2014 IEEE 17th International Conference on Computational Science and Engineering

# Performance Comparison and Evaluation of Web Development Technologies in PHP, Python and Node.js

Kai Lei[1,2,4], Yining Ma[2], Zhi Tan[3]
1. Institute of Big Data Technologies, Peking University, SHENZHEN 518055, P.R.CHINA
2.Shenzhen Key Lab for Cloud Computing Technology & Applications (SPCCTA), School of Electronics and Computer Engineering (SECE), Peking University, SHENZHEN 518055, P.R.CHINA
3. GuangDong Poya Information & Technology Co.,LTD, GUANGZHOU 511400, P.R.CHINA
4. Corresponding author: leik@pkusz.edu.cn
leik@pkusz.edu.cn, clever53@163.com, tanz@poya.com.cn

***Abstract* — Large scale, high concurrency, and vast amount of data are important trends for the new generation of website. Node.js becomes popular and successful to build data-intensive web applications. To study and compare the performance of Node.js, Python-Web and PHP, we used benchmark tests and scenario tests. The experimental results yield some valuable performance data, showing that PHP and Python-Web handle much less requests than that of Node.js in a certain time. In conclusion, our results clearly demonstrate that Node.js is quite lightweight and efficient, which is an idea fit for I/O intensive websites among the three, while PHP is only suitable for small and middle scale applications, and Python-Web is developer friendly and good for large web architectures . To the best of our knowledge, this is the first paper to evaluate these Web programming technologies with both objective systematic tests (benchmark) and realistic user behavior tests (scenario), especially taking Node.js as the main topic to discuss.**

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

## I. INTRODUCTION

In the rapid development of Web today, many sites are faced with new problems, such as the problem multiuser requests and high concurrency. The dynamic scripting language JavaScript has become enormously popular for client and is widely used in Web development. Node.js stands for one new technology in JavaScript. Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications [1]. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices [1]. Node.js popularity surveys performed by official website indicate that the average downloads are over 35,000 since the version 0.10 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 with 133,137 repositories [3]. And talking about evaluation of Web technologies' performance, many researchers have done the related work. But our work differs from others in these two aspects. Firstly, we consider from both objective systematic tests (benchmark) and realistic user behavior tests (scenario) two sides and use the newest commercial testing tool LoadRunner. Secondly, we mainly concern the performance of supporting concurrent users to meet the demand for IO-intensive real-time websites .

This paper focuses on the impact on Web performance from three different Web technologies: Node.js, PHP and Python-Web. The security and scalability issues are beyond the scope of the paper. We mainly use the benchmark tests and scenario tests. In addition, one universal method of Web development technique's evaluation based on the performance comparison is proposed in the paper, which can be used to evaluate any new Web technology.

The main contributions of this paper are listed as follows.

(1)We consider new web technology Node.js in our experiment and analyze the results of it. Then we compare it with PHP and Python-Web, making a conclusion of which situation they ought to be used.

(2)By means of benchmark tests and scenario tests, we can evaluate performance from both objective systematic tests (benchmark) and realistic user behavior tests (scenario). There is often a dual impact on Web server performance, from the calculation, and from the number of users. Our experiment has taken each of these effects in account.

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

## II. RELATED WORK

There have been lots of studies evaluating and analyzing Web server performance. Lance and Martin experimentally evaluated the impact of three different dynamic content

978-1-4799-7981-3/14 $31.00 © 2014 IEEE
DOI 10.1109/CSE.2014.142

technologies (Perl, PHP, and Java) on Web server performance [4]. The results showed that the overheads of dynamic content generation can reduce the peak request rate supported by a Web server up to a factor of 8. Meanwhile, the results indicated that Java server technologies typically outperform both Perl and PHP for dynamic content generation. Scott used the SPECWeb2005 benchmark to contrast the performance of PHP and JSP with the popular Web servers Apache and Lighttpd according to the enterprise standard and made a conclusion that JSP performs better than PHP [5]. Alok compared the performance of ASP.NET, JSP and PHP using 4 benchmarks [6]. J.Hu and Y.Hu evaluated Web Server performance in LAN environments, but they only concerned static Web content [7, 8]. E.Cecchet used two distinct benchmarks, including online bookstore and auction sites, to identify more general performance tradeoffs relevant to any site using dynamic Web content generation [9]. Ramana analyzed the performance differences between PHP and compiled languages such as C Language, pointing out the relative performance downside of PHP [10]. Warner described the importance to use Web technology such as PHP rather than just JSP for real-world benchmarking [11]. Pedro compared from the two aspects of Java and PHP in performance and security, and then made a conclusion that PHP is more scalability and security relative to Java [12].

The research as above focused on the comparison of Web development technologies. But to the best of our knowledge, all of these studies don't consider the newest Web technology, such as Node.js. The paper is the first one to compare the performance of those three popular Web technologies, Node.js, PHP and Python-Web.

## III. Experimental Environment

### A. Hardware Environment

We set up a testbed consisting of two machines connected by 100-Mbps Ethernet, one for the client, the other for the server, including the Web server and Database server, as shown in Fig.1. We deployed Web server and Database server in one machine in order to avoid the impact of bandwidth during the test.

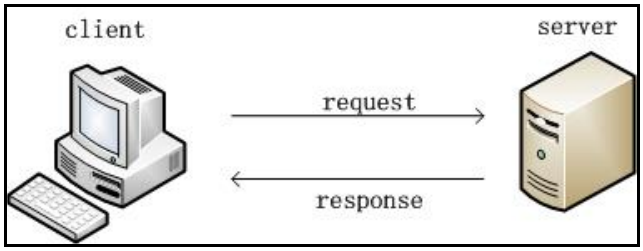


Fig. 1. Testbed setup

The server machine in our test runs Ubuntu Linux 13.04, with an Intel i3 3.30GHz processor, 4 GB of RAM and a 500 GB disk. The client machine runs Windows7 64-bit system, with an Intel i3 3.30GHz processor, 4 GB of RAM and a 500 GB disk. Both machines are connected by 100-Mbps Ethernet. In addition, all non-essential processes on the machines were disabled to minimize the consumption of resources and kept fair in our test.

### B. Server Configuration

It's mainly to compare Web development technologies of PHP, Python-Web and Node.js in our study. So we choose several different modules to build the three environments as follows. The database uses Mysql5.5.

- Apache: Apache 2.4.9. It's the newest stable version. Apache 2.4 uses a hybrid thread and process model in an attempt to improve the server's performance, and it's more scalable compared to Apache 1.3.
- PHP: PHP 5.5.12. The stable version that recently released. PHP is a popular general-purpose scripting language that is especially suited to Web development [13]. PHP is mainly used in dynamic Web page, including CLI (command line interface) and GUI (graphical user interface) program. It has the feature of good across-platforms and easy transplant.
- Python: Python 2.7. Python is a programming language that lets you work quickly and integrate systems more effectively [14]. It's widely used in processing system administration tasks and Web programming since it was born in the 1990s. Its grammar is readable and clear. With the development of several years, PHP has its own frames and easily to develop. In our test, we choose "WebPy" frame to make scenario tests.
- Node.js: Node.js 0.10. Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices [1]. We choose "Express" frame to make tests in our experiment.

### C. Testing Tool

We divided the tests into benchmark tests and scenario tests. We use two testing tools in our experiment in order to make results more reliable. As to benchmark tests, we use ApacheBench (Ab), a tool in Apache. Ab can make requests in local Web server to ensure that the time is just processing time, not including data transmission time on the Internet or calculation time in local machine. Then we choose LoadRunner to make load tests to simulate the behavior of users.

- ApacheBench: The stress testing tool in Apache 2.4. The nature of this kind test is based on Http, and it is a black-box of the Web server performance testing.
- LoadRunner: LoadRunner 11.00. It's also a black-box testing tool. It can record browser behaviors, and simulate real situations to evaluate the performance of Web server.

## IV. TEST METHODOLOGY

The experiment evaluated 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 one-factor-at-a-time experimental design [16] to ensure the accuracy and effectiveness of tests.

### A. Benchmark Test

#### 1) Benchmark Test Methodology

According to one-factor-at-a-time experimental design, we make three fundamental tests – "Hello World", "Calculate Value of Fibonacci" and "Select Operation of DB". "Hello World" module is a basic module to build a good Web server, then output "hello word" and distinguish the differences of those three technologies. "Calculate Value of Fibonacci" module is to calculate some value of Fibonacci and evaluate the performance under compute-intensive tests. "Select Operation of DB" module is to compare different performance through querying some value of DB in the IO-intensive situation. Under all benchmark tests, we keep requests 10000, and then we change users from 10 to 1000. TABLE Ⅰ summarizes the factors in our experiments.

TABLE I. BENCHMARK TEST FACTORS

| | |
|---|---|
| Requests | 10000 |
| Users | 10，100，200，500，1000 |
| Benchmark test module | Hello World, Calculate Value of Fibonacci, Select Operation of DB |
| Web development technology | Node.js，PHP，Python-Web |

#### 2) Benchmark Test Configuration

In the process of test, we found results of same module are similar. For example, we choose PHP to make three tests under requests 10000 and users 100. With the number of requests from 0 to10000, the results of three tests are as shown in Fig.2. The response time doesn't have much difference in three tests with the increase of concurrency requests. The average time of 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 server in every test to make sure more fair in whole experiment.

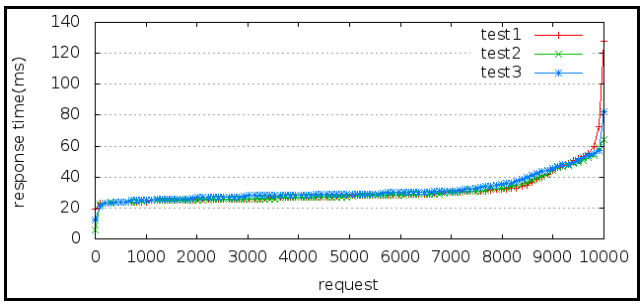


Fig. 2. Three tests of PHP under user 100

In origin 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 TABLE Ⅱ.

TABLE II. THE MODIFIED CODE OF AB

```
bad++;
close_connection(c);
return;
```

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

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
```

In addition, we properly modified the Apache parameters to ensure Apache can deal with requests as large as possible. We modified "MaxClients" to 5000, "ServerLimit" to 5000, and "MaxRequestsPerChild" to 0 in the PreFork mode.

### B. Scenario Test

#### 1) Scenario Test Methodology

Scenario test aims to simulate realistic user behavior . In our experiment, we divide scenario tests into two parts, one is "Login" scenario and the other is "Encryption" scenario. In terms of "Login" scenario, it mainly simulates concurrent users to login at the same time, and then compares the performance of three Web technologies in the real IO-intensive scenario. In the test, we choose 500 users as rendezvous because it appears some errors when users increase to 500. That also means the stress is beyond the maximum 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 performance of those three technologies. Throughput displays the amount of data in bytes the Vusers receive 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. "Encryption" scenario is to simulate a process to encrypt users' login password when users login in. It's mainly to compare performance in the simple real compute-intensive scenario. We choose the same rendezvous as "Login" scenario.

*2) Scenario Test Configuration*

In order to make enough fair in our experiment, we make same configuration as benchmark tests. We reboot server in every scenario test.

## V. EXPERIMENTAL

### A. Results and Analyses of Benchmark Tests

*1) "Hello World" module*

With growth of users, the performance of three technologies shows a trend of increasing before decreasing when keeping the requests at 10000. As FIG.3 and FIG.4 are shown, the "mean requests per second" is the highest which increases to 3703.5 times per second when the users of Node.js are 100. Meanwhile, the "mean time per request" is the shortest 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 around 500 and the highest is 559.42. At this moment, the "mean time 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 "mean time per request" is the shortest 0.336ms. With users increasing, the "mean requests per second" decreases to 200 and remains stable.

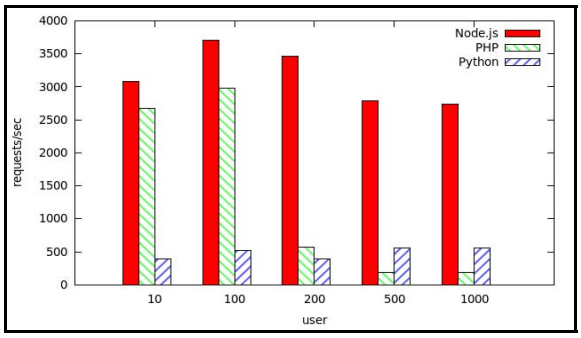


Fig. 3. Results for "Hello world" mean requests per second

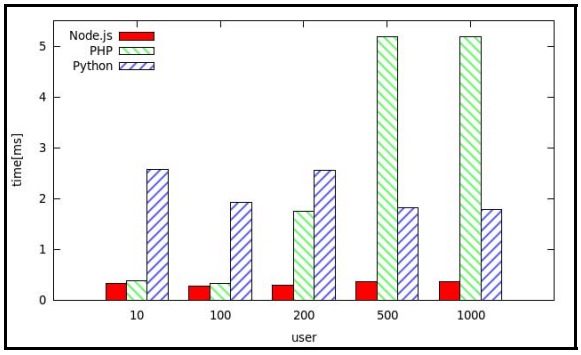


Fig. 4. Results for "Hello world" mean time per request

In short, the performance of Node.js is better than two others at the same number of users. The performance of Node.js is two times larger than PHP on the basic performance comparison 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.

*2) "Calculate Value of Fibonacci" module*

FIG.5 and FIG.6 show the Web performance to calculate the tenth value of Fibonacci when requests are 10000 and users' number increases 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 "mean time 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 peak requests per second decreases 1.5 times comparing with the same condition of "Hello World" benchmark tests. But to Python-Web, this module has similar results as the last module, even better results, at the same condition. PHP is also increased to peak value when users' number is up to 100, the "mean requests per second" at 3127.98. There is not much difference between it and "Hello World" module.

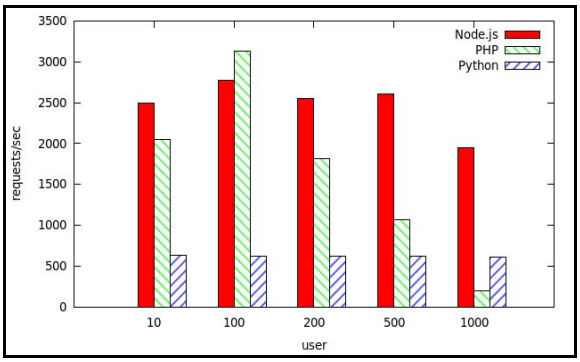


Fig. 5. Results for "Calculate Fibonacci(10)" mean requests per second

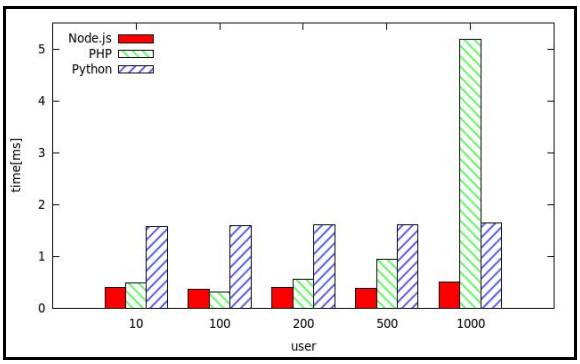


Fig. 6. Results for "Calculate Fibonacci(10)" mean time per request

Obviously, the performance of Node.js is better than two others at "Calculate Fibonacci (10)" tests. But we found they all are very alike to "Hello World" module in the same condition, so we choose other values of Fibonacci to validate the results. We calculate the twentieth and thirtieth value of Fibonacci when users are 10. TABLE Ⅳ shows the results of tests on above. The performance of three Web technologies decreases on different degree with the increase of Fibonacci. Considering the "Calculate Fibonacci (30)" test, all the tests

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

TABLE IV. RESULTS FOR "CALCULATE FIBONACCI (10/20/30))"

| Web development technology | Calculate value of Fibonacci | Mean requests per second [#/sec] | Mean time 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 |

According to the results at above, we make several tests with Node.js in order to find performance difference in various concurrent users as the calculation increases.

FIG.7 and FIG.8 show the results of "Calculate Fibonacci (10/20/30))" as the users grow from 10 to 1000.

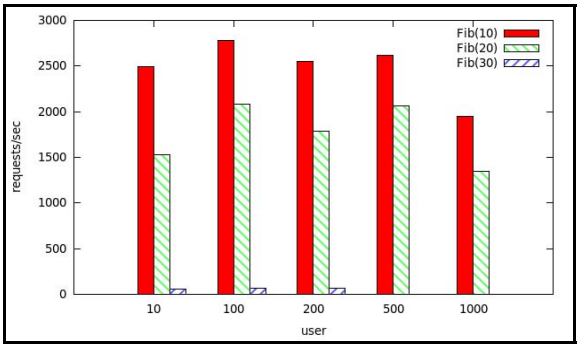


Fig. 7. Results for "Calculate Fibonacci(10/20/30))" mean requests per second with Node.js

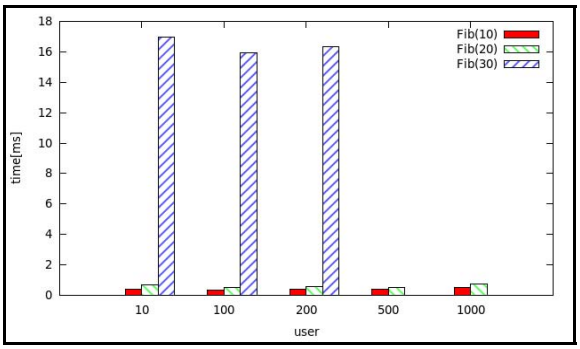


Fig. 8. Results for "Calculate Fibonacci(10/20/30)" mean time per request with Node.js

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

*3) "Select Operation of DB" module*

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

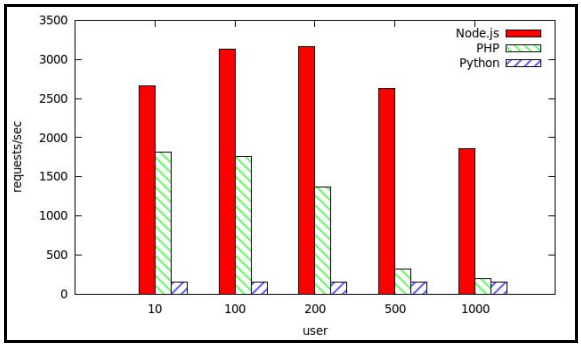


Fig. 9. Results for "Select Operation of DB" mean requests per second

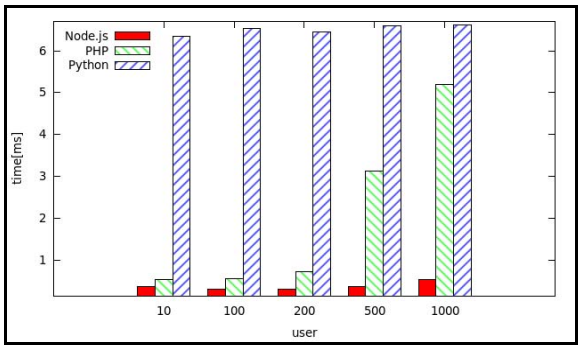


Fig. 10. Results for "Select Operation of DB" mean time per request

The max "mean requests per second" of Node.js is 3164.46, 20 times larger than Python-Web and 2 times larger than PHP. In addition, the peak value of "Select Operation of DB" module with Node.js doesn't have much difference with "Hello World" module. The "mean requests per second" of Python-Web keeps around 150 and maintains stable as users increase. But to PHP, the "mean requests per second" slows down and the "mean time per request" is longer. It proves that Node.js is

more suitable for IO-intensive application among the three, while PHP is applicable to small scale website.

### B. Results and Analyses of Scenario Tests

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

#### 1) "Login" scenario

We choose peak users at 500 to do tests in "Login" scenario. We mainly observe "hits per second", throughput and average transaction response time as users goes up.

From FIG.11 to FIG.14, the results of "Login" scenario are shown. The horizontal axis represent the number of concurrent users, the vertical axis representing hits per second, throughput, throughput trend, average transaction response time.

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

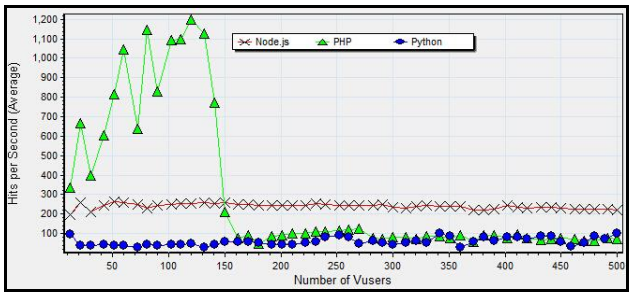


Fig. 11. Hits per second

In addition, throughput in FIG.13 slows down clearly when users are 150. Meanwhile, average transaction response time in FIG.14 increases much and it means the load of system is at a peak. It leads to longer response time with the increase of users, and it appears some users' timeout. So "hits per second" in FIG.11 decreases.

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 a 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 concurrent situation, while PHP is applicable to middle and small scale website.

The average transaction response 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 average 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 increasing. 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 better place when user requests increase.

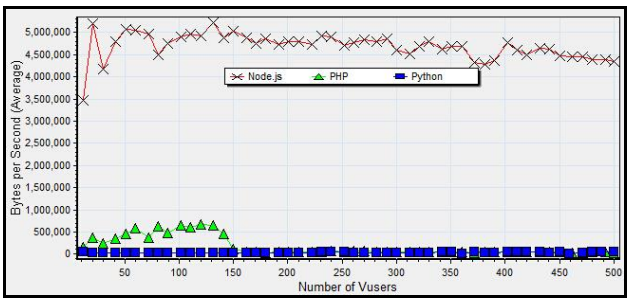


Fig. 12. Throughput

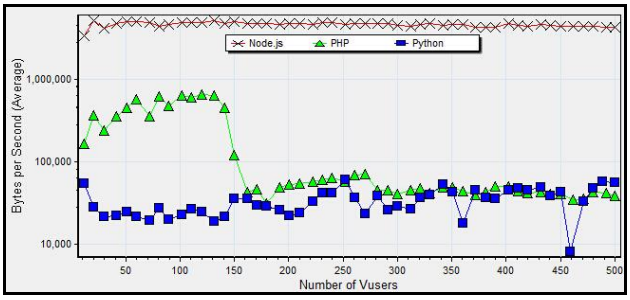


Fig. 13. Throughput trend

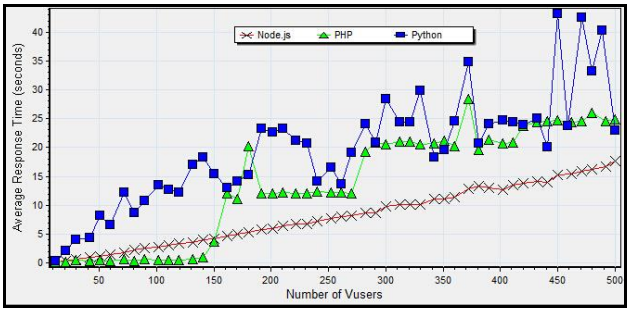


Fig. 14. Average transaction response time

#### 2) "Encryption" scenario

We simulate a real situation to encrypt password on basis of "Login" scenario in order to validate the performance of three technologies in 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" decreases in large degree when users are up to 50. The decrease time moves up compared to it in "Login" scenario because the new "encryption" scenario is more complex and reduces the performance of PHP. The stress 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 keep between 3,000,000 byte/s and 4,000,000 byte/s. All at once, it's steadily falling as users increase. PHP is similar to it in "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 "Login" scenario to keep stable, but has a slight decline around 50,000 byte/s. In short, three technologies all aren't adapted to compute-intensive application, especially PHP. However, "Encryption" scenario brings PHP the least effect among the three. Node.js is more suitable for IO-intensive application rather than compute-intensive sites.

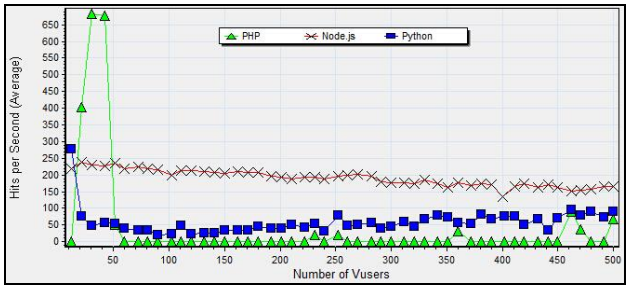


Fig. 15. Hits per second

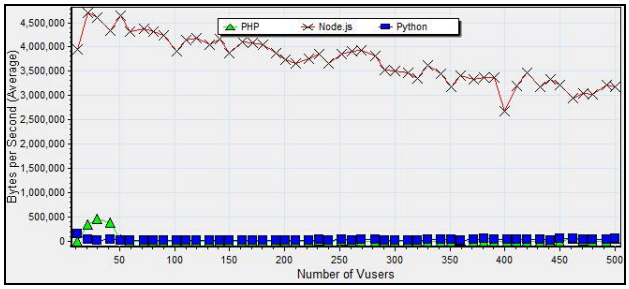


Fig. 16. Throughput

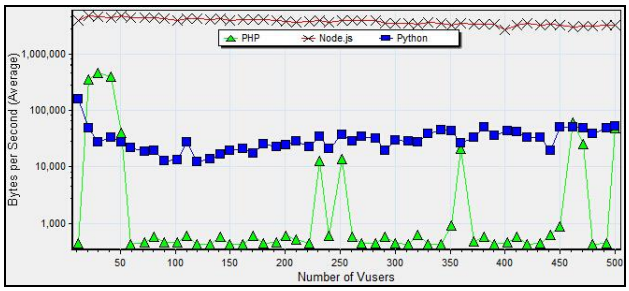


Fig. 17. Throughput trend

The average transaction response time is shown in FIG.18 and the time of PHP shows very unstable with the increasing users. On one hand, it's due to the effect of "Encryption" scenario. 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 increasing trend with the increasing users.

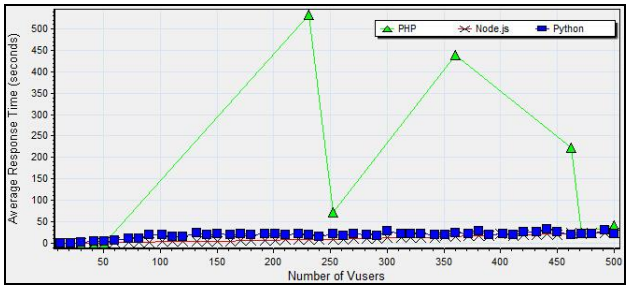


Fig. 18. Average transaction response time

### C. *The Universal Method*

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

From the experiment, we conclude a universal way to compare performance of different Web development technologies. Firstly, we must define the goal to compare, that's what kind of Web technologies we want to compare. After that, we use fundamental tests to do experiment according to the way one-factor-at-a-time experimental design, We can design different situations in experiment according kinds of objectives. Then we compare results in various situations and adjust the experimental approaches. At last, we make a conclusion what situation those technologies prefer to be used. In addition, we can find the neck of Web development technologies and improve our method so that we could make the performance better.

## VI. CONCLUSION

This paper presents a measurement study of three Web techniques. To the best of our knowledge, this is the first paper to compare and analyze performance of different Web development technologies including new technology Node.js from both objective systematic tests (benchmark) and realistic user behavior tests (scenario) two aspects. It can get rid of the deviation of a single kind test and make the results more referenced and practical.

In short, Node.js performs much better than the traditional technique PHP in high concurrency situation, no matter in benchmark tests or scenario 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 the compute-intensive website.

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 advantages in 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 small and middle scale sites.

In our experiments, we only use the most fundamental tests to compare and evaluate the performance of Web technologies. We just consider from the technologies, not including the architecture design. So our future work is focused on the architecture and tries to improve the performance. The paper mainly concerns the comparison of performance, but security and extensibility also requires further examination. With the popularity of NoSQL, we could bring it into our future experiments.

## ACKNOWLEDGEMENT

We are grateful to the anonymous reviewers for their valuable suggestions to improve this paper. This project has been financially supported by National Development and Reform Commission Fund of China ([2013]1309), Shenzhen Gov Projects (No: JSGG20140516162852628 , JCYJ20130331144541058 and JCYJ20130331144416448).

## REFERENCES

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

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

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

[4] T.Lance, 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, R.Kumar, J.Dhar, "A Comparative Study between Dynamic Web Scripting Languages", Data Engineering and Management, 2012.

[7] J.Hu, S.Mungee, and D.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, A.Nanda, and Q.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.