A Comparative Analysis of New Cow Concept of Node.Js With Php

Riya Jain[1]
Student, Computer Engineering
riyasoni393@gmail.com
Shrinathji Institute of Technology & Engg, Nathdwara, India

Ankit Sen[2]
Student, Computer Engineering
senankit19@gmail.com
Shrinathji Institute of Technology & Engg, Nathdwara, India

Komal Paliwal[3]
Asst. Prof. Computer Engineering
engineerkomal_1987@yahoo.com
Shrinathji Institute of Technology & Engg, Nathdwara, India

***Abstract*—The current environment of web applications demands performance and scalability. Several previous approaches have implemented threading, events, or both, but increasing traffic requires new solutions for improved concurrent service. Node.js is a new web framework that achieves both through server-side JavaScript and event-driven I/O. Tests will be performed against two comparable frameworks that compare service request times over a number of cores. The results will demonstrate the performance of JavaScript as a server-side language and the efficiency of the non-blocking asynchronous model. The paper discuss that node.JS is a suitable framework for development of scalable web servers, can be scaled and distributed across multiple nodes using clustering and replication mechanism.**

Keywords—node.js, JavaScript, performance, scalable, function

I. INTRODUCTION

In this era of computer and internet, dynamic web is the ubiquitous source for socializing; news updates to name a few amongst many other uncountable usages [1].The Internet is continually evolving and presents a number challenges to web application designers [1].High traffic demands services to better handle concurrent sessions [2]. With the advent of social net- working sites, huge amount of people have been visiting and spending time simultaneously on such sites. This poses the requirement that sites need to be very quick in performance, scalable and distributed despite of large number of concurrent users. Currently, we can take facebook and google+ as some of the examples of popular social sites to get the idea of how they need to be scalable without any degradation of performance to support large number of concurrent users and to keep the users interested on visiting sites more often. So, to avoid the user frustration on using web application requires smoothness of access and support of any number of simultaneous users without performance degradation [6]. In this paper, test driven development of a web application has been carried out using relatively new programming language 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 easy to use as an expressive, functional language for server-side programming that's popular among developers [7]. Node utilizes the JavaScript V8 engine, developed by Google [9], a fast and powerful implementation of JavaScript [8] that helps Node achieve top performance. In the following experiments, Node will be compared to Apache to evaluate practical web frameworks for the challenges posed by the current web. JavaScript layer is allowed to access the main thread only whereas C layer is open for multithreading [5]. Multithreaded programming approach is easy and efficient way to make use of multiple cores for concurrency but it has some pitfalls like deadlocks, accessibility of shared resources and frequent switching of processes [6]. Whereas Event driven model of node.JS provides more scalable solution of switching between tasks making use of event notification functionalities of underlying OS like select(), poll() along with epoll, kqueue, kevent calls [10].

II.LITERATURE SURVEY

A literature survey was done throughout this thesis work. Scientific and research papers were studied, however since node.js is yet a neoteric technology there were certain papers available on the topic. Therefore, online documentation and blog posts were also deliberated.

In recently published article on IEEE author Tilkov, Stefan summarizes that node is one of the better-known frameworks and environments that support server-side JavaScript devel-opment.
Similarly, the author of [Til10] summarizes that node.js can be used to build high performance network programs due to its evented asynchronous style of programming. It bridges the gap between JavaScript being used only on the client side and dependency of other platforms in server side by making a single platform usable in both environments.
Leavitt N. in his publication in IEEE as "Will NoSQL Databases Live Up to Their Promise?" based on interviews of some database experts, cites nosql databases as scalable and good performance data storage solution for applications

where data precision and consistency are not major concerns (like bank transactions) compared to its scalability and performance.

III.APPROACH & TECHNIQUES USED

Node.js uses an event-driven model where the web server accepts the request, spins it off to be handled, and then goes on to service the next web request. When the original request is completed, it gets back in the processing queue and when it reaches the front of the queue the results are sent back (or whatever the next action is). This model is highly efficient and scalable because the web server is basically always accepting requests because it's not waiting for any read or writes operations. (This is referred to as "non-blocking I/O" or "event-driven I/O".)

You use your web browser to make a request for "/about.html" on a Node.js web server. The Node server accepts your request and calls a function to retrieve that file from disk. While the Node server is waiting for the file to be retrieved, it services the next web request. When the file is retrieved, there is a callback function that is inserted in the Node servers queue. The Node server executes that function which in this case would render the "/about.html" page and send it back to your web browser. Now, sure, in this case, it may only take microseconds for the server to retrieve the file, but Microseconds matter!

Particularly when you are talking about highly-scalable web servers! This is what makes Node.js different and of such interest right now. Add in the fact that it also uses the very common language of JavaScript, and it is a very easy way for developers to create very fast and very scalable servers.

TABLE I. A simple HTTP file server. Events trigger anonymous functions that execute input or output operations.

```
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/");
```

IV.PERFORMANCE BENCHMARK

This section focuses on the results obtained after benchmarking the node.js against PHP with some very simple tasks. These benchmarks are obtained as a result of load testing using a highly distributed, concurrent benchmarking testing tool.I have done few very basic performance tests to see how node.js server behaves compared to Apache when serving very simple pages. For comparison I have used node.js 0.1.103 on one side, and Apache 2.2.14 with prefork MPM and PHP 5.2.10 on the other side. I am discussing all the tests one by one following are test:-

Test 1

I have used node.js 0.1.103 on one side, and Apache 2.2.14 with prefork MPM and PHP 5.2.10 on the other, hitting them with Apache Bench 2.3 and total of 100,000 requests with 1,000 concurrent requests during first test:

`ab -r -n 100000 -c 1000` http://127.0.0.1:8000/

And then with total of 1,000,000 requests and 20,000 concurrent requests during the second one:

`ab -r -n 1000000 -c 20000` http://127.0.0.1:8000/

Total Request: 100,000; Concurrency Level: 1,000

Node. Js Result

TABLE II. Node.js vs. PHP Code.

Node.js Code	**PHP Code**

var sys = require('sys'),
http = require('http');
var body='Hello World'
http.createServer(**function**(req, res) {
res.writeHead(200, {'Content-Type'**:**
'text/html'});
res.end();
}).listen(8000);
sys.puts('Server running at
http://127.0.1.8000/')
	<?php
	echo '<p>Hello
	World</p>';

To illustrate CPU and memory load changes I have measured them during tests using dstat (first for node.js, and then Apache right afterwards), with following results:

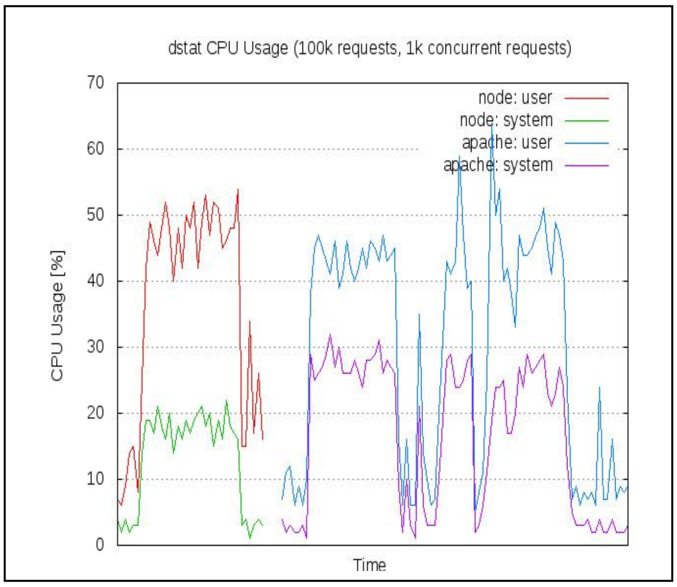

Fig 1. . CPU Usage: Node.js vs. Apache/PHP in Apache Bench test - 100k requests, 1k concurrent requests

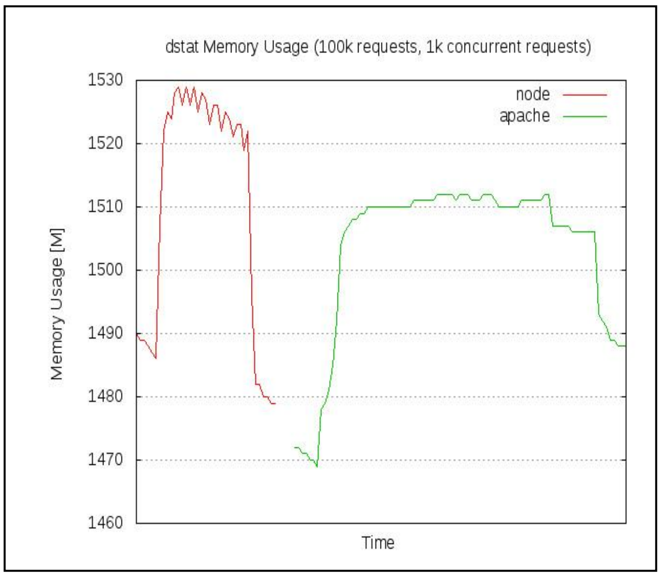

Fig 2.. Memory Usage: Node.js vs. Apache/PHP in Apache Bench test - 100k requests, 1k concurrent requests

The test sent 1000 requests (same request, same params), with 50 concurrent requests.

The result is:

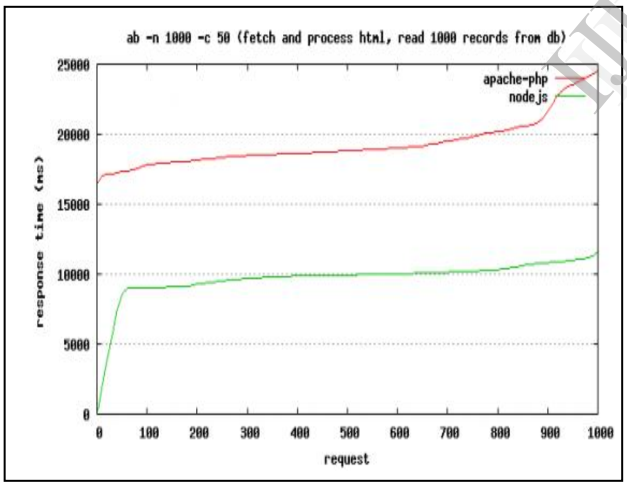

Fig 3. ab -r -n 1000 -c 50 http://127.0.0.1:8000/

As the above graph shows, Node.js performed better than PHP & Apache. The results were encouraging enough for me to make the switch to Node.js (for this particular script).The important result
is that Node.JS was around 80% faster than PHP & Apache. I used Apache Benchmark for this test (with Gnu plot to create the image). The results above will certainly vary depending on your system, the resources available, etc.

V.APPLICATION OF NODE IN THE ENTERPRISE

Voxer is using Node and Riak to build its walkie talkie-style mobile app (and have open sourced their in-houseRiak client).Yahoo has used Node to develop their Mojito platform and a handful of others.WalMart Labs has been using lots of Node in mobile development.
I have also seen all kinds of promising game development done using tools like Node, socket.io, and Redis. I'm excited to see how far this goes and whether or not large gaming companies like Zynga will come to favor Node over Action Script, Erlang, and other languages and frameworks.
AppFog has used Node extensively in our console and in other crucial parts of our application architecture after various flirtations with EventMachine turned out disastrously for us in testing. We're a new company, but we're making an investment in Node and we are not alone.
Even Oracle has gotten involved in the Node community. Last fall, they announced that they planned to develop Nashorn, a JVM-based JavaScript engine set to be available in late 2012. They've explicitly talked up the ability to use Nashorn to do crazy things like create Node.jar files and more generally to make the power of Node more readily Java-compatible.
So what does that mean for the future of Node? Well, first of all, I think it will mean that the demographics of the Node community will change. Thus far, your typical Node developer is still young, a resident of a small handful of urban areas in the U.S., and not very "corporate."
That profile is changing already. Older hands are getting involved as Node loses its reputation as being intrinsically immature and unstable. Second, I think there's a possibility that the biggest qualitative leaps in Node development might someday come from the enterprise rather than from the vast ecosystem of autonomous developers. This is an uncomfortable thought for plenty of Node hackers, but if we see more of what we've seen so far from Yahoo, Voxer, and others is any indication, this could well turn out to be the case. There will always be thousands of developers contributing modules to NPM and providing incremental improvements and additions to Node, but big leaps often require larger projects demanding whole teams of developers. Big shake-ups in the database space, for example, have often followed on the heels of working papers publicized by Google (like this one on its Spanner architecture, which could also have a huge impact outside Google). The Node community could come to function in a similar fashion.
In short: Node is already in the enterprise, and it's set to expand there. But that doesn't mean that you have to delete your Node-related GitHub repos and tear down your Ryan Dahl poster. It's simply the run of things in tech. There will still be plenty of Nodes hacking to be done for the foreseeable future. But a lot of the most interesting stuff might be at a Wal-Mart or an Oracle.

VI.CONCLUSION & FUTURE WORK

Node accomplishes its goals of providing highly-scalable servers. It uses an extremely fast JavaScript engine from Google, the V8 engine. It uses an Event-Driven design to keep code minimal and easy-to-read. All of these factors lead to Node's desired goal it's relatively easy to write a massively-scalable solution.

Just as important as understanding what Node is, it's also important to understand what it is not? Node is not simply a replacement for Apache that will instantly make your PHP web application more scalable. That couldn't be further from the truth. It's still early in Node's life, but it's growing extremely quickly, the community is very actively involved, there are a lot of good modules being created, and this growing product could be in your business within a year.

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 extenstion.

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
[8] Paruj Ratanaworabhan, Benjamin Livshits, and Benjamin Zorn. JSMeter: Comparing the behavior of JavaScript benchmarks with real web applications. In USENIX Conference on Web ApplicationDevelopment (WebApps), June 2010.
[9] Google Javascript V8, http://code.google
[10] http://teddziuba.com/2011/10/node-js-is-cancer.html accessed10/26/11
[11] http://hns.github.com/2010/09/21/benchmark.html accessed 11/18/11
[12] 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.
[13] 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.
[14] Sun Microsystems. RPC: Remote Procedure Call Protocol Specication Version 2. Internet Network Working Group RFC1057, June 1988.