performance in terms of loading a simple page between Node.js and Php+Apache.

Requests\Platform	Apache/2.4.34 (Amazon) PHP/5.6.37	Node.js (v8.12.0)
100	0.031s	0.017s
1000	0.297s	0.174s
10000	2.916s	1.77s
100000	29.262s	16.829s

Table 2. Time taken for tests

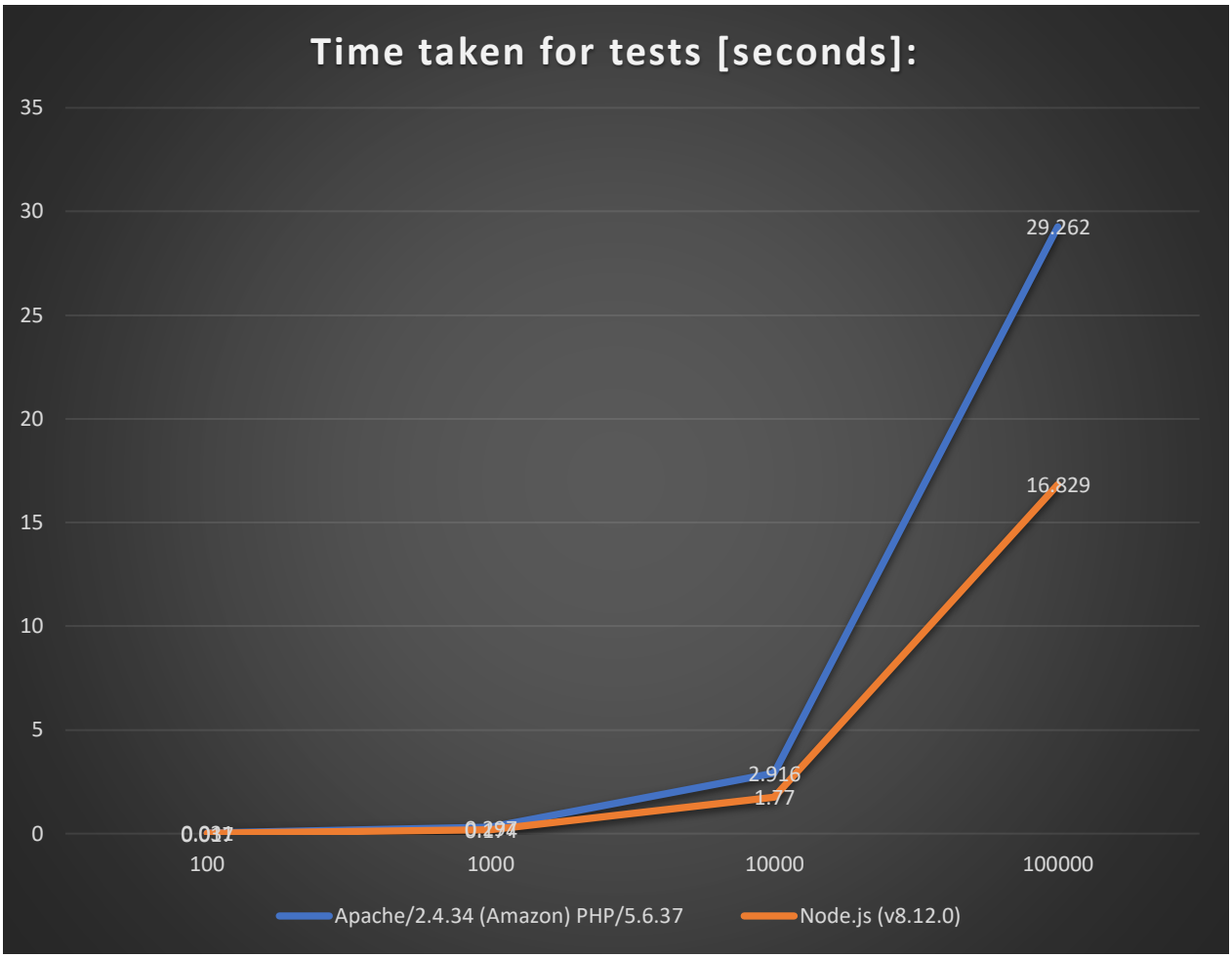


Figure 2. Time taken for tests

From table 1 and table 2 we find that the tests running for Node.js takes nearly half time as the one running on Apache/Php. As the number of requests grow, the difference between the two framework goes higher. This concludes that for simple HTTP requests, Node.js is about 70% faster than APACHE/PHP.

Further discussions and recommendations

Further discussions

We find that for completing simple http requests, Node.js is nearly 50% better than PHP. This is because everything in Node.js is running in a single process with non-blocking I/O while PHP+APACHE adopt standard blocking I/O. This means that tasks can be completed concurrently while in PHP, tasks can be executed only after previous commands finish. Hence

Potential bottlenecks

Node.js

Since node.js is running in a single thread of process, it would not be suitable for large scale application which focuses on CPU-intensive tasks. Also, as Node.js just came out few years ago, it is not mature as Apache/PHP and it is cumbersome to debug.

Apache/PHP

When the number of requests grow very large, Apache/PHP will be very slow. In this experiment, when we sent 100,000 requests with a concurrency level of 10 and the time taken for completing the test (i.e. 29.262s) is 74% more than the one running on Node.js (i.e. 16.829s). The security of Apache/PHP site also needs to be concerned.

Recommendations

We recommend using node.js for single page application and real time application because it utilizes non-blocking I/O. It handles concurrent requests in a very efficient way. However, when it comes to CPU-intensive Applications like online game, Node may not be suitable as it uses a single thread. We recommend using PHP/APACHE for large scale application that require ease of maintenance and easy integration with relational database. It is considered more mature than Node.js and has a larger supporting community.