# **CSE 503S Performance Evaluation Study**

Haiyu Wan: 458812

### Apache and Nginx comparison on the same AWS instance type:

1. Experiment Method:

Apache and Nginx are the two most common open source web servers in the world.  To compare the performances of Apache and Nginx, I use a simple html file:

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <meta name="viewport" content="width=device-width, initial-scale=1.0">
6. <meta http-equiv="X-UA-Compatible" content="ie=edge">
7. <title>Document</title>
8. </head>
9. <body>
10. Hello World!
11. </body>
12. </html>

Using the commond line:

1. ab -kc 100 -n 1000 http://ec2-18-188-41-39.us-east-2.compute.amazonaws.com/~h.wan/public\_html/module8/the.html

It will connect to the 100 concurrent connections 10.000 times. The result shows below:

1. This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
2. Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
3. Licensed to The Apache Software Foundation, http://www.apache.org/
4. Benchmarking ec2-18-188-41-39.us-east-2.compute.amazonaws.com (be patient)
5. Completed 100 requests
6. Completed 200 requests
7. Completed 300 requests
8. Completed 400 requests
9. Completed 500 requests
10. Completed 600 requests
11. Completed 700 requests
12. Completed 800 requests
13. Completed 900 requests
14. Completed 1000 requests
15. Finished 1000 requests
16. Server Software: Apache/2.4.39
17. Server Hostname: ec2-18-188-41-39.us-east-2.compute.amazonaws.com
18. Server Port: 80
19. Document Path: /~h.wan/public\_html/module8/testhtml.html
20. Document Length: 238 bytes
21. Concurrency Level: 100
22. Time taken for tests: 6.817 seconds
23. Complete requests: 1000
24. Failed requests: 13
25. (Connect: 0, Receive: 0, Length: 13, Exceptions: 0)
26. Non-2xx responses: 987
27. Keep-Alive requests: 987
28. Total transferred: 459040 bytes
29. HTML transferred: 234906 bytes
30. Requests per second: 146.68 [#/sec] (mean)
31. Time per request: 681.749 [ms] (mean)
32. Time per request: 6.817 [ms] (mean, across all concurrent requests)
33. Transfer rate: 65.75 [Kbytes/sec] received
34. Connection Times (ms)
35. min mean[+/-sd] median max
36. Connect: 0 1 2.1 0 8
37. Processing: 0 447 1454.4 1 6807
38. Waiting: 0 386 1359.0 1 6807
39. Total: 0 448 1456.2 1 6815
40. Percentage of the requests served within a certain time (ms)
41. 50% 1
42. 66% 1
43. 75% 1
44. 80% 1
45. 90% 7
46. 95% 5001
47. 98% 5769
48. 99% 5791
49. 100% 6815 (longest request)

The result show the data of Apach running on the instance of AWS.

Then we can use this line to install the Nginx and change the server:

1. sudo yum install epel-release
2. sudo yum install nginx
3. sudo apachectl stop
4. sudo nginx

Run the same ab commend to get the result.

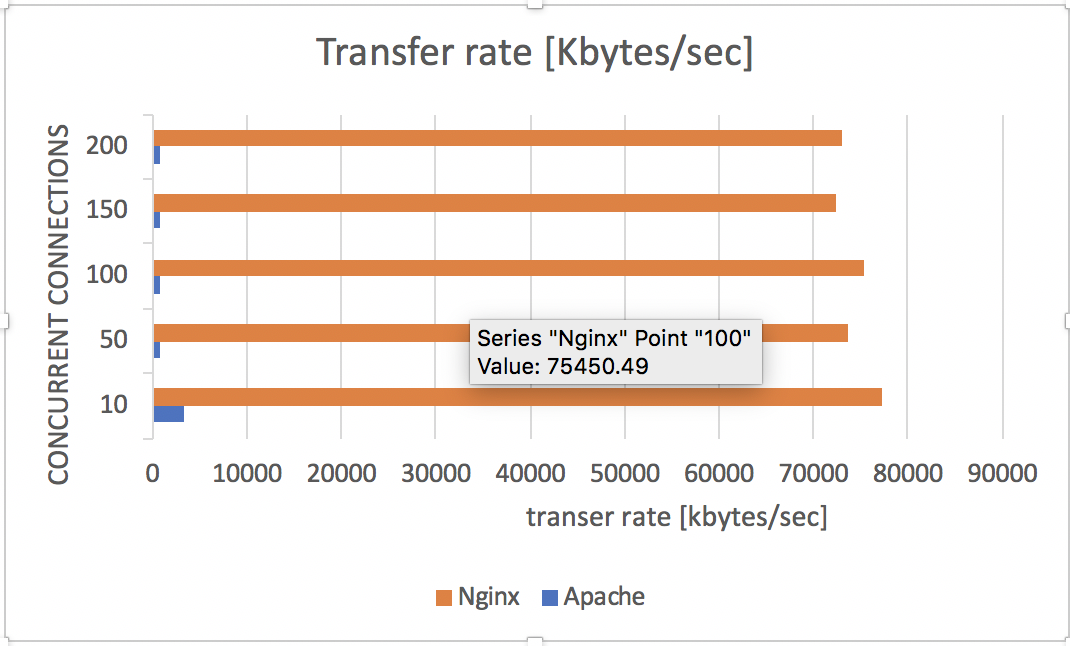
1. Set up:

In this experiment, I send the different concurrent connections (10, 50, 100, 150, 200) and 10000 requests in total. And the result is shown below.

1. Result:

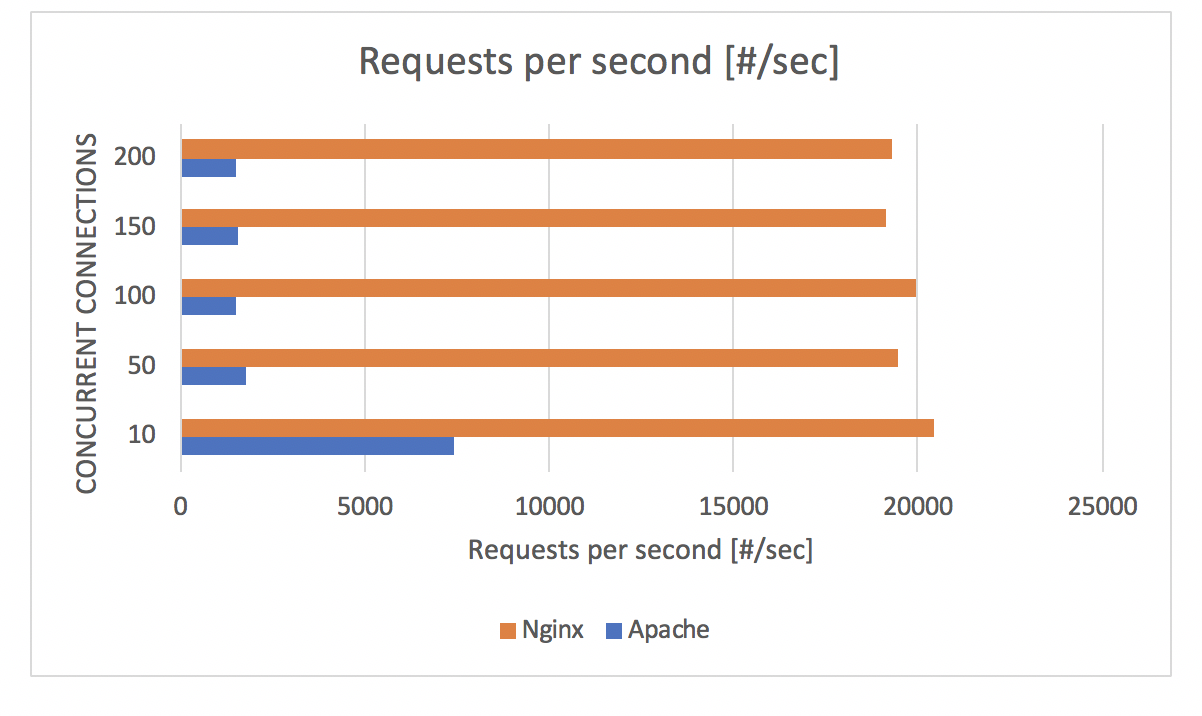
Transfer rate [Kbytes/sec]:

|  |  |  |  |
| --- | --- | --- | --- |
| kc | n | Apache | Nginx |
| 10 | 10000 | 3366.3 | 77372.44 |
| 50 | 10000 | 795.18 | 73574.11 |
| 100 | 10000 | 684.15 | 75450.49 |
| 150 | 10000 | 693.69 | 72477.24 |
| 200 | 10000 | 691.66 | 73081.51 |



Requests per second [#/sec]:

|  |  |  |  |
| --- | --- | --- | --- |
| kc | n | Apache | Nginx |
| 10 | 10000 | 7419.4 | 20451.82 |
| 50 | 10000 | 1752.68 | 19447.76 |
| 100 | 10000 | 1509.9 | 19943.76 |
| 150 | 10000 | 1530.84 | 19157.64 |
| 200 | 10000 | 1530.06 | 19317.36 |



1. Conclusion:

Transfer rate analysis:

The data transfer rate (DTR) is the amount of digital data that is moved from one place to another in a given time. The data transfer rate can be viewed as the speed of travel of a given amount of data from one place to another. In general, the greater the bandwidth of a given path, the higher the data transfer rate. According to the result, we can find that the transfer rate is much better when we used the Nginx than Apache. With the concurrent connects getting large, the performance of Apache is getting slower.

Requests per second analysis:

requests per second is a measurement of how fast the server can process requests, the more requests per second the server can handle, the more capable the server is to handle a lot of requests. According to the result, we can find that the requests per second of Nginx is better than Apache. With the concurrent connects getting large, the performance of Apache is getting slower.

Bottlenecks:

The reason why Nginx is better is because Nginx came onto the scene after Apache, with more awareness of the concurrency problems that would face sites at scale. Leveraging this knowledge, Nginx was designed from the ground up to use an asynchronous, non-blocking, event-driven connection handling algorithm.

Recommendation:

According to the result and analysis above, when facing the high-traffic situation.

I strongly suggest users to choose Nginx, which is much better than the Apache for this situation.

### Node.js and Apache+PHP comparison on the same AWS instance type

1. Experiment Method:

In this experiment, I decide to compare the Apache+php and Node.js in the same instance. I create simple php file and the node.js file to test the performance:

1. <?php
2. $loop=10000000;
3. while($loop>0){
4. $loop--;
5. }
6. echo "loop=10000000"
7. ?>
8. let http = require('http');
9. let app = http.createServer(function(req, res){
10. let loop=10000000;
11. while(loop>0){
12. loop--;
13. }
14. res.writeHead(200);
15. res.end('10000000');
16. });
17. app.listen(3456);

Then I can use the commend line to test the performances of them:

1. ab -n 25000 -c 100 <http://ec2-18-188-41-39.us-east-2.compute.amazonaws.com/~h.wan/public_html/module8/testphp.php>

The result is shown below:

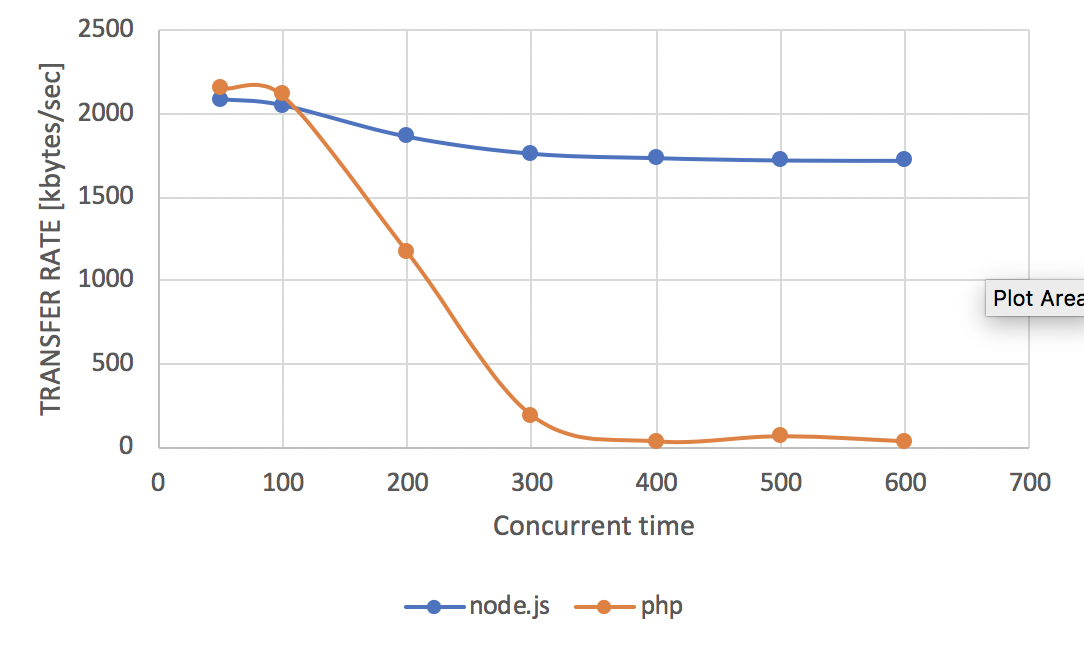
1. This is ApacheBench, Version 2.3 <$Revision: 1843412 $>
2. Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
3. Licensed to The Apache Software Foundation, http://www.apache.org/
4. Benchmarking ec2-18-188-41-39.us-east-2.compute.amazonaws.com (be patient)
5. Completed 2500 requests
6. Completed 5000 requests
7. Completed 7500 requests
8. Completed 10000 requests
9. Completed 12500 requests
10. Completed 15000 requests
11. Completed 17500 requests
12. Completed 20000 requests
13. Completed 22500 requests
14. Completed 25000 requests
15. Finished 25000 requests
16. Server Software: Apache/2.4.39
17. Server Hostname: ec2-18-188-41-39.us-east-2.compute.amazonaws.com
18. Server Port: 80
19. Document Path: /~h.wan/public\_html/module8/testphp.php
20. Document Length: 236 bytes
21. Concurrency Level: 100
22. Time taken for tests: 4.808 seconds
23. Complete requests: 25000
24. Failed requests: 0
25. Non-2xx responses: 25000
26. Total transferred: 10675000 bytes
27. HTML transferred: 5900000 bytes
28. Requests per second: 5199.38 [#/sec] (mean)
29. Time per request: 19.233 [ms] (mean)
30. Time per request: 0.192 [ms] (mean, across all concurrent requests)
31. Transfer rate: 2168.10 [Kbytes/sec] received
32. Connection Times (ms)
33. min mean[+/-sd] median max
34. Connect: 0 0 0.3 0 3
35. Processing: 2 19 0.7 19 20
36. Waiting: 0 18 0.7 18 20
37. Total: 2 19 0.6 19 22
38. Percentage of the requests served within a certain time (ms)
39. 50% 19
40. 66% 19
41. 75% 19
42. 80% 19
43. 90% 19
44. 95% 20
45. 98% 20
46. 99% 20
47. 100% 22 (longest request)
48. set up:

In this experiment, we fixed the total request is 10000 and use different concurrency levels: 50, 100, 200, 300, 400, 500, 600. The file is shown above (testphp.php and test.js)

1. Result:

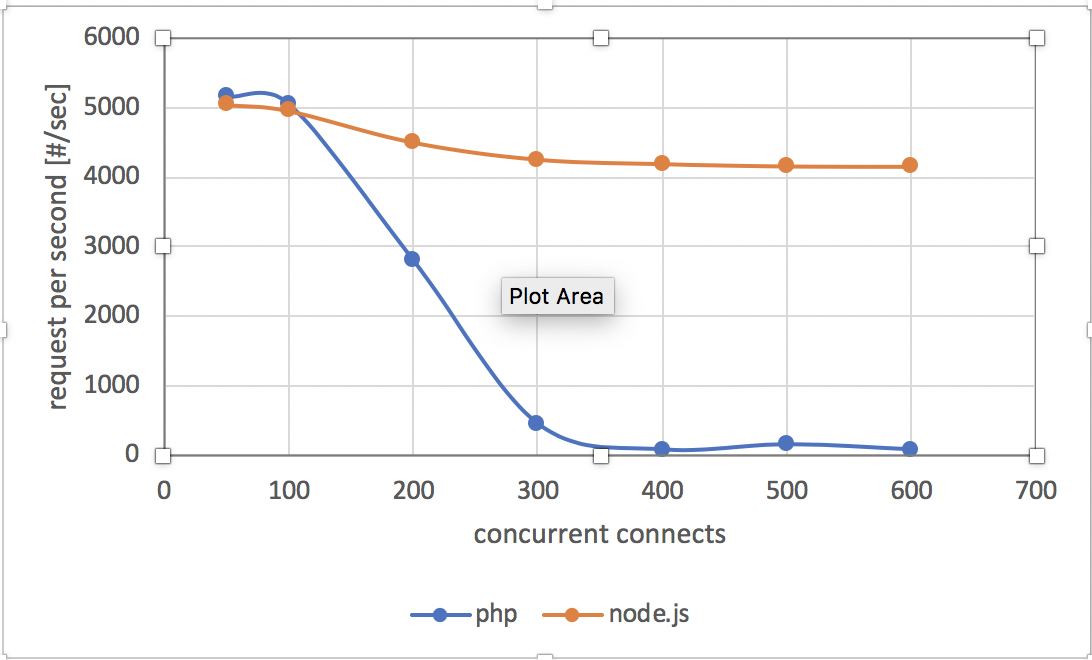
Transfer rate [Kbytes/sec]:

|  |  |  |  |
| --- | --- | --- | --- |
| kc | n | php | node.js |
| 50 | 10000 | 2152.26 | 5040.83 |
| 100 | 10000 | 2113 | 4955.2 |
| 200 | 10000 | *1173.57* | 4506.43 |
| 300 | 10000 | 189.99 | 4262.83 |
| 400 | 10000 | 34.21 | 4199.99 |
| 500 | 10000 | 65.29 | 4166.3 |
| 600 | 10000 | 34.44 | 4161.97 |



Requests per second [#/sec]:

|  |  |  |  |
| --- | --- | --- | --- |
| kc | n | php | node.js |
| 50 | 10000 | *5161.4* | 5040.83 |
| 100 | 10000 | 5067.25 | 4955.2 |
| 200 | 10000 | 2814.37 | 4506.43 |
| 300 | 10000 | 455.65 | 4262.83 |
| 400 | 10000 | 82.23 | 4199.99 |
| 500 | 10000 | 156.73 | 4166.3 |
| 600 | 10000 | 82.81 | 4161.97 |



1. Conclusion:

Transfer rate analysis:

According to the result of transfer, the performance of Node.js and Apache+PHP is basically same at lower concurrency. After 100, the PHP’s performance is getting worse and worse while the Node.js keep the same performance as the lower concurrency.

Request per second analysis:

According to the result of transfer, the performance of Node.js and Apache+PHP is basically same at lower concurrency. After 100, the PHP’s performance is getting worse and worse while the Node.js keep the same performance as the lower concurrency.

Bottlenecks:

Node.js and PHP differ in their approach to concurrency, with Node.js using a non-blocking event loop (running in a single process) and the standard PHP Zend runtime using a blocking process. For this reason, multiple PHP processess are often launched from the web server to be able to keep serving web requests. In Apache, you can set the maximum number of PHP processess allowed to run, and this works fine for many types of web applications.

Recommendation:

According the analysis, I found that the Apache+PHP is not suitable for the high-traffic situations. With the concurrent connects keeping lower, users can use both of them, while

in other situation, Node.JS is a better choices.