International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 9 Issue 7 ǁ 2021 ǁ PP. 29-37

The Better Comparison between PHP, Python-web & Node.js

Harkirat Brar [*1], TaranPreet Kaur[*2], Yash Rajoria[*3]
[*1]*Teacher, CSE Department, MIMIT, Malout, Punjab, India (Font Size -11)*
[*2]*Student, CSE Department, MIMIT, Malout, Punjab, India*
[*3]*Student, CSE Department, MIMIT, Malout, Punjab, India*

ABSTRACT
Large scale, high concurrency, and a vast quantity of data are important trends for the new age of websites. Node.js becomes popular and flourishing to build data-intensive web applications. To analyze and examine the performance of Node.js, Python-Web, and PHP, we used benchmark tests and scenario tests. The test results yield some valuable enforcement data, showing that PHP and Python-Web manage much fewer requests than that Node.js in a certain time. In conclusion, our results demonstrate that Node.js is quite lightweight and effective, which is an ideal fit for I/O intense websites among the three, while PHP is only fitting for small and middle scale applications, and Python-Web is developer-friendly and good for large web structures. To the best of our experience, to judge these Web programming technologies with both objective methodical tests (benchmark) and realistic user behavior tests (scenario), especially taking Node.js as the main topic to talk about.

Keywords: *Development; Performance Evaluation; Node.js; Benchmark Test; Scenario Test*

Date of Submission: 21-06-2021 Date of acceptance: 06-07-2021

I. INTRODUCTION

The Web today, many sites are faced with new obstacles, such as the problem of multiuser requests and high concurrency. The dynamic scripting language JavaScript has become enormously attractive for clients and is widely used in Web development. Node.js stands for one new technology in JavaScript. Node.js is a stage built on Chrome's JavaScript runtime for simply building fast, scalable network applications [1]. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and effective, ideal for data-intensive real-time applications that run across distributed devices [1]. Node.js popularity reviews conducted by the official website indicate that the average downloads are over 35,000 since version 0.10 was released in March 2013. Corporations are quickly realizing the importance of Node.js and five major PAAS providers have supported Node.js [2]. Nowadays, JavaScript has been the first popular language in GitHub. [3]. And expressing about the evaluation of Web technologies' performance, many researchers have done the related work. But our work differs from others in these two features. Firstly, we consider from both objective methodical tests (benchmark) and realistic user behavior tests (scenario) two sides and use the most innovative commercial testing tool LoadRunner. Secondly, we mainly concern about the performance of holding concurrent users to meet the demand for IO-intensive real-time websites.

This paper concentrates on the impact on Web performance from three separate Web technologies: Node.js, PHP, and Python-Web. The protection and scalability issues are beyond the range of the paper. We mainly use benchmark tests and scenario tests. In computing, one universal method of Web development technique's evaluation based on the completion comparison is proposed in the paper, which can be used to evaluate any new Web technology. The main supplements of this paper are listed as follows. (1)We consider new web technology Node.js in our experiment and examine the results of it. Then we examine it with PHP and Python-Web, making a judgment of which situation they ought to be used.

(2) Through benchmark tests and scenario tests, we can judge performance from both objective systematic experiments (benchmark) and realistic user behavior tests (scenario). There is often a dual impact on Web server performance, from the estimate, and the number of users. Our research has taken each of these effects into account.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the testbed and configurations in our experiment. Section 4 details our methodology and experiential design of tests. Section 5 presents and examines the results of all tests. Section 6 makes a result of the paper with a summary of our study and a future direction.

II. METHODOLOGY

The experiment assessed the results from two respects, one from the server to do benchmark tests, the other from the client to simulate the behavior of users to do scenario tests. In all the tests, we must follow a one-factor-at-a-time experimental study [16] to secure the exactness and effectiveness of the tests.

A. Benchmark Test 1) Benchmark Test Methodology

According to a one-factor-at-a-time experimental design, we make three fundamental tests – "THE TEST", "Use of Sorting", and "Select Operation of DB". "THE TEST" module is a basic module to build a good Web server, then output "hello world" and distinguish the differences of those three technologies. The "Use of Sorting" module is to calculate some values of Sorting and assess the completion under compute-intensive tests. The "Select Operation of DB" module is to compare various performances through querying some value of DB in the IO-intensive situation. Under all benchmark tests, we keep requests 10000, and then we turn users from 10 to 1000. TABLE ĉ summarizes the factors in our experiments

2)Benchmark Test Configuration

In the process of the test, we found results of the same module are similar. For example, we choose PHP to make three tests under requests 10000 and users 100. With the number of calls from 0 to10000, the results of the three tests are as shown in Fig.2. The acknowledgment time doesn't have much variation in the three tests with the increase of concurrency requests. The average time of the three tests is separate 0.311ms, 0.305ms, and 0.319ms. So, we use one test result to evaluate performance in our experiment. In addition, we reboot a server in every test to make sure fairer in the whole experiment.

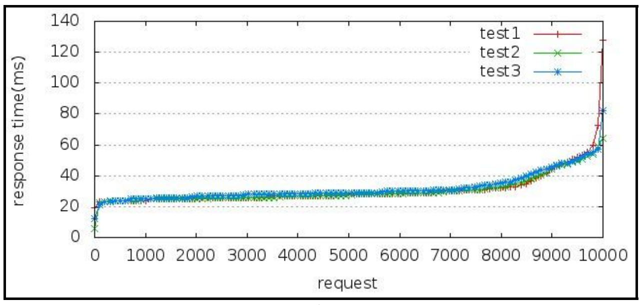

Fig. 2. Three tests of PHP under user 100

In starting tests, we found some interrupt when concurrent requests increased to 200 in Python-Web. Thus, we modified the code of Ab and replaced some codes of line 1449 with codes shown in TABLEĊ.

TABLE II. THE MODIFIED CODE OF AB

bad++;
close connection(c); return;

Meanwhile, we changed the Linux Kernel parameters in case that the system was regarded as an SYN flood attack under high concurrency. The parameters modified were shown in TABLE ċ.

TABLE III. THE MODIFIED CODE OF LINUX KERNEL PARAMETERS

net.ipv4.conf.default.rp_filter = 1 net.ipv4.conf.all.rp_filter = 1
net.ipv4.tcp_syncookies = 0
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv4.tcp_max_syn_backlog = 819200
net.ipv4.tcp_synack_retries = 1
net.ipv4.tcp_max_tw_buckets = 819200
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1

ensure Apache can deal with requests as large as possible. We modified “MaxClients” to 5000, “Server Limit” to 5000, and “MaxRequestsPerChild” to 0 in the PreFork mode.

B. Scenario Test 1) Scenario Test Methodology

The scenario test aims to simulate realistic user behavior. In our analysis, we divide scenario tests into two parts, one is the "Login" situation and the other is the "Encryption" situation. In terms of the "Login" situation, it mainly affects concurrent users to log in at the same time, and then analyzes the performance of three Web technologies in the real IO-intensive situation. In the test, we choose 500 users as rendezvous because it appears some errors when users rise to 500. That also means the pressure is beyond the highest range Web server can stand. We then use correct results when users are 500. In the test, we make statistics of throughput, the average transaction response time, and "hits per second" to compare the performance of those three technologies. Throughput displays the volume of data in bytes the Vusers received from the server at any given second. "Hits per second" displays the number of hits made on the Web server by Vusers during each second of the load test. The "Encryption" situation is to simulate a process to encrypt users' login passwords when user's login in. It's mainly to compare performance in the simple real compute-intensive situation. We choose the same rendezvous as the "Login" scenario.

2) Scenario Test Configuration

To make enough fair in our research, we make the same arrangement as benchmark tests. We reboot the server in every situation test.

.

Subheading

Subheading should be Font Size- 10pt, Font Type- Cambria, justified.

Subheading

Subheading should be 10pt Times new Roman,

III. MODELING AND ANALYSIS

A. Outcomes and Analyses of Benchmark Tests 1) Ā"THE TEST" module

With the increase of users, the execution of the three technologies shows an aim of increasing before contracting when keeping the requests at 10000. As FIG.3 and FIG.4 are shown, the "mean requests per second" is the highest which rises to 3703.5 times per second when the users of Node.js are 100. Meanwhile, the "meantime per request" is the lowest which is 0.27ms. Then the "mean requests per second" slows down and maintains a steady-state around 2700. As to Python Web, the "mean requests per second" keeps stable at around 500 and the highest is 559.42. At this time, the "meantime per request" is the shortest which is 1.788ms. To PHP, it is also at a peak when users are 100. The "mean requests per second" is 2977.54 at that time. The "meantime per request" is the shortest 0.336ms. With users rising, the" mean" requests per second" decreases to 200 and remains stable.

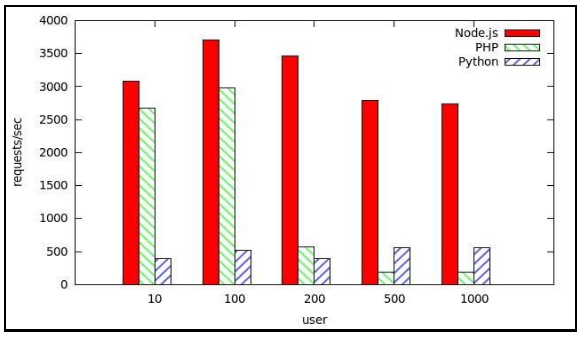

Fig. 3. Results for "THE TEST" mean requests per second

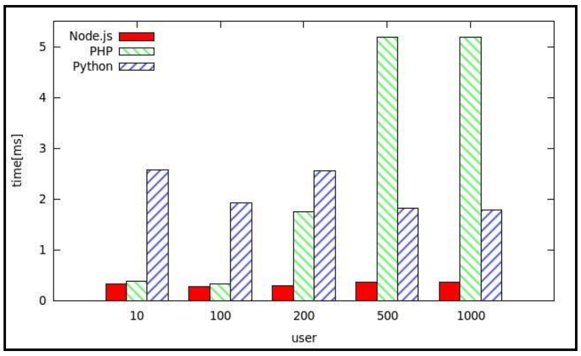

Fig. 4. Results for "THE TEST" mean time per request

In short, the execution of Node.js is better than two others at the same quantity of users. The performance of Node.js is two times larger than PHP on the basic performance measurement and six to seven times larger than Python-Web. In addition, the current users that Node.js can hold are far more than PHP, let alone Python-Web. So its performance is much better than PHP and Python-Web when there are lots of users. decrease 1.5 times compared with the same condition of "THE TEST" benchmark tests. But to Python-Web, this module has similar results as the last module, even better decisions, at the same condition. PHP is also developed to top value when users' number is up to 100, the "mean requests per second" at 3127.98. There is not much difference between it and the "THE TEST" module.

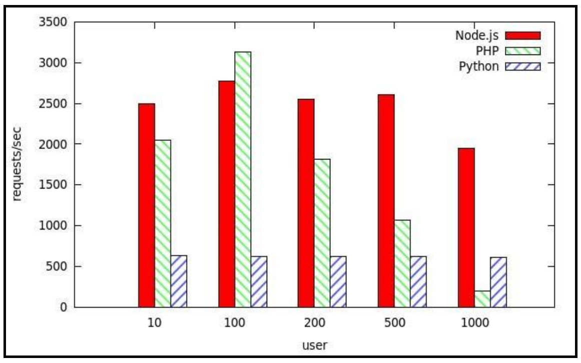

Fig. 5. Results for "Use of Sorting (10)" means requests per second

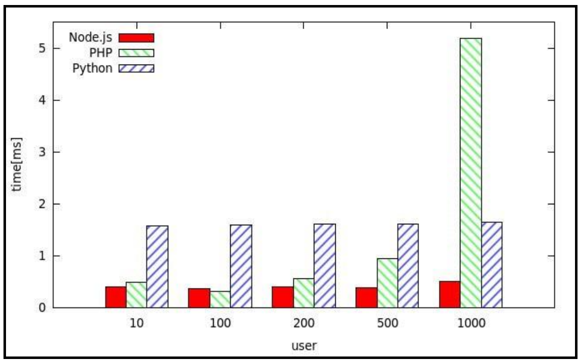

Fig. 6. Results for "Use of Sorting (10)" mean time per request

The results and discussion may be combined into a common section or obtainable separately. They may also be broken into subsets with short, revealing captions. An easy way to comply with the conference paper formatting requirements is to use this document as a template and simply type your text into it. This section should be typed in character size 10pt Times New Roman.

The reduce much in performance, especially PHP which "mean Web technologies all don't adapt to compute-intensive requests per second" drops from 2000 to 2. Besides PHP, application. However, Node.js performs better among the three Python-Web reduces from 600 to 3. Node.js also reduces in that test. much from 2500 to 60. This phenomenon means those three

TABLE IV. RESULTS FOR "Use of Sorting (10/20/30))"

Web development technology	Use of Sorting	Mean requests per second [#/sec]	Meantime per request [ms]
Node.js	Fib (10/ 20/ 30)	2491.77/ 1529.4/ 58.85	0.401/ 0.654/ 16.993
Python-Web	Fib (10/ 20/ 30)	633.68/ 209.89/ 2.9	1.578/ 4.764/ 345.307
PHP	Fib (10/ 20/ 30)	2051.22/ 168.8/ 1.78	0.488/ 5.942/ 560.553

2) "Use of Sorting" module

FIG.5 and FIG.6 show the Web performance to estimate the tenth value of sorting when requests are 10000 and users' number rises on and on. In terms of Node.js, the "mean requests per second" is the highest value reaching up to 2777.72 times per second, and the "meantime per request" is 0.36ms when users are 100. In addition, the "mean requests per second" of Node.js keeps from 2000 to 2800. The top requests per second "Use of Sorting (10)" tests. But we found they all are very alike to the "THE TEST" module in the same condition, so

we choose other values of Sorting to validate the outcomes. We calculate the twentieth and thirtieth values of Sorting when users are 10. TABLE Č shows the results of the tests above. The performance of the three Web technologies decreases to a different degree with the increase of Values of Sorting. Considering the "Use of Sorting (30)" test, all the tests According to the results above, we make several tests with Node.js to find performance differences in various concurrent users as the calculation rises.
FIG.7 and FIG.8 show the results of "Use of Sorting (10/20/30))" as the users grow from 10 to 1000.

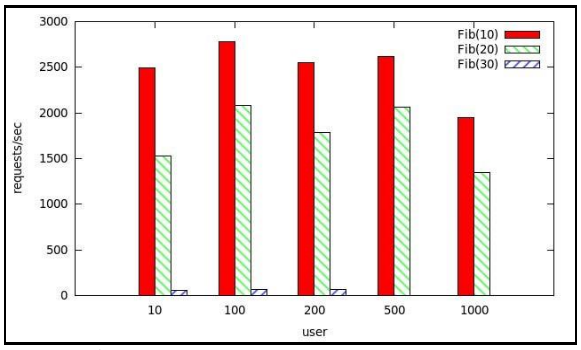

Fig. 7. Results for "Use of Sorting (10/20/30))" means requests per second with Node.js

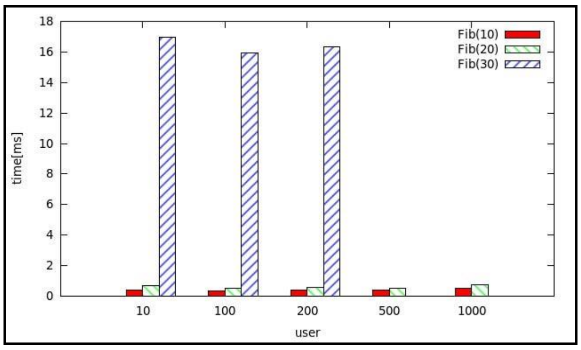

Fig. 8. Results for "Use of Sorting(10/20/30)" mean time per request with Node.js

Although there is a little variation between the results of various concurrent users at the same calculation, the whole trend keeps stable. The "mean requests per second" of Sorting (10) is between 2000 and 2800. Sorting (20) is between 1300 and 2000. Sorting (30) is around 60. Meanwhile, the test of Sorting (30) is interrupted when users are up to 500. From the above, the increment of calculation brings much more effective than the larger users. We make a simple conclusion that Node.js is more adapted to IO-intensive applications, not the compute-intensive application because compute-intensive applications don't exploit the good advantages of Node.js.

3) "Select Operation of DB" module

To validate the conclusion to prove Node.js is adapted to IO-intensive applications, we design a "Select Operation of DB" module. FIG.9 and FIG.10 show the results of this module.

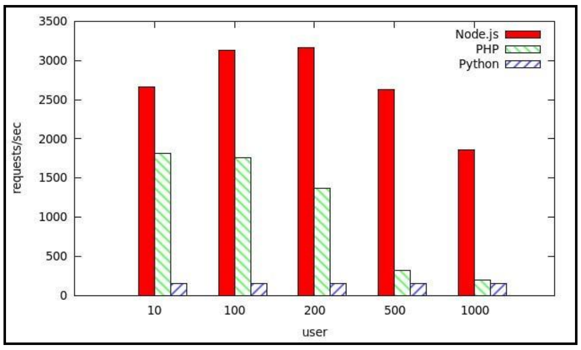

Fig. 9. Results for "Select Operation of DB" mean requests per second

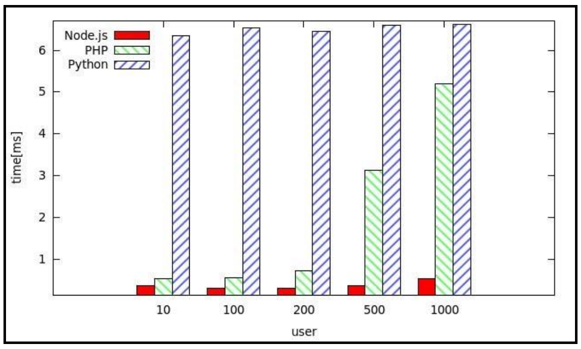

Fig. 10. Results for "Select Operation of DB" mean time per request

The max "means requests per second" of Node.js is 3164.46, 20 times larger than Python-Web and 2 times larger than PHP. In expanding, the peak value of the "Select Operation of DB" module including Node.js doesn't have much difference from the "THE TEST" module. The "mean requests per second" of Python Web keep around 150 and supports stability as users rise. But to PHP, the "mean requests per second" slows down and the "meantime per request" is longer. It explains that Node.js is more suitable for IO-intensive applications among the three, while PHP applies to small-scale websites.

B. Outcomes and Judgments of Situation Tests

To validate the results on benchmark tests, we choose two situations as follows.

1) A Login" situation

We choose peak users at 500 to do tests in the "Login" situation. We mainly observe "hits per second", throughput and common performance acknowledgment time as users go up.

From FIG.11 to FIG.14, the results of the "Login" situation are exhibited. The horizontal axis represents the number of parallel users, the vertical axis representing hits per second, throughput, throughput trend, common action response time.

FIG.11 shows "hits per second" for different concurrent users. "Hits per second" measures the number of HTTP requests sent to the Web server from virtual users per second in performance tests. "Hits per second" is longer and the importance to the Web server is larger. It can be seen from FIG.11 that "hits per second" reduces to a large degree when users are up to 150. It is to say the system can't hold so many users.

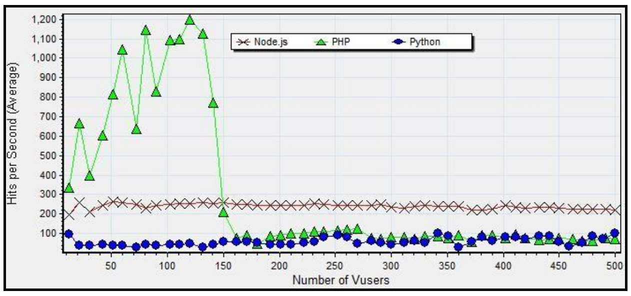

Fig. 11. Hits per second

In expanding, throughput in FIG.13 slows down simply when users are 150. Meanwhile, the average transaction acknowledgment time in FIG.14 extends much and it means the load of the system is at a peak. It leads to a higher response time with the development of users, and it appears some users' timeout. So "hits per second" in FIG.11 reductions.

The throughput of Node.js is about 5,000,000 byte/s in FIG.12. However, the throughput of PHP and Python-Web are both below 1,000,000 byte/s. We can see more detailed data from FIG.13 that the throughput of Node.js is between 4,000,000 and 5,000,000 byte/s. The throughput of PHP stays 500,000 byte/s when the users are less than 150 and it's similar to Python-Web after that, keeping to 70,000 byte/s. In general, the throughput of Node.js is far more than PHP and Python, more adapted to IO-intensive requests. On the other hand, Node.js is more suited to the concurrent situation, while PHP applies to middle and small-scale websites.

The common transaction reply time of PHP and Node.js are very close within the first 150 users, even the time of PHP is less than Node.js in FIG.14. At the same time, the common transaction response time of PHP and Node.js are both less than 4s. After that, the response time of PHP is much larger than Node.js with users rising. However, Node.js stays steady growth trend because its rate to deal with requests is higher than PHP with the concurrent users are up. Thus Node.js takes a better place when user requests increase.

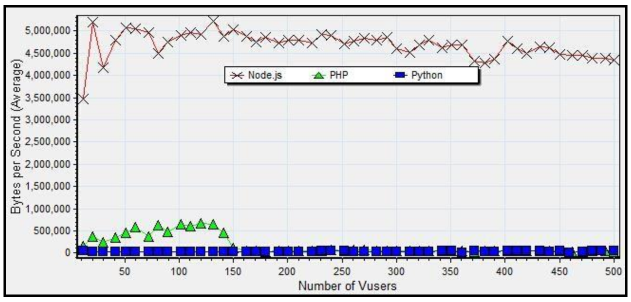

Fig. 12.Throughput

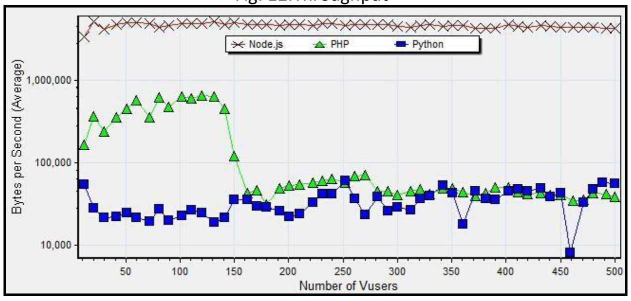

Fig. 13.Throughput trend

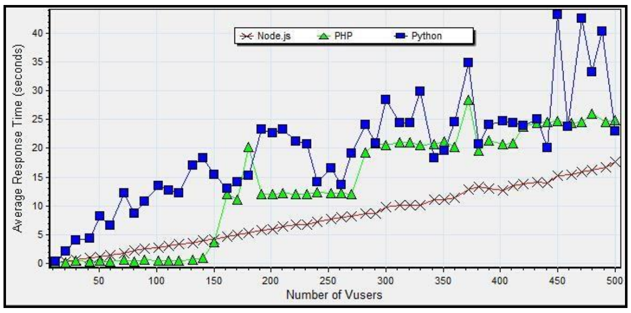

Fig. 14. Average transaction response time

2) “Encryption” scenario

We simulate a real situation to encrypt the password on basis of the "Login" situation to validate the performance of three technologies in a compute-intensive situation. It also makes experiments when users are 500 at max.

FIG.15 shows the results of “hits per second”. It can be seen from the figure that “hits per second" reduces to a large degree when users are up to 50. The decreased time moves up compared to it in the "Login" scenario because the new “encryption” situation is more complex and reduces the performance of PHP. The extension server can undertake is to limit when the users are up to 50. It also can be seen in FIG.17 that the trend of throughput is relative to “hits per second” for PHP.

In FIG.16, the throughput of Node.js is 4,000,000 byte/s, going down 1,000,000byte/s contrasting with FIG.12. From the trend, the throughput of Node.js is kept between 3,000,000 byte/s and 4,000,000 byte/s. All at once, it's steadily falling as users expand. PHP is similar to it in the "Login" scenario when users are less than 50 and the throughput of PHP stays 500,000 byte/s now. Then it goes down by a large degree less than 1,000 byte/s. Python is alike with its performance of the "Login" situation to keep stable but has a slight decline of around 50,000 byte/s. In short, three technologies aren’t adapted to the compute-intensive application, especially PHP. However, the "Encryption” situation brings PHP the least effective among the three. Node.js is more suitable for IO-intensive applications rather than compute-intensive sites.

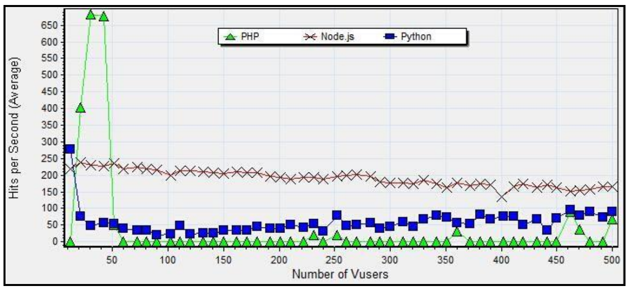

Fig. 15.Hits per second

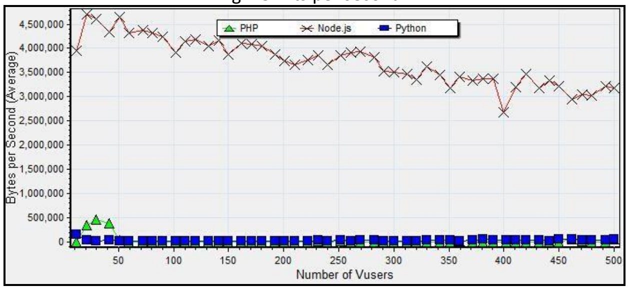

Fig. 16.Throughput

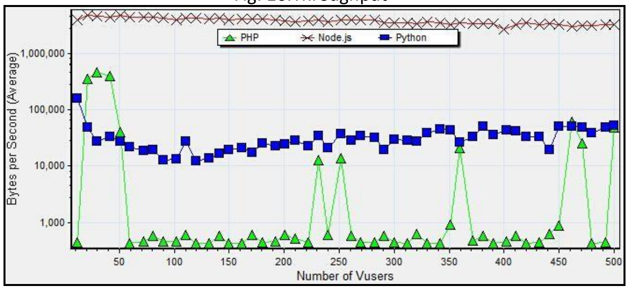

Fig. 17.Throughput trend

The common transaction response time is shown in FIG.18 and the time of PHP shows very unstably with the growing users. On one hand, it's due to the effect of the "Encryption" situation. On the other hand, it's the mode of multi-process in PHP. Nevertheless, the response time of PHP and Node.js are in a good slowly rising trend with the increasing users.

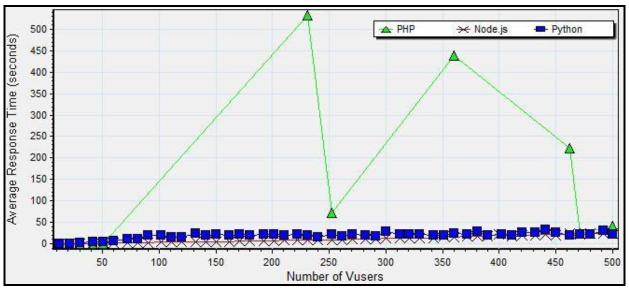

Fig. 18. Average transaction response time

C. The Universal Method

To sum up, PHP applies to small-scale but non- compute-intensive sites, while Node.js is more proper for the IO-intensive and more users' websites. Although Python isn't suitable for compute-intensive and IO-intensive websites, it's maybe the best choice for developers to build large websites due to lots of famous bands like google use it. Node.js can quickly respond to IO requests with its mechanism. It uses an event model based on asynchronous IO, not like multi-processor, responding quickly.

From the research, we achieve a universal way to compare the performance of various Web development technologies. Firstly, we must define the purpose to compare, that's what kind of Web technologies we want to match. After that, we use significant tests to do experiments according to the way one-factor-at-a-time trial design, We can design different conditions in the experiment according to kinds of objectives. Then we compare results in various conditions and modify the experimental ways. At last, we make a judgment what situation those technologies prefer to be used. In addition, we can find the neck of Web development technologies and enhance our method so that we could make the performance better

IV. CONCLUSION

This paper presents an analytical study of three Web techniques. Too much of our knowledge, this is the to examine and test the performance of different Web development technologies including latest technology Node.js from both outside systematic tests (benchmark) and realistic user behavior tests (situation) two aspects. It can get rid of the deviation of a single kind of test and make the results more referenced and practical.

In short, Node.js performs much better than the old technique PHP in high concurrency situations, no matter in benchmark tests or situation tests. PHP handles small requests well but struggles with large requests. Besides, Node.js prefers to be used in the IO-intensive situation, not compute-intensive sites. Python-Web is also not suitable for compute-intensive websites.

In general, Python-Web has many mature frames to develop large-scale websites, like YouTube and Source Forge. Node.js is an emerging technology and has many benefits in an IO-intensive situation, but it’s a little hard for developers who don’t familiar with asynchronous programming. As to PHP, it’s an old technique and popular to be used in short and middle-scale sites.

In our experiments, we only use the most major tests to compare and evaluate the performance of Web technologies. We just consider the technologies, not including the construction design. So our future work is focused on the construction and tries to improve the performance. The paper mainly concerns the comparison of performance, but security and extensibility also require further investigation. With the

REFERENCES

[1] http://Node.js.org/.

[2] http://strongloop.com/developers/Node-jsinfographic/?utm_source=ourjs.com#3.

[3] https://github.com/search?l=JavaScript&o=desc&q=stars%3A%3E1&s= stars type=Repositories.

[4] Tulane, A. Martin, and W. Carey, “Performance Comparison of Dynamic Web Technologies”, ACM SIGMETRICS Performance Evaluation Review, Volume 31 Issue 3, December 2003.

[5] T. Scott, T. Michiaki, S. Toyotaro, T. Akihiko, and O. Tamiya, “Performance Comparison of PHP and JSP as Server-Side Scripting Languages”, Middleware, 2008.

[6] A.Ranjan, Kumar, J.Dhar, “A Comparative Study between Dynamic Web Scripting Languages”, Data Engineering and Management, 2012.

[7] J. Hu, S.Mungee, and Schmidt, “Techniques for Developing and Measuring High-Performance Web Servers over ATM Networks", Proceedings of IEEE INFOCOM, San Francisco, CA, March/April 1998.

[8] Y. Hu, Ananda, and Yang, “Measurement, Analysis, and Performance Improvement of the Apache Web Server", Technical Report No. 1097-0001, University of Rhode Island, 1997.

[9] E.Cecchet, A.Chanda, S.Elnikety, J.Marguerite, and W.Zwaenepoel, “Performance Comparison of Middleware Architectures for Generating Dynamic Web Content", Proceedings of 4th Middleware Conference, Rio de Janeiro, Brazil, June 2003.

[10] U.Ramana, T.Prabhakar, “Some Experiments with the Performance of LAMP Architecture”, Proceedings of the 2005 Fifth International Conference on Computer and Information Technology, 2005.

[11] S.Warner, J.Worley, “SPECWeb2005 in the Real World: Using Internet Information Server (IIS) and PHP”, 2008 SPEC Benchmark Workshop, 2008.

[12] P.Neves, N.Paiva, J.Durães, “A comparison between JAVA and PHP”, C3S2E '13 Proceedings of the International C* Conference on Computer Science and Software Engineering, 2013.

[13] http://www.PHP.net/.

[14] https://www.Python.org/.

[15] R. Jain, “The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling”, John Wiley & Sons, Inc., New York, NY, 1991.

[16] S. Tilkov, S.Vinoski, “Node.js: Using JavaScript to Build High-performance Network Programs”, IEEE Internet Computing, 2010.