Dennis Durairaj

Compare Web Technologies

2013-09-01

Abstract: In the modern world, web applications play an important role for many industries. Large scale and high concurrency are important factors for the new era of web applications. The Web technologies involved in developing these applications play an important role. Many technologies such as PHP, ASP.NET, Node, Python, Ruby etc are available in the market for companies to achieve this goal. PHP has been in the web development industry for many years and a large number of web applications still run on PHP while Node has gained popularity in the last few years. Node has become a popular technology to build data-intensive web applications. To study and analyse the performance of Node and PHP, we use benchmark and scenario tests. The results obtained have some valuable performance data, showing that PHP handles much less requests than Node in a certain time. Results show that Node is lightweight and efficient, which is an idea fit for I/O intensive websites between the two, while PHP is relatively more suitable for small and middle scale applications.

1. Introduction

An application server is a framework that provides developers with facilities to create web applications and a server on which we can run them. Application Server Frameworks contain a detailed service layer model where the application server acts as a group of components accessible to the code developer through a regular API outlined for the platform itself. For web applications, these parts are sometimes performed within the same running environment as their web server(s), and their main job is to support the development of dynamic pages. However, several application servers target way more than simply web page generation: they implement services like clustering, fail-over, and load-balancing, thus developers can concentrate on implementing the business logic. Application server also refer to the computer hardware on which the services run.

Application servers are platforms where web applications or desktop applications run. Application servers comprise of web server connectors, PC programming languages, runtime libraries, database connectors, and the organization code expected to be deployed, design, oversee, and connect these parts on a web host. An application server keeps running behind a web server (e.g. Apache or Microsoft Internet Information Services (IIS)) and quite often before a SQL database (e.g. PostgreSQL, MySQL, or Oracle). Web applications are codes which keep running on application servers and are composed in the language(s) the application server supports and call the runtime libraries and parts the application server offers.

Numerous application servers exist. The decision of choosing a server architecture impacts the cost, execution, reliability, adaptability, and viability of a web application. Exclusive application servers give system benefits in an all-around characterized yet proprietary manner. The application engineers create programs as per the specification of the application server. This project aims to compare the performance of a selected set of web server technologies and analyse the results based on various circumstances and conclude with which web server architecture should be used under what situations. The benchmarking of these various web technologies will be done using two popular benchmarking tools called "LoadRunner" and "JMeter". The project also aims to compare the two benchmarking tools, analysing their features and results.

1.1. Purpose of the project

In the rapid development of Web today, many sites are faced with new problems, such as the problem of multiuser requests and high concurrency. The dynamic scripting language JavaScript has become enormously popular for client and is widely used in Web development. Node stands for one new technology in JavaScript. Node 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 177,352 repositories and growing [3]. And talking about evaluation of Web technologiesâĂŹ performance, many researchers have done the related work. But the work described in this thesis differs from others in two aspects. Firstly, we consider from both objective systematic tests (benchmark) and realistic user behaviour tests (scenario).

Not long ago even a 2-second page response time was considered as an acceptable one. However, web users have become increasingly impatient when it comes to speed these days. Earlier, speed was considered a feature and now it is deemed a necessity. Additionally, technological innovation in mobile space has raised the bar for speed. Hence, speed makes a lot of economic sense now. A recent research found that 250-450 milliseconds are the magical numbers that decide the winner in the race of web speed []. Research also indicates that the slower the site, the lesser would be the number of clicks and transactions performed on the site which would eventually result in the loss of users.

In order to perform these tests on the various Web technologies we will make use of the testing tools such as JMeter and LoadRunner. In addition to testing the various Web technologies, this project will also focus on the quality of the testing tools, the differences between them and if they produce different results under certain circumstances. Complex systems make increasing demands on web servers, multiple objects can interfere when one process is handling a request for a specific user. High volumes can overwhelm systems if they are not scaled correctly. Fixes need to be identified early in the project so that server crashes and vulnerabilities can be caught in advance Clients have scalability concerns and we must warranty some level of scalability with industry accepted metrics. In order to achieve this, our web servers.

This paper focuses on the impact on Web performance from two different Web technologies: Node and PHP. The security and scalability issues are beyond the scope of the thesis. We mainly use the benchmark tests and scenario tests. In addition, one universal method of Web development techniqueâĂŹs 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 in our experiment and analyse the results of it. Then we compare it with PHP 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 behaviour tests (scenario). There is often a dual impact on Web server performance, from

the calculation, and from the number of users. The research herein has taken each of these effects in account.

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

2. Technologies and Toolset

2.1. Node

The official website (http://www.nodejs.org) defines Node as "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]."

Regardless, JavaScript is the world's most famous programming languages. If you have done any programming for the web, it's unavoidable. JavaScript, in view of the sheer reach of the web, has satisfied the "compose once, run anywhere" dream that Java had back in the 1990s. Around the season of the Ajax insurgency in 2005, JavaScript went from being a "toy" language to something individuals wrote genuine and noteworthy projects with. A portion of the eminent firsts were Google Maps and Gmail, yet today there are a large group of web applications from Twitter to Facebook to GitHub.

JavaScript has for some time been the true standard for frontend side web development. While about all frontend code is composed in JavaScript, server-side development is a variety of choices between PHP, Java, and various different technologies. Life as a web engineer would be substantially more straightforward if a single language was utilized all around. Since JavaScript overwhelms in the browser, it bodes well to utilize it on the server too.

The idea of server-side JavaScript is not a new one. Netscape initially introduced JavaScript into the server world in 1994. Since that time, a lot of projects have endeavoured, and failed, to advance JavaScript as a server-side language. Execution, or scarcity thereof, restricted JavaScript from picking up a genuine a dependable balance in the server space.

Throughout the years, JavaScript has seen gigantic upgrades in performance. Because of its pertinence in the program, enormous players like Google have contributed a considerable measure of time and cash to make JavaScript as fast as possible. In 2009, Ryan Dahl of Joyent, put the large part of that recently discovered execution to great use on the server when he made the Node.js structure. Dahl assembled Node.js over Google's V8 JavaScript engine. V8 is a similar engine that has given Google Chrome its astounding JavaScript performance, and helped it turn into the most well-known browser on the planet.

2.1.1. Technical Details

— **Threading**

Node.js works in a single threaded environment, utilizing non-blocking I/O calls, enabling it to support a huge number of simultaneous connections without incurring the cost of thread context switching [17]. The idea of sharing a single thread among every one of the requests that utilization the observer design is for building exceptionally concurrent applications, where any function performing I/O must utilize a callback. With a specific end goal to accommodate the single-threaded event loop, Node.js uses the libuv library that, utilizes a fixed-sized thread pool that is in charge of a portion of the non-blocking async I/O operations [18].

A drawback of this single-threaded approach is that Node.js does not permit vertical scaling by increasing the amount of CPU cores of the machine it's running on while not using a further module, like cluster, StrongLoop process Manager, or pm2. However, developers can increase the default range of threads within the libuv thread pool; these threads are probably to be distributed across multiple cores by the server software system [19].

Execution of concurrent tasks in Node.js is handled by a thread pool. the main thread decision functions post tasks to the shared task queue that threads within the thread pool pull and execute. Inherently non-blocking system functions like networking interprets to kernel-side non-blocking sockets, whereas inherently blocking system functions like file I/O run in an exceedingly block method on its own thread When a thread in the thread pool completes a task, it informs the main thread of this, which wakes up and execute the callback. As callbacks are handled synchronously on the main thread, long lasting computations and other CPU-bound tasks will freeze the whole event-loop till completion.

— **V8 Engine**

V8 is the JavaScript execution engine built for Google Chrome and open-sourced by Google in 2008. Written in C++, V8 compiles JavaScript source code to native machine code instead of interpreting it in real time [18].

Node.js makes use of libuv to handle asynchronous events. Libuv is an abstraction layer file system and network functionality on each Windows and POSIX-based systems like UNIX operating system, macOS, OSS on NonStop, and Unix.

The core functionality of Node.js resides in a JavaScript library. The Node.js bindings, written in C++, connect these technologies to each other and to the operating system.

— **Package Management**

npm is an in-built package manager for the Node.js servers. It is utilized to install Node.js programs from the npm registry, sorting out the installation and

administration of third-party Node.js programs. npm isn't to be mistaken for the CommonJS âĂŸrequire()âĂŹ statement. It is not utilized to load code; rather, it is utilized to install code and manage dependencies from the command line. The bundles found in the npm registry can extend from basic helper libraries, for example, Lodash to task runners suck as Gulp.

— **Unified API**

Node.js may be combined with a browser, a database supporting JSON information (such as Postgres, MongoDB, or CouchDB) and JSON for a unified JavaScript development stack. With the difference of what were basically server-side development patterns like MVC, MVP, MVVM, etc., Node.js permits the reuse of the same model and service interface between client-side and server-side.

— **Event Loop**

Node.js registers itself with the software system so as to be notified once a connection is created, and also the OS can issue a callback. inside the Node.js runtime, every connection could be a small heap allocation. historically, comparatively heavyweight OS processes or threads handled every connection. Node.js uses an event loop for scalability, rather than processes or threads [74]. In distinction to different event-driven servers, Node.js's event loop doesn't need to be known as explicitly. Instead callbacks are outlined, and also the server mechanically enters the event loop at the end of the callback definition. Node.js exits the event loop once there are not any any callbacks to be performed.

— **Non-blocking**

The question of whether an operation is blocking or non-blocking refers to the fact that it must finish before the next operation begins. Non-blocking operations are said to be asynchronous and blocking operations are said to be synchronous. Node is non-blocking i.e. operations don't have to happen consecutively.

— **Scalability**

Node.js utilizes a single threaded model with event callbacks. Events encourage the server to react in a non-blocking way and makes the server scalable compared to traditional servers which give restricted access to threads to deal with requests. Node utilizes a single threaded program to provide services to a substantially bigger number of requests than traditional servers like Apache HTTP Server.

— **Memory Management**

Since Node is single-threaded, that implies that every one of your users will be sharing the same memory allocation. At the end of the day, unlike to in the browser, you must be mindful so as not to store user-specific information in closures where different connections can affect it.

2.1.2. Middleware

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. Express provides a thin layer of fundamental web application features, without obscuring Node.js features [17].

Express is the most prominent framework for Node applications, and it highlights middleware utilizing continuation passing. When you need to run a similar code for conceivably a wide range of routes, the perfect place for that code is likely middleware. Middleware is a function that gets passed the request and response objects, alongside a continuation function to call, called next(). Envision that you need to add a requestId to each request/response pair with the goal that you can follow them back to the individual request when you're troubleshooting or debugging your logs for something.

You can write some middleware like this:

```
require('dotenv').config();
const express = require('express');
const cuid = require('cuid');

const app = express();

// request id middleware
const requestId = (req, res, next) => {
  const requestId = cuid();
  req.id = requestId;
  res.id = requestId;

  // pass continuation to next middleware
  next();
};

app.use(requestId);

app.get('/', (req, res) => {
  res.send('\n\nHello, world!\n\n');
});

module.exports = app;
```

2.2. PHP

PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML [18].

Instead of lots of commands to output HTML (as seen in C or Perl), PHP pages contain HTML with embedded code that does "something" (in this case, output "Hi, I'm a PHP script!"). The PHP code is enclosed in special start and end processing instructions <?php and ?> that allow you to jump into and out of "PHP mode."

What distinguishes PHP from something like client-side JavaScript is that the code is executed on the server, generating HTML which is then sent to the client. The client would receive the results of running that script, but would not know what the underlying code was. You can even configure your web server to process all your HTML files with PHP, and then there's really no way that users can tell what you have up your sleeve [18].

PHP began as a small open source venture that advanced as an ever-increasing number of people discovered how valuable it was. Rasmus Lerdorf released the primary rendition of PHP back in 1994.