Performance and Scalability Analysis of Node.js and PHP/Nginx Web Application

Yoseph Pandji D. [1]
22084414@students.ukdw.ac.id

Willy Sudiarto Raharjo[2]
willysr@ti.ukdw.ac.id

Abstract

Node.js is an application framework that can be used to build network server and web application. Due to its maturity, Node.js should be tested in various aspects such as performance and scalability to build dynamic web application. For comparison, we use PHP/Nginx web development stack to build web application to compare and analyze Node.js web application performance and scalability.

For research purpose, we build dummy applications based on Dijkstra Algorithm to calculate shortest path between nodes, in this case Trans Jogja shelters. Using load generator, we simulate concurrent user requests to test performance and scalability of Node.js and PHP/Nginx web application. The general results of this research showed that Node.js application had higher performance and scalability than PHP/Nginx application.

Keywords: Node.js, PHP/Nginx, Performance, Scalability

1. Introduction

World Wide Web Technology has rapidly become one of most popular internet tool, with exponentially increasing number of users. To support the increasing number of users,better software is needed. It also presents challenge for software designers to build better performance and scalabilitysoftware. Ryan Dahl has developed an application framework to build network and application server that focused in performance and scalability. Node.js utilize the high performance and scalability V8 Google Chrome Engine as JavaScript interpreter to build network server and also web application.

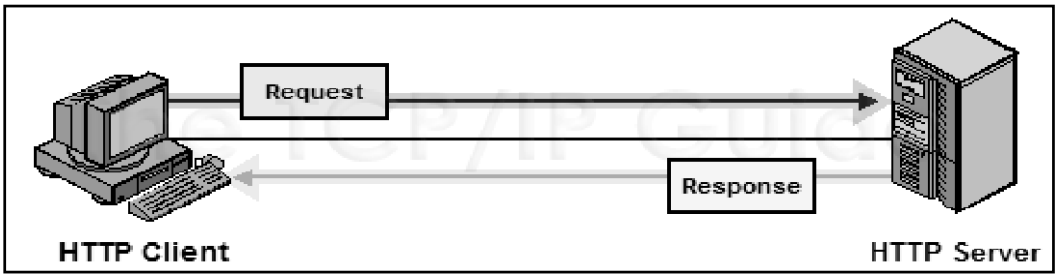
Before this journal was published, (McCune, 2011) has done a research to test the performance and scalability of Node.js Web Server. His research was focused in I/O performance and scalability of Node.js Web Server in comparison with Apache and Ruby Web Server. The results of the research showed that Node.js has better performance and scalability than Apache and Ruby. In the following research, we will focused in testing performance and scalability of Node.js Web Application combined with Node.js Web Server. For comparison, we use PHP/Nginx application development stack to build a web application. Nginx has become third most popular web server in the world and PHP as most popular server side scripting language. From this research, we will see how far Node.js web application can perform and scale compared with PHP/Nginx web application.

[1]*Informatics Department, Faculty of Information Technology, Duta Wacana Christian University, Yogyakarta*
[2]*Informatics Department, Faculty of Information Technology, Duta Wacana Christian University, Yogyakarta*

2. Theoretical Background

2.1. HTTP

HTTP is an application level protocol for distributed, collaborative, and hypermedia *data transmission.* HTTP is a primary protocol that has been used in World Wide Web(WWW) architecture. The standard HTTP Protocol that is currently used is HTTP 1.1 ("Hypertext transfer protocol," June). This protocol use TCP for transport protocol that provides reliable and connection oriented communication. In a HTTP communication, the initiator of communication is a client that transmit HTTP request. After receiving the request, HTTP server will send HTTP response to the client.



Picture 1. HTTP Client-Server Communication

2.2. Node.js

Node.js is an asynchronous JavaScript framework that can be used to build network application. It is a further development of V8 JavaScript Engine that has a built in network and I/O libraries (Tilkov & Vinoski, 2010). The architecture of Node.js application is an asynchronous event driven non-blocking I/O to handle concurrency. Node.js has brought JavaScript into server environment, being a server side language. Using Node.js, developers can build his own HTTP server and web applications with one language, JavaScript.

2.3. Nginx / PHP

Nginx is a general purpose HTTP Server. Developed by Igor Sysoev, it was first published in 2004. Nginx use asynchronous event driven non-blocking I/O architecture to handle concurrency. Nginx run two kinds of processes in the operating system: worker process for receiving, sending, forwarding the request and master process for worker process management. It is possible to run multiple worker processes with one thread each process so Nginx can utilize multi core processors. Nginx can be combined with PHP to serve dynamic content via Fast CGI. When request has been accepted by server, Nginx will forward the request to PHP interpreter with proxy mechanism. After PHP interpreter finished the process, the result will be forwarded back to Nginx and it will forward the results in form of HTTP response. Combination of Nginx and PHP is also called an application web development stack.

2.4. Dijkstra Algorithm

Dijkstra Algorithm is a graph algorithm founded by E.W Dijkstra. This algorithm is an algorithm to solve optimum path problems between nodes in a graph. This algorithm can also being used to solve optimum path from one node to all nodes in a graph concurrently. Here is the step by step in calculating shortest path problem using Dijkstra Algorithm:

$$G = (N, E) \quad [1]$$

Where,
G = Graph
N = Node in the graph
E = Edge (path weight in graph)

While calculating the shortest path, there are several data structures that are used to complete the algorithm:
Dist. : Array that contains nodes where the shortest path from start node has been decided
Queue : Array that contains remaining edge that have not been checked
Prev : Array that contains shortest path estimation from all nodes

Here is the algorithm pseudo code:

a. SET Prev and Queue variable
b. SET Dist to 0,
c. IF there are edges in graph THEN,
d. Shortlist edges in Queue based on shortest path from start node
e. Add u andthe closest edge in Queue to Dist
f. Check if all edges in Queue are connected to u.

2.4. Performance and Scalability

"Performance measures how fast and efficiently a software system can complete certain computing tasks, while scalability measures the trend of performance with increasing load."(H.Liu, 2009). For a given environment that consists of hardware and properly configured operating system, if the performance of a software system can't be improved or continue to deteriorate rapidly even with hardware upgrade, then the software system is not scalable.
While conducting performance and scalability test, we use these metrics:

a. Response Time:the time spent by the client send a request until the response is received. It is quantified in ms.
b. Throughput:How many requests that can be served by server for a given time. It is quantified in request/sec.
c. ResourcesUsage:How much resources are being used when system doing its task. It consists of processor usage (% total processor time), RAM usage (%total RAM usage), and network traffic (KB/sec).

To estimate the concurrent user, we use little law to quantify how many concurrent user that are for a given load level.

$$R = \frac{Nvu}{Xo} - Z \qquad [2]$$

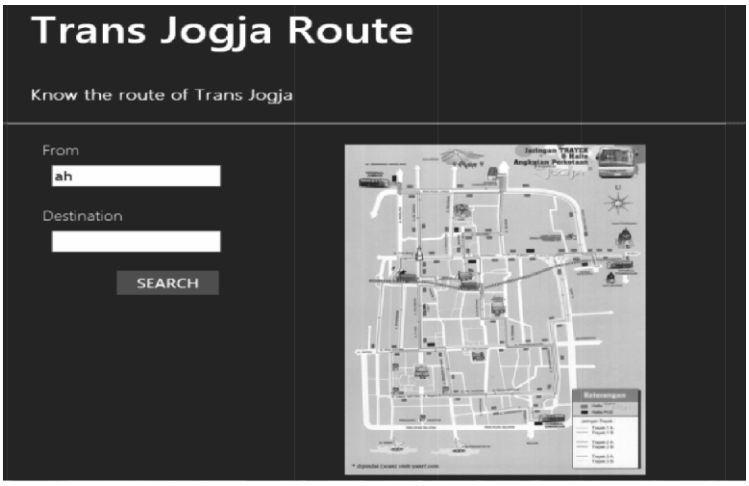
Where,

R represents average response time under the given load level
Nvu represents concurrent user under the given load level.
Z represents the average user think time under the given load level.
Xo represents average throughput under the given load level.

3. Results And Discussion

3.1 System Implementation

We built web applications using Node.js and PHP as server side language. The web applicationsis a system to calculate shortest path between Trans Jogjashelters.Client will send HTTP request that contains two parameters: start shelter and destination shelter. The application that resides on the server will process the request, calculate the shortest path and send the response that contains list of shortest path from start shelter to destination shelter.



Picture 2. Choose Start and End Shelter User Interface

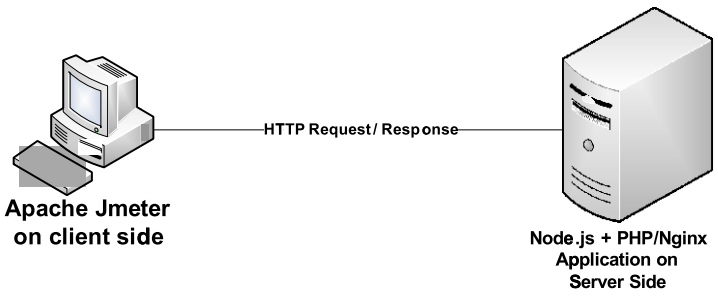
User inputs the start and destination shelter and then click the submit button to send the HTTP request to server. The difference between these two applications is the web server and back end engine that generates the contents.

Picture 3. Shortest Path List Page

After receive the request that contains start and destination shelter, the applications will operate shortest path calculation using Dijkstra Algorithm. After it is finished, server will send the response that contains shortest path list web page back to the client.

3.2. Testing Methodology

For performance and scalability testing, we use point to point network topology, a directly connected client and server. As a load generator that simulates concurrent HTTP request, we use Apache Jmeter in client side. Apache Jmeter will simulate concurrent communication channel that transmit HTTP request and receive HTTP response.



Picture 4. Network Topology for Testing

When conducting performance testing, we configure Apache Jmeter to simulate HTTP Request that are sent by 1000 users. There are 4 kinds of HTTP Request that represent 4 longest path from all possible shortest path. To analyze the data, we use Z-test to compare the average response time and throughput results with 30 times sampling.

When conducting scalability testing, we configure Apache Jmeter to simulate 4000 concurrent HTTP requests. The increasing load will be implemented in load time that increase from 30 second until the performance of applications isdegraded.

3.3.Performance Test Results

a. Response Time.

From performance test results, we can see that in all scenarios, the average response time of Node.js application was lower than the average response time of PHP/Nginx. In all scenarios, the average response time of Node.js application were in the range of 4 ms while PHP/Nginx application were in the range of 8 ms.

Table 1.
Average Response Time

Scenario	Node	Nginx
1	4.566666667	8.966667
2	4.366666667	8.9
3	4.3	8.466667
4	4.266666667	7.833333

From hypothesis test result, we can see that average response time of Node.js application were lower than PHP/Nginx in all scenarios.

Table 2. Hypothesis Testing Results of AverageResponse Time

Scenario	Hypothesis Test Result
1	The average response time of Node.js application was lower than average response time of PHP/Nginx application.
2	The average response time of Node.js application was lower than average response time of PHP/Nginx application.
3	The average response time of Node.js application was lower than average time of PHP/Nginx application.
4	The average response time of Node.js application was lower than average response time of PHP/Nginx application.

b. Throughput.

From performance test results, we can see that in all scenarios, the average throughput of Node.js application was lower than the average response time of PHP/Nginx.

Table 3. Average Throughput

Scenarios	Node	Nginx
1	336.3994	331.6277
2	332.035	331.8205
3	328.8786	334.1049
4	333.509	335.0806

From hypothesis test result, we can see that average throughput of Node.js application were lower than PHP/Nginx in all scenarios.

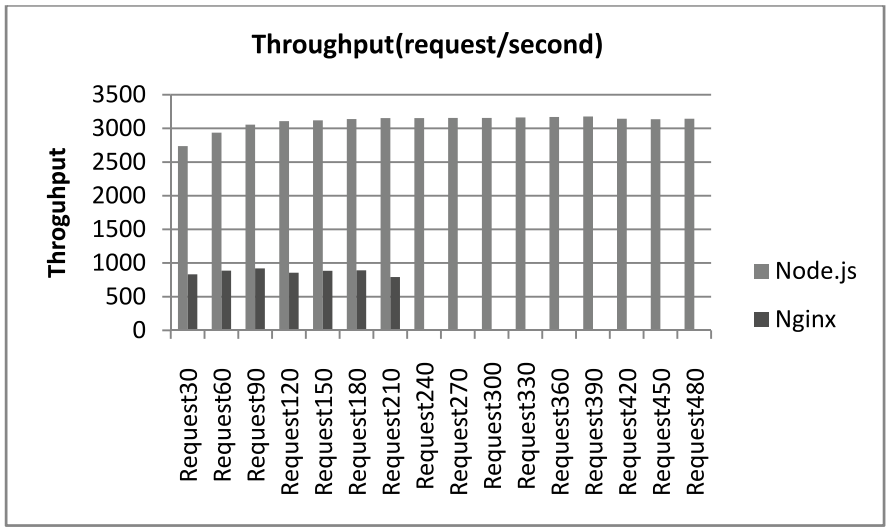
Table 4. Hypothesis Testing Results of Throughput

Scenario	Hypothesis Test Result
1	The average throughput of Node.js application is lower than average throughput of PHP/Nginx application.
2	The average throughput of Node.js application is lower than average throughput of PHP/Nginx application.
3	The average throughput of Node.js application is lower than average throughput of PHP/Nginx application.
4	The average throughput of Node.js application is lower than average throughput of PHP/Nginx application.

3.4.Scalability Test Results

a. Maximum Throughput

From the test results, it seems that the maximum throughput of Node.js application was around 31746.434263 request /second with the average response time1190 was ms. The maximum throughput of PHP/Nginx application was around 920.1226039 request/second and the average response time was 1074 ms. Scalability testing for Node.js application was stopped at 480 second load time because the application performance was degraded since the load time is at 390 second. Scalability testing for PHP/Nginx application was stopped at 210 second load time because the RAM usage was beyond capacity and the server started to crash.



Picture 5. Average Throughput in Scalability Testing

Using little law, we could estimate the concurrent users that stressed the Node.js application (Nvu) :

$$Nvu = \frac{3176.434263 \times 1190}{1000} = 3779.95677297$$

From the calculation, we concluded that maximum performance of Node.js application is when there are 3780 concurrent that are actually stressing the system with average throughput is around 3176.434263 request / second and average response time is around 1190 ms.

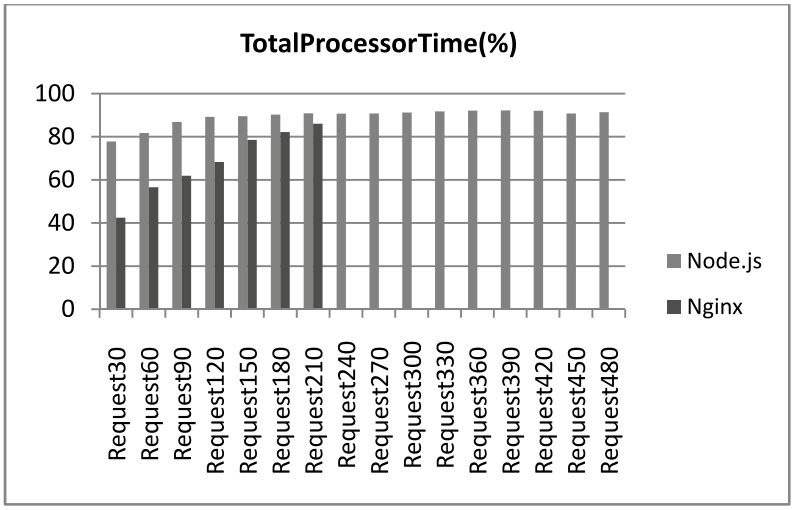
Maximum throughput of PHP/Nginx application was around 920.1226039 request / second with average response time around 1320 ms. Using Little Law, the estimation of concurrent users that stressed the PHP/Nginx application (Nvu) :

$$Nvu = \frac{920.1226039}{1000} = 1214.561837148$$

From the calculation, we concluded that maximum performance of PHP/Nginx application is when there are 1215 concurrent users that are actually stressing the system.

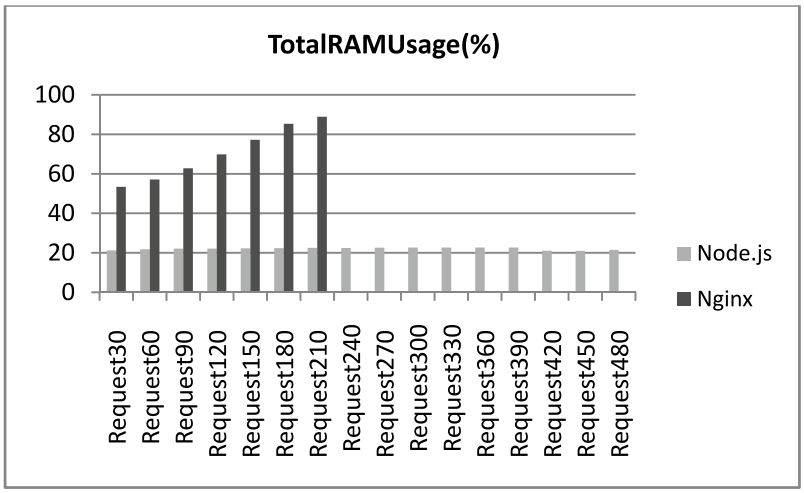
b. Resource Usage

From processor usage perspective, the average processor usage of Node.js application was beyond 70% when the load time was 30 second and continued to increase until it reach around 92% while processor usage of PHP/Nginx application was around 42% and continued to increase until the server started to crash.



Picture 6. Processor Usage

The RAM usage of Node.js application was constantly around 23 % while the PHP/Nginx continue to increase until the server started to crash. The high usage of RAM was the reason of application performance degradation.



Picture 7. Total RAM Usage

3.5. Analysis

a. Performance

In all scenarios, Average response time of Node.js application are lower than PHP/Nginx application. It means that the speed of Node.js application to process all the http

request is faster than PHP/Nginx application. In other hand, average throughput of Node.js Application is higher than PHP/Nginx application.

b. Scalability

From throughput results, we can see that Node.js application's throughput is higher than PHP/Nginx application. Node.js application can handle more requests concurrently than PHP/Nginx application did. Performance of Node Application continue to increase when the load is increasing whereas PHP/Nginx application got saturated when the load is increasing.

From the resource usage, the usage of processors of those two applications was getting increased, but if we see from the average throughput that can be reached, Node.js application could continue to scale while PHP/Nginx application started to deteriorate. The overloaded RAM usage has been the main reason why PHP/Nginx performance was degraded.

4. Conclusion

In term of performance, Node.js application response time was lower than PHP/Nginx application, it means that Node.js application had faster responsiveness than PHP/Nginx application. In term of performance, PHP/Nginx application could handle more request than Node.js application. PHP/Nginx application had higher throughput than PHP/Nginx. In term of scalability, Node.js application can handle more request than PHP/Nginx. Throughput of Node.js continue to increase when the workload increase, in other side PHP/Nginx throughput was saturated when the workload increase.In term of scalability, Node.js application used resource more efficient than PHP/Nginx application especially in RAM usage. Node.js application used slightly lower RAM usage than PHP/Nginx application.

References

Hypertext transfer protocol. (June, 1999). Retrieved from http://www.ietf.org/rfc/rfc2616.txt (last accessed April 2, 2012).

H.Liu, H. (2009). *Software Performance and Scalability: A Quantitative Approach*. Wiley.

McCune, R. R. (2011). *Node.js Paradigm and Benchmark*. Informally published manuscript, University of Notre Dame, Indiana

Tilkov, S., & Vinoski, S. (2010). Node.js: Using JavaScript to Build High-Performance Network Programs. *Internet Computing, IEEE*, *14*(6), 80 - 83. doi: 10.1109/MIC.2010.145