Performance Evaluations Report

Ren Wan 441422

Yiheng Ding 441394

Experimental 1 - MongoDB and MySQL comparison on the same AWS instance type

Setup:

MySQL: Sysbench
MongoDB: Mongoperf

We set up the performance test tool on t2.micro AWS instance with following environment:

Pymongo == 2.7.2
Bottle == 0.11.6
Argparse == 1.2.1
Requests >= 2.4.3
Psutil >= 2.1.3
Ordereddict >= 1.1
Python >= 2.7.X < 3.0
mongo shell >= 2.7.7-pre- (at revision 881b3a97fb5080b4e5d5ce11ad016da73ea23931 or newer)

Experimental result:

Result about MySQL:

Result of IO performance test: (data of events execution speed:ms)

	512MB	1024MB	1536MB	2048MB
sequence w	6.1961	14.6446	23.0989	31.5416
sequence r	0.1007	16.208	24.9111	33.8265
sequence r/w	6.1192	14.6197	23.0702	31.516
random w	0.0814	0.091	0.1016	0.1081
random r	0.0385	1.2301	2.1117	2.5407
random r/w	0.0593	0.8356	1.3376	1.6484

Result of CPU performance test: (data of events execution speed:ms)

	100	2000	5000	10000
primes generator	0.0198	1.1691	4.2663	10.9232

Result of Memory Access performance test: (data of events execution speed:ms)

	10G	20G	30G	40G	50G	60G	70G	80G	90G	100G
speed	6.2933	12.3742	18.565	24.7281	30.9815	37.329	43.3141	48.888	56.1996	61.1191

Result of MongoDB:

Result of insert operation (data of insert operation per second)

inserts per second	1 thread	2 threads	4 threads
file 1	12236.08186	11775.80885	11421.26168
file 2	11595.75097	11368.60092	11354.16572
file 3	11331.83058	11361.49782	11167.45808
file 4	9610.999986	9422.083544	9770.388423
file 5	9779.189022	9378.831996	9329.436269
file 6	8825.697548	8860.500143	8361.623205
file 7	8847.794751	8707.621952	8295.248907
file 8	12871.06625	12659.94677	12696.93569
file 9	12861.13732	12575.23663	12586.50007
file 10	11715.71104	11610.15897	11334.55016
file 11	11895.4241	11885.58417	11788.19487
file 12	10708.02843	10406.40996	10152.13605
file 13	10651.34638	10759.36843	10516.63887
file 14	11512.6857	11292.43136	11252.13907
file 15	11267.28452	11373.95273	11421.01792
file 16	11127.29872	10993.22282	11084.19529
file 17	13456.45594	13235.31867	13469.37698
file 18	11608.03435	11610.10703	12019.01252
file 19	11748.24873	11325.64944	11260.45841
file 20	13135.78885	13258.02706	12961.35492
file 21	12003.74469	11678.75227	11568.05952
file 22	10022.82638	9825.349999	9850.247481
file 23	12992.92061	12759.63059	12841.55325
file 24	12155.44506	11404.4868	11685.2273

Result of multi update operation (data in operations per second)

multiupdate	1 thread	2 threads	4 threads
file 1	2.076901737	2.029447868	1.847119382

file 2	0.495485434	0.496007438	0.365235404
file 3	1.829170846	1.685415352	1.597490024
file 4	0.367629792	0.389848197	0.302558019
file 5	67.98144107	65.89658848	66.17399149
file 6	19.27180342	18.91483407	18.5311904
file 7	10648.18801	10490.97159	10499.64613
file 8	8935.049819	9008.446881	9041.633991
file 9	7366.207005	7340.883202	7317.954776
file 10	6062.094495	6006.999934	5989.562659
file 11	10554.09393	10593.91798	10567.58802
file 12	8836.216357	8966.068732	8890.770914
file 13	57.58744018	55.38812933	53.8750476
file 14	21.1964284	22.0192159	22.44057382
file 15	3669.807335	3623.310497	3615.253735
file 16	2038.088313	2069.404282	2083.454349
file 17	11048.94312	11040.86562	10998.34746
file 18	9447.722237	9503.238575	9531.27245
file 19	12824.07296	12690.40076	12690.46014
file 20	12824.19476	12695.00558	12738.59939

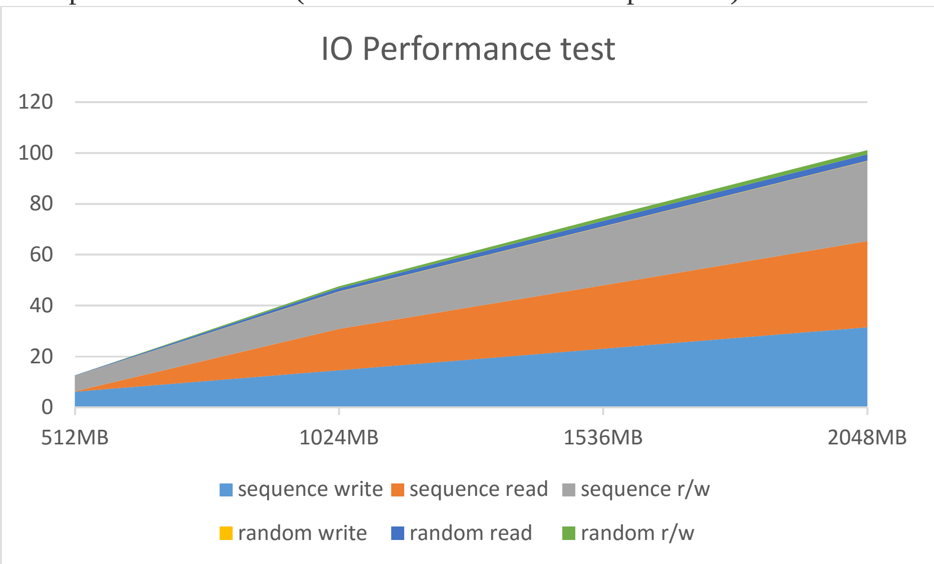
Result of update operation (data of operations per second)

update	1 thread	2 threads	4 threads
file 1	9360.48268	9145.706444	9183.558685
file 2	8273.223674	8126.110808	8102.598315
file 3	8048.60864	8052.283043	8052.910815
file 4	10634.52779	10633.63859	10654.57396
file 5	10805.11799	10771.87326	10726.55631
file 6	9272.550425	9318.812626	9268.504563
file 7	9395.182443	9232.277744	9215.240247
file 8	8012.988295	7982.933797	7948.463972
file 9	8027.189934	7939.471231	7890.880138
file 10	10762.61587	10800.29906	10789.99726
file 11	9180.664057	9082.323272	9088.373042
file 12	10779.67661	10735.46736	10683.90746
file 13	9452.034398	9461.248698	9181.946079
file 14	9847.47011	9764.631123	9782.685752
file 15	9382.242537	9400.803989	9539.653666
file 16	9791.4043	9722.135144	9778.705248
file 17	9470.689361	9717.226684	9750.566553
file 18	13514.35673	13235.56205	13090.39984
file 19	411.9356968	581.6262682	674.8991774
file 20	11314.73567	11309.89831	10967.24734
file 21	10922.0636	10894.71581	10965.46267

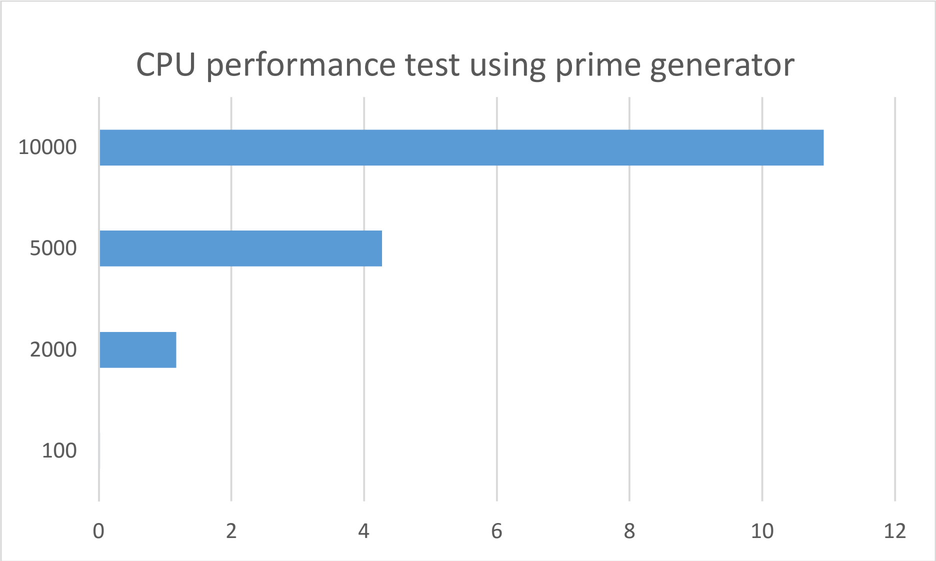
file 22	10449.73739	10539.58027	10402.73969
file 23	9880.535408	9885.039357	9856.99306
file 24	9390.057737	9554.686913	9395.66084
file 25	9878.838211	9830.139949	9737.62594
file 26	9113.192885	9121.890658	8955.607416
file 27	9224.786357	9138.463049	9052.825497

Visualization:

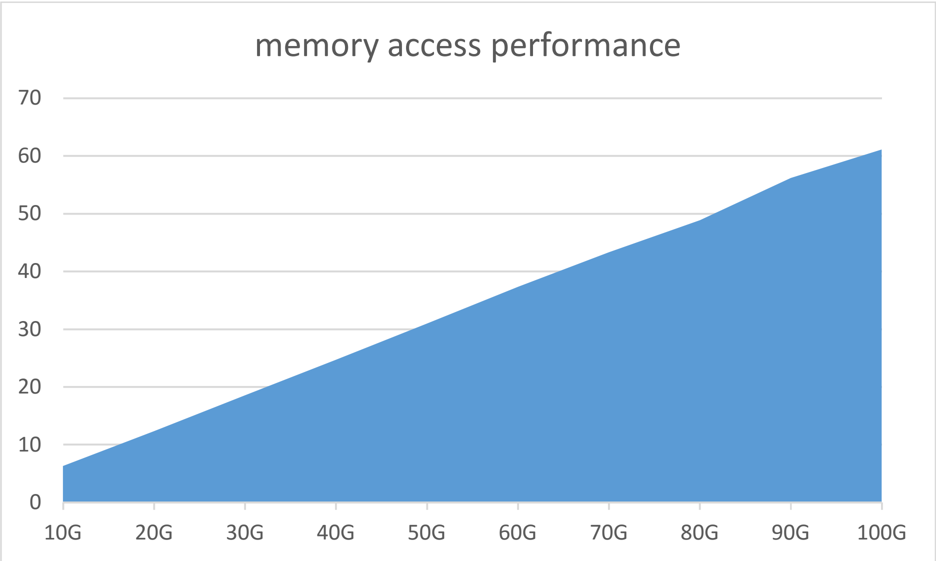
Result of IO performance test: (data of events execution speed:ms)



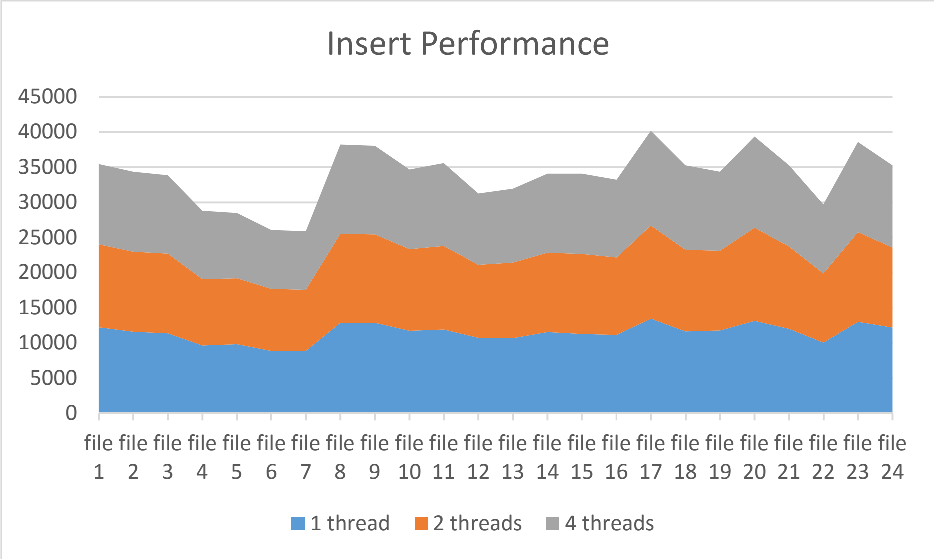
Result of CPU performance test: (data of events execution speed:ms)



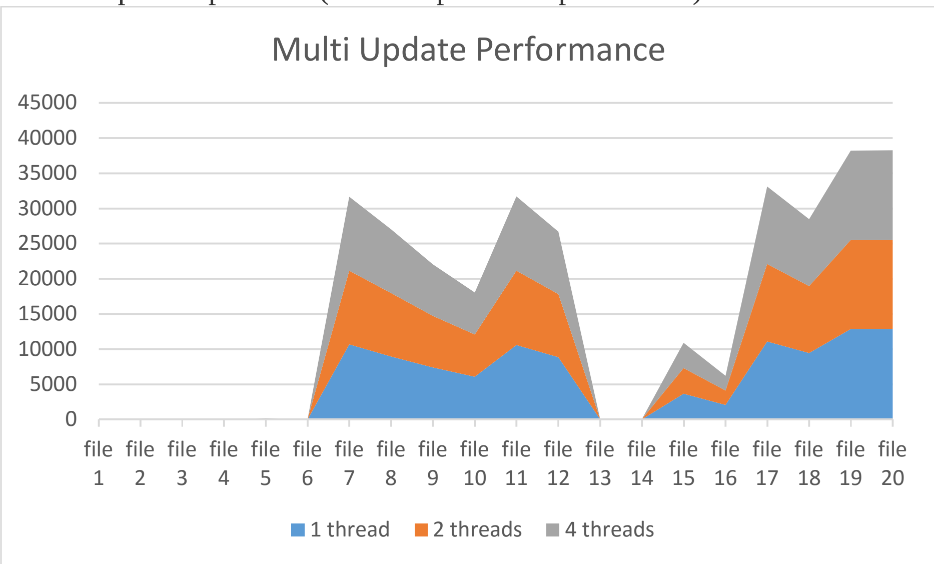
Result of Memory Access performance test: (data of events execution speed:ms)



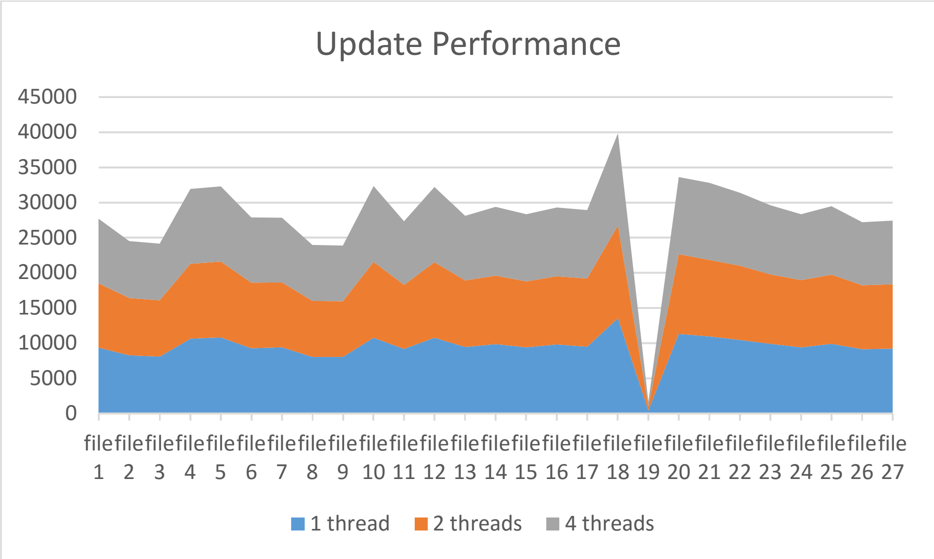
Result of insert operation (data of insert operation per second)



Result of multi update operation (data in operations per second)



Result of update operation (data of operations per second)



Discussion:

- **Result discussion:**
  The result data in the chart before indicating that MySQL is pretty stable about fileIO, CPU and memory access performance and slow. And conversely, the data of MongoDB is pretty, for example, with 4 threads environment, it can reach 30000 update operations per second. And for other two operations, it is still better than MySQL.

  Be more specific, talking about the IO performance test, with the increase of the size of the file, the average event execution time obviously raises. The size of file increase for 4 times, the exe times raises for 5 times, which means the bigger the file to input or output, the slower the MySQL will become.

  And checking the result of MongoDB, the file1~27 varies in sizes, in a random sequence. And as what we can see, the performance is pretty stable. The gap in the second and third graph is due to the breakdown of the test case. So please ignore them.

  For example, like the insert operation of MongoDB, the file 1 is simple insert with relatively simple data structure and smaller data size and the file 2 is complex insert and do the converse. But as what we can see, the OPS remains roughly same, which suggest it's stable.

- **Bottlenecks**
  Then why we use MySQL if it's so lame? No, it's not lame. MySQL maintains clear and beautiful structure between data. And the interaction with them is pretty simple. And what's more, MySQL supports complex transaction which MongoDB doesn't.

The bottlenecks of MySQL is the poor performance due to low ability dealing with large data, once the data becomes huge, its performance rapidly drops down and clearly affect the performance the server. And if the scheme for all the data is extreme difficult to define and maintain, using MySQL is not a good practice.

And about the MongoDB, on the contrary, if the data scheme is pretty important and the necessity to using complex transaction overcome the tradeoff of losing performance, then the MySQL is a better choice.

- **Suggestion:**
  They are both useful, so you need to make choice according to the real situation. Like, if the performance is the first consideration, the MongoDB is better choice, if the structure of row and table is more important, MySQL is better choice.

Experimental 2 – Node.js and Apache server comparison on the same AWS instance type

Software setup:

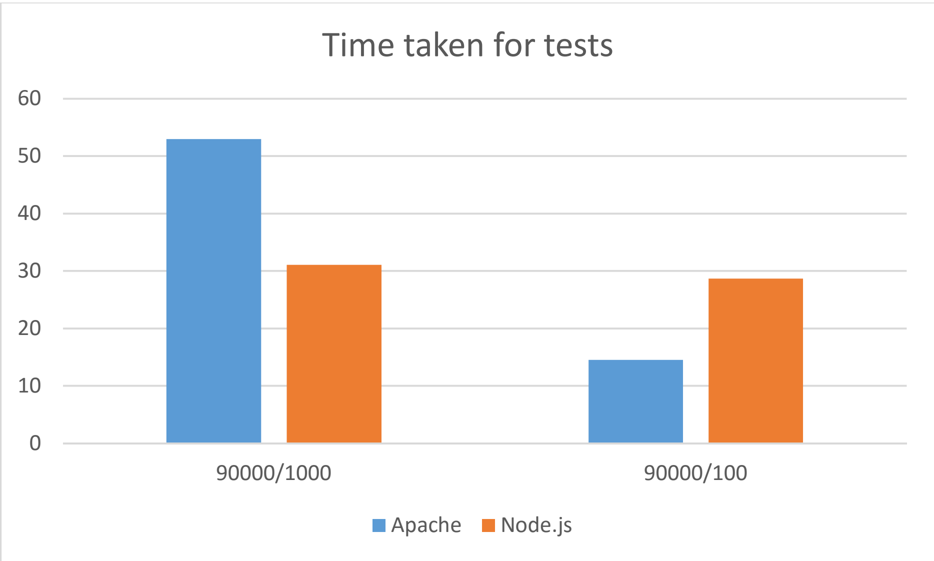
Apache: 2.2.31

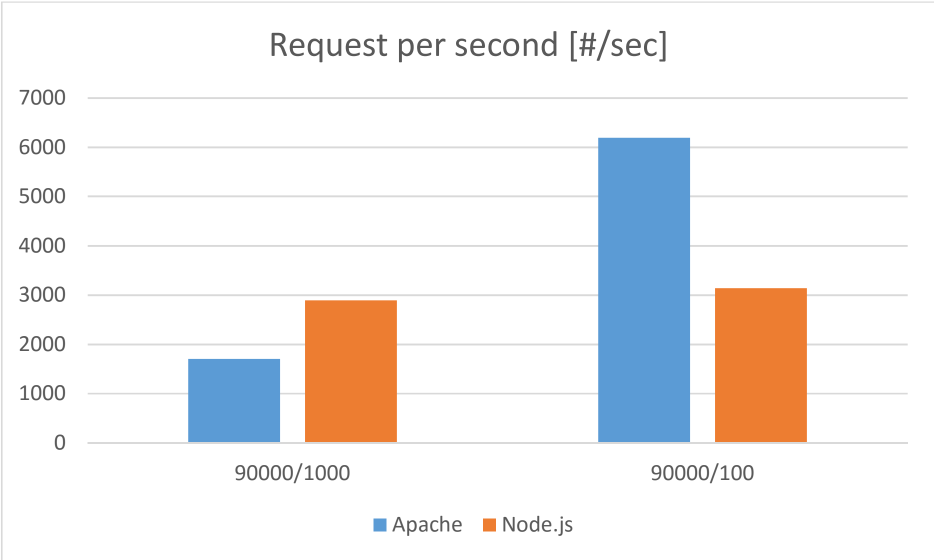
Node: 4.2.1

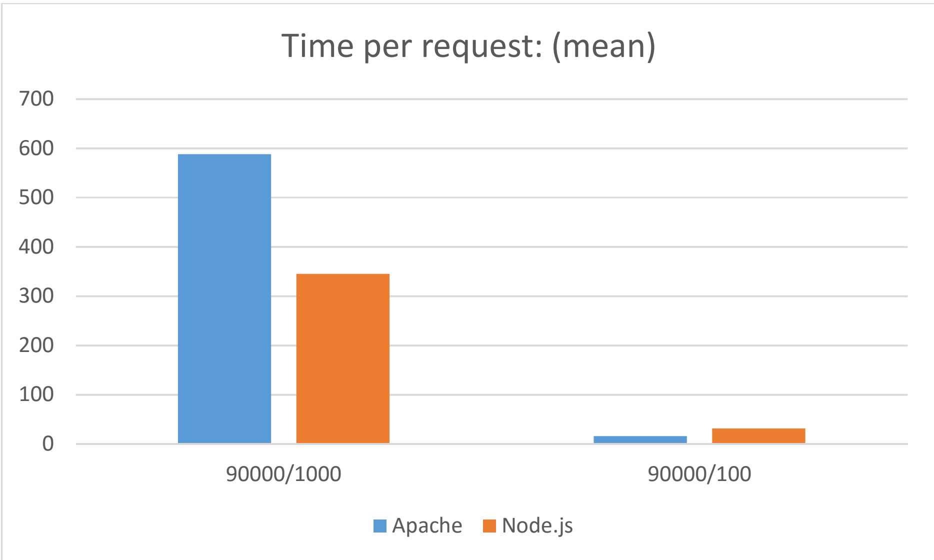
Experimental results (apache benchmark output):

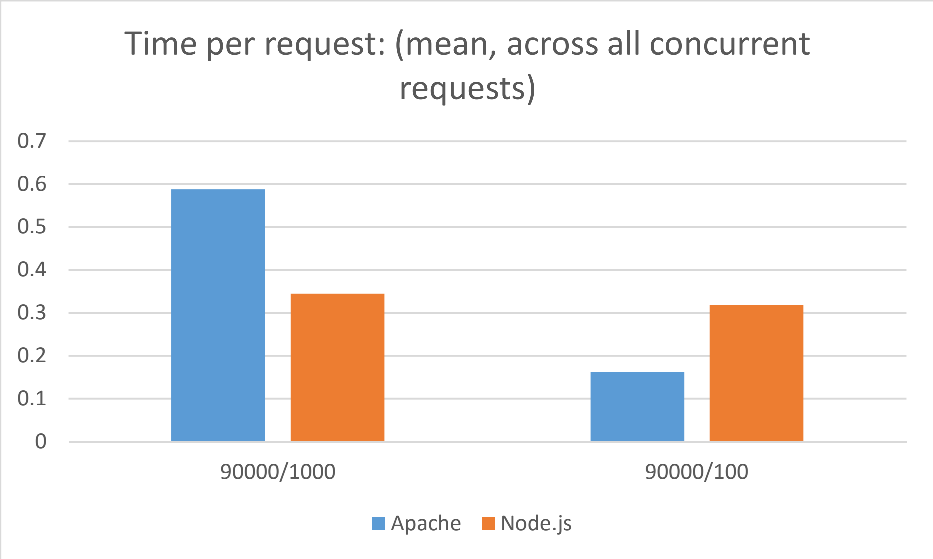
Requests/Concurrency	90000/1000		90000/100	
Server Type:	Apache	Node.js	Apache	Node.js
Document Length: bytes	503	503	503	503
Concurrency Level	1000	100	100	100
Time taken for tests (seconds)	52.945	31.074	14.541	28.652
Requests per second [#/sec]	1699.88	2896.33	6189.2	3141.18
Time per request: (mean)	588.276	345.264	16.157	31.835
Time per request: (mean, across all concurrent requests)	0.588	0.345	0.162	0.318
Transfer rate: [Kbytes/sec]	1284.09	1705.56	4669.29	1849.74

Visualization of experimental results:

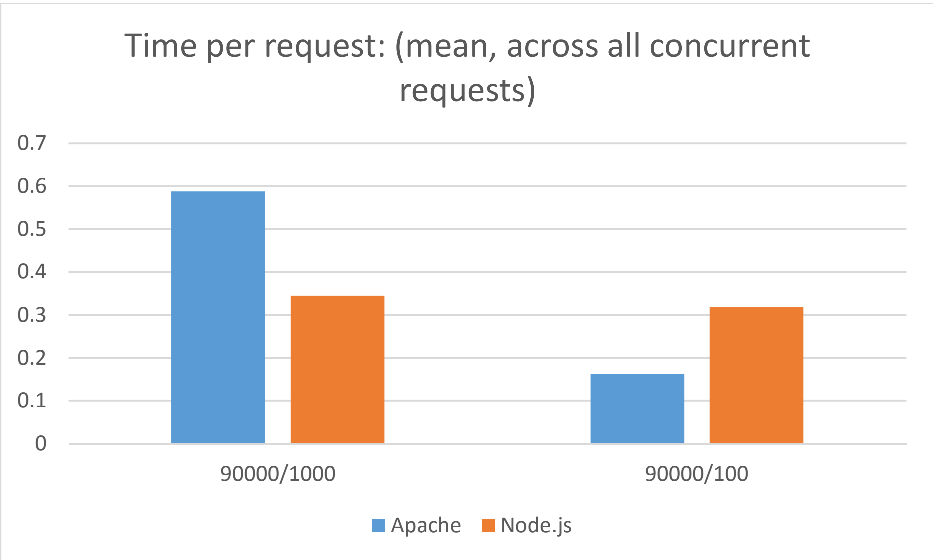



Request per second [#/sec]
7000
6000
5000
4000
3000
2000
1000
0
90000/1000
90000/100
Apache
Node.js

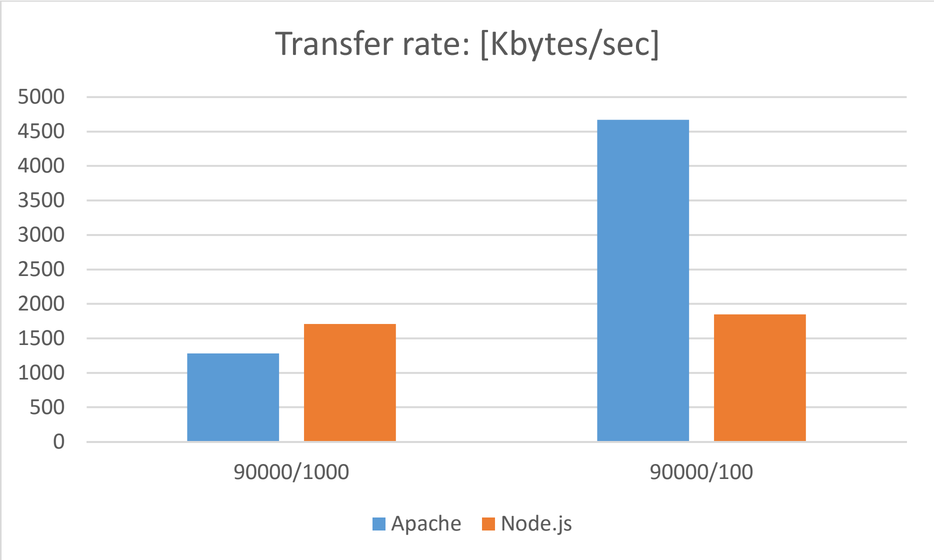

Time per request: (mean)
700
600
500
400
300
200
100
0
90000/1000
90000/100
Apache
Node.js



Time per request: (mean, across all concurrent requests)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
90000/1000
90000/100
Apache
Node.js



Time per request: (mean, across all concurrent requests)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
90000/1000
90000/100
Apache
Node.js



Detailed discussion of results:

In this experiment, we did the tests of serving static file for both Node.js and Apache server. The result shows that Apache server's performance is quite good when the concurrency is not high (100 requests). It performs better than Node.js in most cases. However, Node.js performed better than Apache server when concurrent connections increases. Because Node.js has better concurrency handling, so the Node.js' server takes less time on each requests.

The potential bottlenecks of the system:

Both systems are not able to handle too much concurrency connection on AWS micro instance. Apache has very high chance to get error message when the number of request is over 100000. (Error message shows apr_socket_recv: Connection reset by peer).

Recommendation:

An Apache server can served by using PHP. It may takes less time to develop, since it is very easy for the beginners. The Node.js is event driven, which means it handles high concurrency very well. It is written in JavaScript, it is not hard to use if the user knows the syntax of JavaScript. Also, npm also helps the user to deploy the application. The choice of the technology really does depend on the needs of the website. If the website has very high demand of concurrency, like Amazon, Node.js would be a better choice. If the server does not serve for many users at the same time, Apache is sufficient. Based our observation, we found that a micro AWS instance can only server 1000 requests at a time. We may want to consider to use a bigger instance if our website requested over 1000 at same time.