Scientific Journal Impact Factor (SJIF): 1.711
e-ISSN: 2349-9745
p-ISSN: 2393-8161
International Journal of Modern Trends in Engineering and Research
www.ijmter.com

PERFORMANCE INVESTIGATION OF NODE.JS

Komal Paliwal
M.Tech. Scholar, Mewar University,Chittorgarh, India

Abstract— The current environment of web applications requests execution and adaptability. A few past methodologies have actualized threading, occasions, or both, yet expanding activity require new server for enhanced simultaneous request. Node.js is another web structure that accomplishes both through server-side JavaScript and occasion driven I/O. Tests will be performed against two equivalent framework that compare service request times over various core. The outcomes will exhibit the execution of JavaScript as a server-side language and the effectiveness of the non-blocking non concurrent model. The paper examine that node.JS is a suitable structure for improvement of adaptable web servers , can be scaled and distributed across multiple nodes utilizing clustering and replication system.

Keywords-Node, Scalability, Performance, Non Blocking, Apache.

I. INTRODUCTION

In this time of computer and web, element web is the universal hotspot for standardizing; news redesigns to give some examples amongst numerous other uncountable utilizations [1]. With the approach of social networking destinations, immense measure of individuals have been going by and investing time all the while on such sites. This represents the necessity that site need to be speedy in execution, versatile and distributed despite of expansive number of simultaneous clients. Presently, we can take facebook and google+ as a portion of the cases of well known social websites to get the thought of how they have to be scalable without degradation of performance to help extensive number of simultaneous clients and to keep the users interested on visiting sites more frequently. Along these lines, to stay away from the client dissatisfaction on utilizing web application require smoothness of access and support of any number of concurrent user without execution performance degradation . In this paper, test driven improvement of a web application has been completed utilizing generally new programming framework called node.JS(server side JavaScript technology) . Node.js which is suitable language for development of scalable network application has been used to develop a prototype. Node.js is one recent framework to implement the event model through the entire stack. Developed in 2009 by Ryan Dahl, Node.js (or just Node) is a single-threaded server-side JavaScript environment implemented in C and C++ [7]. Nodes architecture makes it simple to use as an expressive, functional language for server-side programming that's is well known among developers [7]. Node uses the JavaScript V8 engine, developed by Google [9], a quick and effective usage of JavaScript[8] that helps Node accomplish top performance. In the following examinations, Node will be contrasted with Apache, a multithreaded web server, and Even machine, an evented Ruby web server, to evaluate practical web framework for the difficulties postured by the current web. As demonstrated in figure1 node.JS makes utilization of event loop to handle events and line the call-backs connected with events. V8 is high performance java script engine written in C++ and embedded in chrome which is the core design component for node.JS. JavaScript layer is permitted to get to the main thread just though C layer is open for multithreading. Multithreaded programming methodology is simple and proficient approach to make utilization of numerous cores for

concurrency yet it has a few pitfalls like deadlocks, accessibility of shared resources and successive exchanging of procedures. Though Event driven model of node.JS gives more adaptable solution of exchanging between tasks making utilization of event notification functionalities of hidden OS like select(), poll() alongside epoll, kqueue, kevent calls [10].

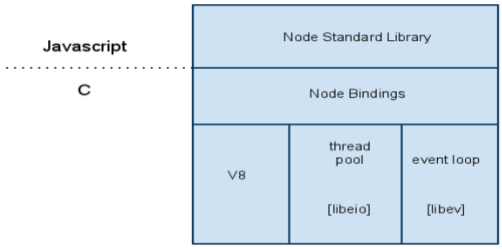

Figure 1 Node.JS architecture stack courtesy

II. RELATED WORK

Having been released in 2009, Node is still beta release and currently lacks much peer-reviewed literature. The development community has posted several experiments on the web that support the evented approach [10, 11, 12], but Node still lacks a definitive evaluation. Conversely with cutting edge framework, the debate between threading and events has a since a long time ago. The requirement for event based architectures has been identified in recent years [5, 13, 14] while thread based concurrency stays predominant in recent server applications [15]. Both thread and event methodologies have continuously advanced. Apache uses limited thread pools, where the number of open thread is topped in order to limit resource allocation [3]. Additionally, event driven concurrency has been refined, including the event queue to simplify handling and modularize code [3]. All the more as of recent, scientists out of UC Berkeley found a mixture approach in SEDA that uses both threads and events to make a hybrid arranging part to modularize web architectures [3]. The methodology performs well for high volume yet adds extensive many-sided quality to a framework. While critical, SEDA is not appropriate to current investigations.

III. THE PROGRAMMING MODEL

Node's I/O approach is strict: nonconcurrent collaborations aren't the exception; they're the standard. Each I/O operation will be taken care of by method for higher-request functions that is, functions taking functions as a parameter that detail what to do when there's something to do. In just uncommon circumstances have Node's developers included a convenience function that works synchronously for instance, for evacuating or renaming documents. Be that as it may, by and large, when operations that may require network or file I/O are invoked, control is quickly returned to the caller. At the point when something fascinating happens for instance, if information gets to be accessible for reading from a network socket, a yield stream is prepared for composing, or a slip happens — the suitable get back to capacity is called. Figure 2 is a straightforward sample of executing an HTTP Web server that serves static files from disk. Indeed to non-Web developer, Java-Script's syntax ought to be genuinely clear for those with prior introduction to any C-like language. One of the more particular topics is the function (...) structure. This creates an anonymous function: JavaScript is a functional language and, as such, helps higher-order functions. A developer writing or taking at a Node program will see this all over. The program's main flow is determined by the functions that are explicitly called. These functions never block on anything I/O-related, yet rather register appropriate handler callbacks. If you've seen a same concept in eventing libraries for different programming languages, you might wonder where the explicit blocking call to invoke the event loop hides. The event loop concept is so core to Node's behaviour that

it's hidden in the implementation; the main program's aim is simply to set up appropriate handlers. The http. Create Server function, which is a cover around low-level efficient HTTP rules of conduct putting into use, is moved a function as the only argument. This function is called whenever data for a new request is ready to be read. In different environment a simple implementation might create the effect of eventing by synchronously reading a file and sending it back. Node serves no opportunity to read a file synchronously the only way is to register another function via read File that gets invoked whenever data can be read [4].

```
var sys = require("sys"),
  http = require("http"),
  url = require("url"),
  path = require("path"),
  fs = require("fs");
http.createServer(function(request, response) {
  var uri = url.parse(request.url).pathname;
  var filename = path.join(process.cwd(), uri);
  path.exists(filename, function(exists) {
    if(exists) {
      fs.readFile(filename, function(err, data) {
        response.writeHead(200);
        response.end(data);
      });
    } else {
      response.writeHead(404);
      response.end();
    }
  });
}).listen(8080);
sys.log("Server running at http://localhost:8080/");
```

Figure. 2. Events trigger functions that execute input or output operations

IV. DEBUGGING & PERFORMANCE ANALYSIS

While dynamic language is to a great degree well known for fast improvement, they are likewise famously hard to troubleshoot underway situations. Notwithstanding being a relative newcomer, Node.js has officially created modern framework for both runtime investigations, frequently surpassing those of adult situations like Java. Such tools are getting to be progressively basic as the developing ubiquity of distributed computing frameworks. I will examine top to bottom our work building and conveying open source tools to encourage investigating and runtime profiling of Node.js projects underway. I will explain real issues we have diagnosed utilizing these tools that we were not able to illuminate with existing systems and afterward impart a portion of the best practices I've figured out how to support examination. I will then examine DTrace help for Node.js, which empowers developers to profile self-assertive Node programs underway to comprehend where projects invest their time on CPU. At last, we will demonstrate a percentage of the novel visualizations I've grown (alongside the open source devices to make them) that total and present this substantial measure of execution information. I have observed that Node.js scales truly well. The non-blocking event loop considers an extraordinary measure of activity contrasted with our old, very enhanced PHP stack going through Apache. With PHP and PostgreSQL we could scale up, yet it felt truly hard and provided for us numerous restless evenings. Utilizing Node.js with a MongoDB backend scaling up is brisk and simple; but since Node can deal with more movement, you don't have to as fast. For correlation and execution investigation I have utilized node.js 0.1.117 on one side, and Apache with prefork MPM and PHP on the other, running them with ApacheBench and assemble of 100,000 solicitations with 1,000 simultaneous requests amid first test [4]:

ab -r -n 100000 -c 1000 <url>

1,000,000 requests and 20,000 concurrent requests:

ab -r -n 1000000 -c 20000 <url>

"Shrinathji Baba" node.js server used for implementation & testing and equally basic "Shrinathji Baba" PHP file for Apache as shown in Table 1 shows

TABLE 1.CODE FOR NODE.JS VS PHP

Testing code using node.js server	**Testing code using PHP file**

var sys = require('sys'),
http = require('http');
http.createServer(function(req, res) {
res.writeHead(200, {'Content-Type': 'text/html'});
res.write(' Shrinathji Baba ');
res.end();
}).listen(8080);
	<?php
	echo ' Shrinathji Baba ';

TABLE 2. NODE.JS & APACHE RESULT

node.js results	**Apache results**
Concurrency Level: 1000 Time taken for tests: 21.162 seconds Complete requests: 100000 Failed requests: 147 (Connect: 0, Receive: 49, Length: 49, Exceptions: 49) Total transferred: 8096031 bytes Requests per second: 4725.43 [#/sec] (mean) Time per request: 211.621 [ms] (mean) Transfer rate: 373.61 [Kbytes/sec] received Connection Times (ms) min mean[+/-sd] median max Connect: 0 135 821.9 0 9003 Processing: 1 40 468.5 25 21003 Waiting: 1 30 64.1 25 12505 Total: 2 175 949.1 26 21003 Percentage of the requests served within a certain time (ms) 50% 26 66% 33 75% 36 80% 39 90% 55 95% 94 98% 3030 99% 3090	Concurrency Level: 1000 Time taken for tests: 121.451 seconds Complete requests: 100000 Failed requests: 879 (Connect: 0, Receive: 156, Length: 567, Exceptions: 156) Total transferred: 29338635 bytes Requests per second: 823.38 [#/sec] (mean) Time per request: 1214.510 [ms] (mean) Transfer rate: 235.91 [Kbytes/sec] received Connection Times (ms) min mean[+/-sd] median max Connect: 0 38 321.8 20 9032 Processing: 0 565 5631.0 51 121380 Waiting: 0 262 2324.1 41 52056 Total: 29 603 5641.7 73 121431 Percentage of the requests served within a certain time (ms) 50% 73 66% 78 75% 82 80% 83 90% 89 95% 105 98% 4251 99% 13205

To explain & draw CPU and memory load variation I have calculated them during tests using dstat with following results:

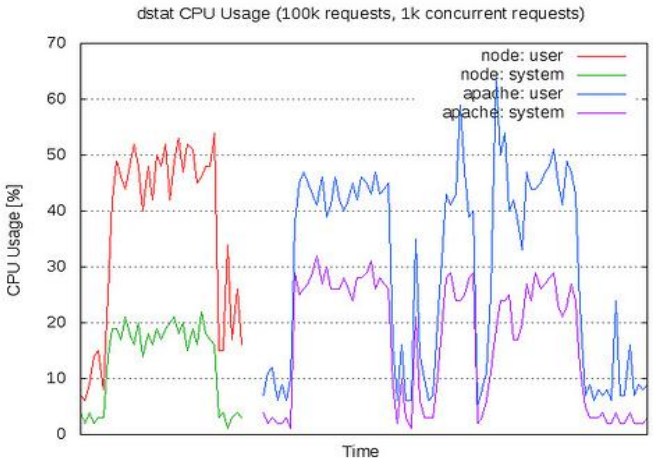

Fig 3 Node.js vs Apache/PHP CPU Usage

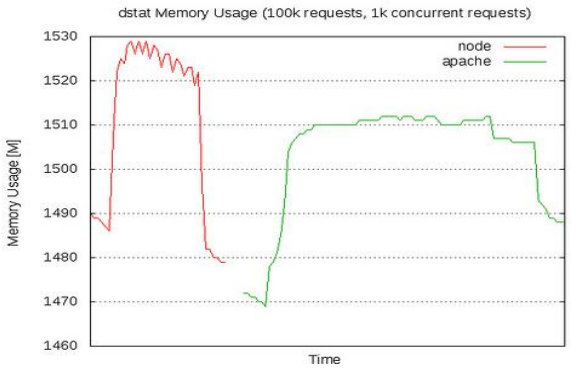

Fig 4 Node.js vs Apache/PHP Memory Usage

V. STRENGTH OF NODE.JS

Of course, with Node.js being in writing in JavaScript, if you currently know JS for frontend development, you are doing pretty well in the Node.js realm. You currently understand the syntax; all that is left is an API for interacting with the system. The package management system, npm, is great (although the website departs much to be desired). Instead of having to follow strict guidelines and formatting obligations (e.g. PHP's PEAR), any person can put anything into npm (even me!). It is alike to the Android market. Certain, you can get some crappy things out of it, but if one utilizes widespread sense they won't be downloading a virus [26].Server constructed in, the stack is a allotment easier, there are few points of downfall, and there is more rule over what you can do with HTTP answers (ever try overwriting the name of the world wide world wide world wide web server using PHP?). Node.js biggest power, IMO, is that it is happening driven. Node apps run large over long periods of time. The happening emitter cipher, while attractive easy at its centre, provides a mighty and reliable interface for triggering cipher execution when required. It has a web server built in. Having the server constructed in means that you doesn't have the inapt .htaccess config thing going on. Every demand is appreciated to proceed through the identical process, without having to hunt through the document scheme and figure out which script to run. The number one hindrance with web application is not the time it takes to measure CPU hungry action, but alternatively mesh I/O. If you need to reply to a client call after doing a database call and dispatching an internet message, you can present the two actions and reply when both are complete.

VI. CONCLUSION

Experiments were conducted to evaluate Node's implementation of the evented-I/O model against another framework. While Node outperformed the competition in these experiments, further work can provide a clearer picture of Node's strengths and weaknesses. Node accomplishes its goals of supplying highly-scalable servers. It uses an exceedingly very quick JavaScript engine. It benefits an Event-Driven design to hold code negligible and easy-to-read. All of these components lead to Node's yearned goal it's somewhat very simple to compose a massively-scalable solution. Just important as comprehending what Node is, it's also significant to realize what it is not? It is not simply a replacement for Apache that will instantly make your PHP world wide World Wide Web application more expandable. That couldn't be farther from the reality. It's still new in Node.js life, but it's very often, the community is very energetically quickly as needed, there are a lot of nice modules being understood, and this growing merchandise could be in your business inside a year. It is used to offer better combine with third party support libraries. Many third party programmers have developed plug-in for it, including adding file server support and MySQL support. Look for it is used to start integrating that into the core functionality. Eventually, I also expect this sever to support some type of dynamic page module, whereby you could do the types of things in an HTML file that you can do in PHP and JSP's. Lastly, at few places expect a ready-to-use this fast server that you will download and install, and simply drop your HTML files in to as you can with Apache or Tomcat. It's growing quickly and could be on your horizon soon. It is also about improving your PHP code. An important step throughout this work is refactoring and improving your PHP code such that it is easier to convert it to Node.js code. This thesis isn't just about making a new Node.js codebase. It is about improving your PHP codebase and creating a new Node.js. It is about both PHP language and your Node.js codebase. Converting your PHP codebase to Node.js can make you a better PHP developer. It expressly compares it to PHP so you can see the nuts and bolts of what it examines like in the language that you understand as well as in the language you are discovering. You might even find a few corners of PHP you weren't previously cognizant of, because a few of those PHP corners are centred notions in Node.js. Even better, by matching Node to an exact distinct language, such as PHP, it will give you a good idea as to how much of Node is the identical as PHP. Comparing two languages or, even better, showing how to convert or dock from one language to another, is a most powerful way to become a professional in both languages. It is codebases would be completely purposeful and, going ahead, new characteristics and bug repairs could be evolved on both codebases simultaneously. This conclusion avoids a much simpler approach, which would have been a "convert-and-discard" alteration method where the PHP would be unsynchronized and probably not employed at the end of the conversion method and the developer's only option would be to advance ahead with the its codebase by itself. Now more than ever it is imperative to keep up-to-date with the freshest tech and to consistently invigorate your company's place in the digital space. Routinely pushing bleeding-edge expertise, particularly in a large business, carries a component of risk. It is thus vital to accept in mind the technological and financial benefits that this directs shows when considering the adoption of Node.js as your new server-side dialect for world wide World Wide Web applications. Once you get utilized to event-based async programming, resolve on a set of benchmark development patterns and an architectural style. You'll rapidly start to reap the benefits of employed solely in JavaScript. This direct demonstrates not only why Node.js is a fascinating, popular and ingenious solution but furthermore an extremely time and cost productive one. Though this thesis has covered performance of node.js in future we doing implementation of World Wide Web submission using node.JS and scalability of database, there are more research areas that could be realized as a future extension.

ACKNOWLEDGMENT

I would like to thank Mr. B.L Pal for his support, guidance, and patience he gave throughout writing this paper. Their encouragement and dedication is numerous to mention.

REFERENCES

[1] Labovitz, C., Iekel-Johnson, S., McPherson, D., Oberheide, J., and Jahanian, F. Internet Inter-Domain Traffic. SIGCOMM '10 (2010).

[2] L. A. Wald and S. Schwarz. The 1999 Southern California Seismic Network Bulletin. Seismological Research Letters, 71(4), July/August 2000.

[3] Matt Welsh, David Culler, and Eric Brewer, "SEDA: An Architecture for Well-Conditioned, Scalable Internet Services", ACM Symposium on Operating Systems Principles, 2001.

[4] Benchmarking Node.js - basic performance tests against Apache + PHP.

[5] John Ousterhout, "Why Threads are a Bad Idea (for most purposes)", talk given at USENIX Annual Conference, September 1995.

[6] 2011 Scalable web application using node.JS and CouchDB.

[7] Tilkov, S., Vinoski, S. Node.js: Using Javascript to Build High-Performance Network Programs. Internet Computing, IEEE, 2010 STRIEGEL, GRAD OS F'11, PROJECT DRAFT 6.

[8] Paruj Ratanaworabhan, Benjamin Livshits, and Benjamin Zorn. JSMeter: Comparing the behavior of JavaScript benchmarks with real web applications. In USENIX Conference on Web Application Development (WebApps), June 2010.

[9] Google Javascript V8, http://code.google.com/p/v8/, accessed 11/11.

[10] http://zgadzaj.com/benchmarking-nodejs-basic-performancetests against-apache-php accessed 10/26/11.

[11] http://teddziuba.com/2011/10/node-js-is-cancer.html accessed10/26/11.

[12] http://hns.github.com/2010/09/21/benchmark.html accessed 11/18/11.

[13] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An Efcient and Portable Web Server. In Proceedings of the 1999 Annual Usenix Technical Conference, June 1999.

[14] M. Welsh, D. E. Culler, and E. A. Brewer. SEDA: An architecture for well conditioned, scalable Internet services. In Symposium on Operating Systems Principles, pages 230243, 2001.

[15] Sun Microsystems. RPC: Remote Procedure Call Protocol Specication Version 2. Internet Network Working Group RFC1057,June 1988

