Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Informatyki

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria Systemów Informatycznych

Tytuł Pracy

Dennis Durairaj
Numer albumu 281093

promotor
<tu wpisz tytuł> Julian Myrcha

WARSZAWA 2018

Streszczenie

Tytuł pracy: <tu wpisz tytuł po polsku>

Słowa kluczowe:
<tu wpisz słowa kluczowe>

(podpis opiekuna naukowego) (podpis dyplomanta)

Abstract

Title of the thesis: <tu wpisz tytuł po angielsku>

Słowa kluczowe:
<tu wpisz słowa kluczowe>

(podpis opiekuna naukowego) (podpis dyplomanta)

Oświadczenie o samodzielności wykonania pracy

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych

Ja niżej podpisany/a:

<tu wpisz imię i nazwisko> , <tu wpisz nr indeksu>

student/ka Wydziału Elektroniki i Technik Informacyjnych Politechniki Warszawskiej, świadom/a odpowiedzialności prawnej przedłożoną do obrony pracę dyplomową inżynierską pt.:

<tu wpisz tytuł>

wykonałem/am samodzielnie pod kierunkiem

<tu wpisz tytuł, imię i nazwisko promotora>

Jednocześnie oświadczam, że:

- praca nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 o prawie autorskim i prawach pokrewnych, oraz dóbr osobistych chronionych prawem cywilnym,
- praca nie zawiera danych i informacji uzyskanych w sposób niezgodny z obowiązującymi przepisami,
- praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem dyplomu lub tytułu zawodowego w wyższej uczelni,
- promotor pracy jest jej współtwórcą w rozumieniu ustawy z dnia 4 lutego 1994 o prawie autorskim i prawach pokrewnych.

Oświadczam także, że treść pracy zapisanej na przekazanym nośniku elektronicznym jest zgodna z treścią zawartą w wydrukowanej wersji niniejszej pracy dyplomowej.

Warszawa, dnia (podpis dyplomanta)

Oświadczenie o udzieleniu Uczelni licencji do pracy

Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych

Ja niżej podpisany/a:

<tu wpisz imię i nazwisko> , <tu wpisz nr indeksu>

student/ka Wydziału Elektroniki i Technik Informacyjnych Politechniki Warszawskiej, niniejszym oświadczam, że zachowując moje prawa autorskie udzielam Politechnice Warszawskiej nieograniczonej w czasie, nieodpłatnej licencji wyłącznej do korzystania z przedstawionej dokumentacji pracy dyplomowej pt.:

<tu wpisz tytuł>

w zakresie jej publicznego udostępniania i rozpowszechniania w wersji drukowanej i elektronicznej*.

Warszawa, dnia (podpis dyplomanta)

*Na podstawie Ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (Dz.U. 2005 nr 164 poz. 1365) Art. 239. oraz Ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz.U. z 2000 r. Nr 80, poz. 904, z późn. zm.) Art. 15a. "Uczelni w rozumieniu przepisów o szkolnictwie wyższym przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli uczelnia nie opublikowała pracy dyplomowej w ciągu 6 miesięcy od jej obrony, student, który ją przygotował, może ją opublikować, chyba że praca dyplomowa jest częścią utworu zbiorowego."

Contents

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 servers are platforms where web applications or desktop applications run. Application server also refer to the computer hardware on which the services 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 "ApacheBench" and "JMeter". The project also aims to compare the two benchmarking tools, analysing their features and results.

1. INTRODUCTION

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 [2]. 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 [2]. 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 [1]. Nowadays, JavaScript has been the first popular language in GitHub with 177,352 repositories and growing [3].

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 numbers that decide the winner in the race for 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 tool ApacheBench. Complex systems make increasing demands on web servers, high volumes can overwhelm systems if they are not scaled correctly. Multiple objects can interfere when one process is handling a request for a specific user. 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 three different Web technologies: Node, PHP and Python-Django. The security and scalability issues are beyond the scope of the thesis. We mainly use the benchmark tests to analyse the web technologies. The main contributions of this thesis are listed as follows.

1. We consider web technologies such as Node, PHP and Django in our experiment and analyse their performance based on certain performance tests. Then we compare them making a conclusion of which situation they ought to be used in and in which situation they outperform the other.

2. By means of benchmark tests we evaluate performance from an objective point of view. There is often a dual impact on Web server performance, from the CPU intensive requests, and from the number of users making requests. The research herein has taken each of these effects in account.

1.2 Goals of the project

The goal of this project is to determine under what circumstances do different web technologies perform better than the other and based upon the results making a conclusion in what situations they should be used. With the number of different web technologies available today, it becomes difficult to make a choice on what web technology should be used for a specific application. Whether it is a static web application, dynamic web application, web applications with content management systems such as WordPress and Drupal, social networking application, or CPU intensive web applications such as file compression, audio/video trans-coding. This project aims to determine if the different web technologies perform differently when used for some of the above purposes and find the best one for each case.

The rest of this research is organized as follows - Chapter 2 discusses the technologies and tools used in this project. Chapter 3 describes the test bed and environment configurations. Chapter 4 details the methodology and experimental philosophy of tests. Chapter 5 presents and analyses the results of all tests. Chapter 6 makes a conclusion of the paper with a summary of study and a future direction.

1.3 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 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 . 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].

2 Technologies and Toolset

2.1 Node

The official website (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.

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 [13]. 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 [14].

  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 [15].

  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 [14].

  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 'require()' 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 [1]. 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.

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 [16].

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 [17].

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 [17].

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.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor"
- PHP is a server-side scripting language that is embedded with HTML. It is utilized to manage dynamic content, session tracking, databases even build whole web based e-commerce websites.
- It is coordinated with various well-known databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly fast in its execution, particularly when compiled as an Apache module on the Unix side. The MySQL server, once running, executes even extremely complex queries with tremendous results returned in record-setting time.
- PHP supports countless protocols, for example, POP3, IMAP, and LDAP. PHP included support for Java and distributed object architectures (COM and CORBA), making n-level improvement a plausibility for the first time.
- PHP syntax is C-Like.

PHP is essentially centred around server-side scripting, so you can do anything some other CGI program can do, for example, gather form information, produce dynamic page content, or send and get cookies. In any case, PHP can do significantly more.

There are three main areas where PHP scripts are used -

- Server-side scripting - This is the most traditional and fundamental target field for PHP. In order to make this work you require three things: the PHP parser (CGI or server module), a web server and a web browser. You have to run the web server, with an associated PHP installation. You can get to the PHP program output with a web browser, seeing the PHP page through the server. All these can keep running on your home machine if that you are simply experimenting with PHP.
- Command line scripting - You can make a PHP script to run with no server or program. You just need the PHP parser to utilize it in the appropriate way. This kind of use is perfect for scripts consistently executed utilizing cron (on *nix or Linux) or Task Scheduler (on Windows). These scripts can likewise be used for straightforward script processing tasks.
- Writing desktop applications - PHP is assumed to not be the absolute best language to develop a desktop application with a graphical user interface, yet if you know PHP well, ait is possible to utilize some advanced PHP libraries in the client-side applications to compose such programs. Additionally, you can build cross-platform applications along these lines. PHP-GTK is an expansion to PHP, not accessible in the fundamental distribution.

PHP can be used on all major operating systems, including Microsoft Windows, Mac OS X, Linux, many Unix variants, RISC OS, and probably others. PHP has support for almost all of the web servers today. This includes IIS, Apache, and many others [17]. And this consists of any internet server which could make use of the FastCGI PHP binary, like lighttpd and nginx. PHP works as both a module, or as a CGI processor.

So, with PHP, you have the liberty of selecting an operating system and a web server. Furthermore, you also have the choice of using procedural programming or object-orientated programming (OOP), or a combination of them both.

With PHP, you aren't confined to output HTML. PHP's power includes outputting pictures, PDF documents or even Flash movies (with the usage of libswf and Ming) generated at the fly. You can additionally output easily any text, together with XHTML and another XML document. PHP can autogenerate these documents, and save them within the system, as opposed to printing it out, forming a server-side cache to your dynamic content material [17].

One of the strongest features in PHP is its support for a huge range of databases. Writing a database-enabled application is pretty easy with the help of one of the database specific languages (e.g. mysql), or using an abstraction layer like PDO, or connect to any database that supports the Open Database Connection standard popular via the ODBC extension. Other databases may additionally utilize cURL or sockets, like CouchDB [17].

PHP also has support for communicating with other services using protocols inclusive of IMAP, HTTP, LDAP, POP3, SNMP, NNTP, COM (on windows) and endless others. You may also open raw network connections and interact using every other protocol. PHP has aid for the WDDX complex information exchange among truly all web programming languages.

PHP has a powerful text processing feature, which incorporates the Perl compatible regular expressions (PCRE), and numerous expansions and functions to parse and get XML records. PHP standardises a large part of the XML augmentations on the strong base of libxml2, and expands the list of capabilities by including SimpleXML, XMLReader and XMLWriter support.

2.3 Python - Django

Django is a high level Python Web Framework enabling developers to write clean and pragmatic web applications. Django supports templating, routing, authentication, routing, basic database administation and many other features. Although Django can be used without a database, it comes packed with an object-relational mapper which describes the database layout in simple Python code. Models can be represented in many different ways using the data-model syntax. Django also has a clean and elegant URL scheme and does not put any ambiguities in URLS.

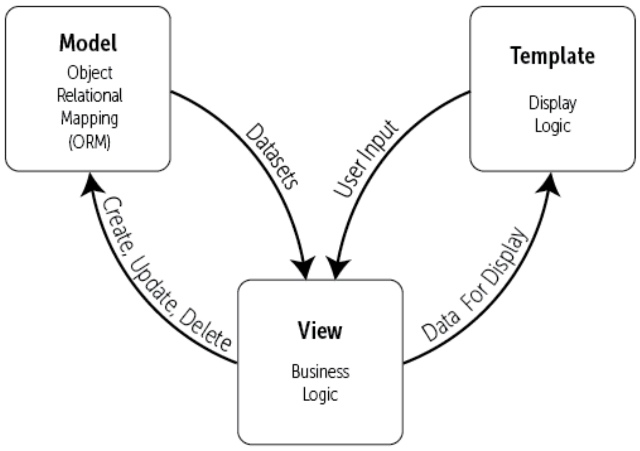
Django is written in the Python programming language. Python is arguably one of the easiest programming languages to read and understand. It uses natural language constructs like paragraph-like layout and indentation along with easy to learn syntax. It makes understanding the structure of the program and flow significantly easy compared to some other programming languages.

Django follows the Model-View-Controller approach. MVC is a design pattern aiming to separate web application into mainly three parts -

- **Model** - The model provides an interface between the database and containing application data. Django's Object-Relational Mapping provides the interface to the application database.

- **View/Template** - The view decides the information that should be presented to the user and also collects the information from the user. It acts as the interface between the client and the server. The template provides display logic and is the interface between the user and your Django application

- **Controller/View** - The controller manages the business logic for the application by acting as an information broker between the view and the model. The view manages the bulk of the applications data processing, application logic and messaging.



This design philosophy allows developers to -

- Run servers separately for the database, media and applications.
- Easily have media served from a CDN (Content Delivery Network)
- Cache content at multiple scopes and levels
- For large web applications, it enables clustering and load-balancing in order to distribute the website across multiple servers

Web programming and web design are two very different ideas. For everything but the small prkects, the programming and design is not done by the same person, in many situations it is not even done by the same company. Django's template system makes it clear that Django programmers and websites must be able to work independent from each other. The profit is a plain text scripting language which makes use of tags to provide the presentation logic that decides what content needs to be displayed in the template.

DRY (Don't Repeat Yourself) is a term that often comes up, and it is one of Django's core principles. The DRY principle is evident in how Django makes use of template inheritance and helps reduce repetition and redundant code. Django comes with a large set of in-built tools that can be used out of the box without prior setup requirement. It provides some common, however complex processes in the form of wrappers. Some of these packages include administration, authorization, database specific features, session management, syndication, etc. Django provides high security by disallowing code execution inside a webpage. This simple approach provides high security compared to other languages such as Javascript which can be executed within the browser and hence keeps the door open to various kind of vulnerabilities.

2.4 .NET

.NET Core is a general purpose, modular, cross-platform and open source implementation of the .NET Standard. It contains many of the same APIs as the .NET Framework (but .NET Core is a smaller set) and includes runtime, framework, compiler and tools components that support a variety of operating systems and chip targets. The .NET Core implementation was primarily driven by the ASP.NET Core workloads but also by the need and desire to have a more modern implementation. It can be used in device, cloud and embedded/IoT scenarios. [18].

.NET Core is supported by Microsoft on Windows, macOS and Linux. On Linux, Microsoft primarily supports .NET Core running on Red Hat Enterprise Linux (RHEL) and Debian distribution families. .NET Core currently supports X64 CPUs. On Windows, X86 is also supported. ARM64 and ARM32 are in progress.

Here are the main characteristics of .NET Core:

- **Cross-platform** - .NET Core provides key functionality to implement the app features you need and reuse this code regardless of your platform target. It currently supports three main operating systems (OS): Windows, Linux and macOS. You can write apps and libraries that run unmodified across supported operating systems. To see the list of supported operating systems, visit .NET Core roadmap.

- **Open source** - .NET Core is one of the many projects under the stewardship of the .NET Foundation and is available on GitHub. Having .NET Core as an open source project promotes a more transparent development process and promotes an active and engaged community.

- **Flexible deployment** - There are two main ways to deploy your app: framework-dependent deployment or self-contained deployment. With framework-dependent deployment, only your app and third-party dependencies are installed and your app depends on a system-wide version of .NET Core to be present. With self-contained deployment, the .NET Core version used to build your application is also deployed along with your app and third-party dependencies and can run side-by-side with other versions. For more information, see .NET Core Application Deployment [18].

- **Modular** - .NET Core is modular because it's released through NuGet in smaller assembly packages. Rather than one large assembly that contains most of the core functionality, .NET Core is made available as smaller feature-centric packages. This enables a more agile development model for us and allows you to optimize your app to include just the NuGet packages you need. The benefits of a smaller app surface area include tighter security, reduced servicing, improved performance, and decreased costs in a pay-for-what-you-use model [18].

.NET Core is composed of the following parts:

- A .NET runtime, which provides a type system, assembly loading, a garbage collector, native interop and other basic services.
- A set of framework libraries, which provide primitive data types, app composition types and fundamental utilities.
- A set of SDK tools and language compilers that enable the base developer experience, available in the .NET Core SDK.
- The 'dotnet' app host, which is used to launch .NET Core apps. It selects the runtime and hosts the runtime, provides an assembly loading policy and launches the app. The same host is also used to launch SDK tools in much the same way.

2.5 Database - PostgreSQL

PostgreSQL is an object-relational database management system that emphasizes on the principles of extensibility and standards compliance. The primary function is to store data in a secure manner and return the data as response when requests are made to the database from software applications. It is able to handle high workloads that range from small single-machine applications to very large-scale applications for purposes such as data warehousing that have many concurrent users. PostgreSQL requires mininum maintenance because it is very stable. Therefore applications developed on PostgreSQL have low total cost of ownership compared to other database management systems.

Some of the important features of PostgreSQL are as follows -

- User-defined types
- Table inheritance
- Sophisticated locking mechanism
- Views, rules, subquery
- Nested transactions

- Asynchronous replication
- Foreign key referential integrity
- Multi version concurrency control

PostgreSQL is standards compliant. Its implementation of SQL strongly conforms to ANSI-SQL:2008 standard. It supports all subqueries including subselects, it is read-committed and serializes transaction isolation levels. The data integrity features included in postgreSQL are primary keys, foreign keys with cascading updates and deletes as well as restrictions, check constraints, not null constraints and unique constraints. Some of the advantages of using PostgreSQL are -

- **Immunity to over-deployment** Over-deployment is a major problem in most proprietary database vendors such as Oracle SQL. But with PostgreSQL, since it has no associated licensing cost for using the software, there are no legal restrictions or compliances to follow.
- **Extensible** The source code for PostgreSQL is open-source. If a database administrator needs to customise or extend PostgreSQL, they are free to do so and require minimum effort with no added costs.
- **Cross platform** PostgreSQL is available on all the major UNIX operating systems as well as Windows and macOS via the cygwin framework.
- **Low staffing and maintenance cost** PostgreSQL is designed to have minimum maintenance and tuning requirements as compared to other proprietary databases, while providing all the features, performance and stability.

2.6 ApacheBench

ApacheBench (ab) is a single-threaded command line computer program for measuring the performance of HTTP web servers. Originally designed to test the Apache HTTP Server, it is generic enough to test any web server. The ab tool comes bundled with the standard Apache source distribution, and like the Apache web server itself, is free, open source software and distributed under the terms of the Apache License. [19] Apachebench generates a flood of queries to the specified URL and returns a variety of metrics. These metrics can be used to compare the performance of various web technologies in our project. Some of the important metrics are requests/second, time per request, and transfer rate. The tool allows testers to load test a URL or server by replicating number of users and number of concurrent requests per second.

2.7 JMeter

JMeter is an Open Source testing software. It is 100% pure Java application for load and performance testing. jMeter is designed to cover categories of tests like load, functional, performance, regression, etc., and it requires JDK 5 or higher. This tutorial will give you great understanding on jMeter framework needed to test an enterprise level application to deliver it with robustness and reliability. [20]

JMeter simulates a group of users sending requests to a target server, and returns statistics that show the performance/functionality of the target server/application via tables, graphs, etc.

The protocols supported by JMeter are -

- **Web** - HTTP, HTTPS sites 'web 1.0' web 2.0 (ajax, flex and flex-ws-amf)
- **Web Services** - SOAP / XML-RPC
- **Directory** - LDAP
- **Service** - POP3, IMAP, SMTP
- Database via JDBC drivers
- Messaging Oriented service via JMS
- FTP Service

Features

- Being an open source software, it is freely available.
- It has a simple and intuitive GUI.
- JMeter can conduct load and performance test for many different server types: Web - HTTP, HTTPS, SOAP, Database via JDBC, LDAP, JMS, Mail - POP3, etc.
- It is a platform-independent tool. On Linux/Unix, JMeter can be invoked by clicking on JMeter shell script. On Windows, it can be invoked by starting the jmeter.bat file.
- It has full Swing and lightweight component support (precompiled JAR uses packages javax.swing.*).
- JMeter store its test plans in XML format. This means you can generate a test plan using a text editor.
- Its full multi-threading framework allows concurrent sampling by many threads and simultaneous sampling of different functions by separate thread groups.
- It is highly extensible.
- It can also be used to perform automated and functional testing of the applications.

2.8 DigitalOcean

DigitalOcean is an IaaS (Infrastructure as a Service) company that delivers fast and easy solutions for developers and bsuinesses to deploy their applications on the cloud. It accelerates software development so that the developers and programmers can spend more time working on the feature building and less tiem and focus on managing the infrastructure.

DigitalOcean virtual servers (VPS) also known as "droplets" use KVM (Kernel-based Virtual Machine) as hypervisor and so they can be created in a variety of different sizes. DigitalOcean has their data center in eight different regions around the world, and provides 6 GNU/Linux distributions and dozens of on-click setup applications. DigitalOcean also offers an additional feature of load balancing on their cloud servers. They provide a scalable infrastructure for programmers to build applications with fast development times.

DigitalOcean's services are specifically created for all scales of applications, from individual based to enterprise based. They offer serveral different levels of cloud based hosting considering different requirements such as storage capacity, volume needs, and also billed either on a monthly or hourly basis. The more the features, will determite the amount of memory, disk space, transfer limit and core processors provided by the server in your chosen plan.

Setting up a DigitalOcean virtual server or "droplet" can be done in under a minute. With each droplet, the developer gets full root access to the server, with the ability to customize the server setup and choose the desired operating system.

All of DigitalsOcean's plans include the following -

- Solid state drives (SSD)
- Simple account control panel
- DNS management
- Global image transfer (gives the developers the ability to load droplets in different datacenters)
- Private networking (different droplets belonging to you can communicate with each other with no restrictions on bandwidth limits)

Technology and Infrastructure

DigitalOcean has a datacenter each in San Francisco, London, Singapore and three each in Amsterdam and New York. The datacenter in Singapore supports IPv6. Cloud servers are build using the KVM virtualization built on Intel's Hex-Core CPUs that have a dedicated ECC RAM and RAID SSD storage. However, the developer can use the control panel or DigitalOcean's name-spaced API to design their own. The control panel also provides a one-click install of common apps such as Linux distributions on the droplets, such as FreeBSD, CentOS, Ubuntu, Drupal or Wordpress. Security is one of the most important concerns when it comes to web hosting, but it becomes even more important on cloud hosting. DigitalOcean restricts access to the most critical systems, so the technical staff does not have any access to backend hypervisors, so only the engineering team can access snapshots of the storage systems as well as backups. The datacenters are protected physically by security staff that are working round-the-clock. Biometric readers and two factor authentication are also provided as a safety measure. DigitalOcean promises a 99.99% uptime, and offers credit if the downtime is below that level.

Features

- **Teams** - This features allows the developers to share resources and accounts between multiple teams and projects. By inviting teams members to streamline a lot of the manual work. The team members can have various roles such as some responsible for managing resources and billing across multiple sites or apps, provide an overview to make sure that all the team members are using the best practices for security, etc.

- **Block Storage** - Whenever more storage is required, it is easy to extend additional storage to the droplets as per our requirement. The storage blocks are separate from the actual server droplet and have multiple copies that are spreat out throughout the network of servers to make sure that the data is protected in rare situations of hardware failure.

- **Resilient Network** - By making use of Tier - 1 bandwidth, DigitalOcean's worldwide network is optimized so that it delivers data wherever it has to go, with high speed and high security. It is possible to instantly deploy by using load balancers which improve the reliability of the droplet server.

- **Spaces** - In addition to block storage, DigitalOcean provides a cloud storage solutions called Spaces. It is used as a store for important data.

- **One-click Installs** - When setting up a droplet, it is possible to imstall many different types of applications, scripts or CMS systems with just one click instead of logging in to the console and do it all from scratch.

3 Experimental Environment

3.1 Local Server Environment

The local servers are setup using the three different web technologies, namely Node, PHP and Django. Each of the servers are able to run simultaneously on the same machine on different ports. Below is the setup and configuration for each of the servers. The servers are setup using the http server module that comes bundled with each of the web technologies.

In order to keep the readings consistent, the local servers are setup on the same machine and are connected to the same network. The operating system on the local server machines is Windows 10, CPU speed 2.6 GHz and 16 GB RAM.

Node Server

In order to setup the Node server, the following pre-requisites are required -

- NPM - Node package manager
- Node
- Express framework

Node package manager gives us access to open-source packages that aid in the development of Node applications. It comes bundled with the installation of Node.

Once we choose a folder in which the server files will reside, we run the terminal and execute the following command - ‘npm install express‘

This installs the express middleware framework that provides us with the necessary modules to develop a Node server. After the installation completes we observe a folder structure as follows -

```
|-node_modules
|-src
|---index.html
|---index.js
|-package.json
```

The node_modules folder contains all the necessary packages that express requires, some of the notable packages are the ’http-server’, ’babel’ etc. Te src files contains all the server related files, index.html is the homepage of the server that is returned when a request is made to the server. index.js is where the server code lives. The simplest method to fire a web server using Node and express is shown below -

```
var express = require('express');
var app = express();

app.get("/*", (req, res) => {
res.sendFile(path.join(__dirname, "index.html"));
});

app.listen(9000, () => console.log("Running on 9000"));
```

The above code fires a server that runs on localhost port 9000. When a request is made to the home page http://localhost:9000/ ('/' indicates homepage) the server receives the request and the app.get("/") function fires and sends a response with index.html file.

index.html is a simple static HTML file that gets served as the homepage when a request is made. Depending on the server and application the homepage can be configured as required.

```
<!DOCTYPE html>
<html lang="en">

<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta http-equiv="X-UA-Compatible" content="ie=edge">
<title>Document</title>
</head>

<body>
<h1>This is API file</h1>
</body>

</html>
```

The homepage here simply displays the title 'This is API file' on the browser when a request is made for it.

The package.json file is where the configurations for the server and dependencies are configured. For the purpose of this project we have utilized the following dependencies - "cors", "express", "pg" these are application specific dependencies.

- **cors** is a package that enables cross-origin resource sharing, which makes use of additional HTTP headers in order to let a user gain permission of resources that are on a different server, in our case to gain access to resources in the remote server configuration.

- **express** as discussed previously, is a Node.js framework for server configuration.
- **pg** is the package that enables connection to the postgreSQL database.

Additionally, Node also supports packages that are only used in the development phase, they are listed under the devDependencies object, some of the important devDependencies used in this project are -

- **body-parser** is a middleware which parses the incoming request's body header in req.body property.
- **babel-cli** used for compilation of different Javascript files written in the latest ECMAScript 6 in order to transpile them to ECMAScript 5 which is understandable by the browsers.
- **nodemon** watches the server files for any changes and restarts the server automatically when changes are made, hence it gets rid of the manual restarting of the server needed to be done by the developer.

The server is fired by running the script in the scripts object present in the package.json file.

```
"scripts": {
"start": "nodemon --exec babel-node -- src/index.js"
},
```

nodemon is set to watch the index.js file which is where our server code lives and updates the server if any changes are made to this file. babel-node executes the ECMAScript 6 code in index.js and transpiles it to ECMAScript 5 which is understandable to the web browser. When the server is ready we get the following message in the console if everything worked -

```
[nodemon] 1.12.1
[nodemon] to restart at any time, enter 'rs'
[nodemon] watching: *.*
[nodemon] starting 'babel-node src/index.js'
Running on 9000
```

Therefore the server is now running and listening to requests on port 9000.

Below is the server setup code for the node server -

```
import express from "express";
import bodyParser from "body-parser";
import cors from "cors";

const app = express();
```

```
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(cors({ origin: "http://localhost:3000" }));

function fibonacci() {
        var a = 1,
        b = 0,
        num = 1400,
        temp;
        while (num >= 0) {
                temp = a;
                a = a + b;
                b = temp;
                num--;
        }
        return b;
}

app.get("/api/fibonacci", (req, res) => {
var result = fibonacci();
var users = req.query.users;
}
console.log(users);
res.status(200).json({ result, users });
});

app.listen(9000, () => console.log("Running on 9000"));
```

PHP Server

Local PHP servers can be set up relatively easily in comparison to the other two web technologies considered in this project. XAMPP for Windows makes setting up a local server on the machine relatively easy. It provides a WAMP stack (Windows, Apache, MySQL and PHP). The steps involved in setting up a PHP server are as follows -

- **Creating Virtual Hosts**

  There are a number of ways to serve static PHP files from the server. One way is to simply keep all server related files within the 'htdocs' folder in XAMPP. However, the drawback of this is that we have to enter long URLs for every page that we want to visit. Instead we can create a virtual host for each of our sites which have different URLs. The virtual hosts set up is located in the following file:

```
/xampp/apache/conf/extra/httpd-vhost.conf
```

At the bottom of the file append the following:

```
NameVirtualHost 127.0.0.1:80

<VirtualHost 127.0.0.1:80>
DocumentRoot "C:/xampp/htdocs/"
ServerName localhost
</VirtualHost>
```

This sets the root of localhost to 'C:/xampp/htdocs'. So all the server calls made to localhost will look for the url in this folder.

- Now all we have to do is run the XAMPP control panel and start the Apache server. This fires the PHP server on port 80 and serves the PHP files located inside the 'C:/xampp/htdocs' folder. If we navigate to 'http://localhost/' in our web browser, the server returns the index.php file since it is the homepage. The xampp/htdocs/ folder structure is as follows -

```
|-img
|-xampp
|-index.php
|-favicon.ico
```

  The 'img' folder contains all the static images that are used by the web application, favicon.ico is the icon that is visible on the title bar of the browser when we navigate to the website. index.php is the homepage of the website and is returned when a request is made from the browser.

Below is the server setup code for the PHP server -

```
<?php
$x = 0;
$y = 1;
$num = 1400;

while($num>=0) {
        $z = $x + $y;
```

```
                $x=$y;
                $y=$z;
                $num--;
        }
        echo JSONResponse($x, ($_GET['users']));
        ?>
```

Django Server

Django is extremely versatile in terms of how it can be installed and configured. Django can be:

- Installed from source, PyPi (Python Package Index) or by using package manager applications in UNIX operating systems.
- Installed on various operating systems.
- Configured to use several of the popular databases.
- Run within a Python virtual environment or within the system environment.

In order to install and setup Django, we need the following -

- **Python** - The Python version used in this project is version 3.6
- **pip** - Python package manager used to install Django.

Once we have these two requirements fulfilled, we simply need to install Django. It is done in the command prompt (on Windows) or terminal (on Linux) by running the following command -

```
pip install django
```

The django version used in this project is 2.0.2. Once everything is installed, we need to create our server, this is done in the terminal by running -

```
django-admin startproject projectname
```

Where project name is the name of the project. This creates a folder with the name provided as the projectname, with the following structure:

```
|-projectname
|---settings.py
|---urls.py
|-applicationname
|---migrations
|---views.py
|---urls.py
|---models.py
|-manage.py
```

The manage.py is the main file through which the server is built. It puts our project's package into the sys.path and also sets the DJANGO_SETTINGS_MODULE environment variable to make it point to our project's settings configuration file.

```
#!/usr/bin/env python
import os
import sys

if __name__ == "__main__":
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "projectname.settings")
try:
from django.core.management import execute_from_command_line
try:
import django
execute_from_command_line(sys.argv)
```

The projectname/settings.py file contains all the configurations for our server, such as database settings, application specific settings, middleware configurations, template routing, authentication handlers etc. projectname/urls.py manages the routing of the website. The 'urlpatterns' list routes URLs to views.

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
url(r'^admin/', admin.site.urls),
url(r'^$', include('fibonacci.urls')),
url(r'^api/fibonacci/', include('fibonacci.urls')),
url(r'^api/books/', include('books.urls'))
]
```

Different applications can be run on the same server. Django provides separation of each application into it's own environment which is then managed by the main server that sends the correct response as per the incoming requests. To create an application we run the following command in terminal -

```
python manage.py startapp applicationname
```

This creates a new application in our Django environment. The application has it's own views, models and urls for routing purposes.

```
|-applicationname
|---migrations
|---views.py
```

```
|---urls.py
|---models.py
```

The urls.py file contains the routing for the respective application. Thus when it gets a request from the user, it checks the url pattern and routes the appropriate view to be sent as a response.

```
from django.conf.urls import url
from . import views

urlpatterns = [
url(r'^$', views.index, name='index'),
]
```

Django processes an incoming request in the following way -

- Django determines which root URLconf package to use. Usually, the value is of the ROOT_URLCONF setting but when the incoming HttpRequest object already has a urlconf which would be set by a middleware, it will be replaced and it's value will be used instead of the ROOT_URLCONF setting.
- Django loads the Python module and looks for the variable called 'urlpatterns'. This is a Python list of django.urls.path() or django.urls.re.path() instance.
- Django checks through each URL pattern in order, and when it finds the first one that matches the requested URL it stops.
- Once a URL pattern match is found, Django imports the given view declared in the url. The view is simply a Python function. It gets the following arguments -
  - The HttpRequest instance
  - A match of the regular expression if the marched URL pattern returns no named groups

A view is a Python function that takes the web request from the client and returns a response. The contents of the response can be HTML, a redirect, an error 404, image, or any other piece of information. It contains the arbitrary logic necessary to return a response. The views.py file is where the logic for each route is written. Each function inside the views.py file represents a route and is executed when the urls.py file finds a matching route to execute and send as a response to the requesting client.

```
from django.http import HttpResponse
import datetime

def current_datetime(request):
now = datetime.datetime.now()
```

```
html = "<html><body>It is now %s.</body></html>" % now
return HttpResponse(html)
```

- First, we import the HttpResponse class from the django.http package, and Python's datetime package
- We define a function which acts as the view function (current_datetime in this case). Each view function take a request object as its first parameter.
- The view returns a response in the form of HttpResponse object which contains the response. Every view function must return an HttpResponse object.

A model in Django is a single source of information about the data in our database. It contains the information such as fields and behaviour of the data. Usually each model maps to a single database table.

- Each model is a Python class of django.db.models.Model
- Each attribute in the model represents a data field in the database
- Enables automatically generated database access API

Below is an example of a model called 'Books' which has values first_name and last_name

```
from django.db import models

# Create your models here.
class Books(models.Model):
name = models.CharField(max_length=255)
```

name is the field in the Books model. Each field specifies as a class attribute. They are attributed as database columns.

The Books model above would translate to the following SQL query:

```
CREATE TABLE myapp_books (
"id" serial NOT NULL PRIMARY KEY,
"name" varchar(255) NOT NULL,
);
```

Below is the code for the Django server setup -

```
                from django.http import JsonResponse
                from decimal import Decimal
```

```
def index(request):
        users = request.GET.get('users');
        a, b = 1, 1
        result = 'result'
        for i in range(1400):
                a, b = b, a + b
        data = {result: '%.2E' % Decimal(a)}
        return JsonResponse(data, users)
```

3.2 Remote Server Environment

The remote server setup is done using a cloud hosting platform DigitalOcean. As discussed in the previous chapter, DigitalOcean provides it's subscribed clients with virtual private servers (also known as droplets). In this section, each of the remote servers are set up on different virtual private networks located in the same data center in order to minimalize the effect of distance between client and server.

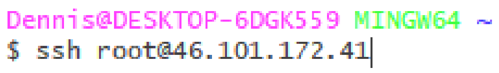
The remote servers are all set up on 64-bit Ubuntu 16.04 machines, with 20 GB SSD drive, 1 CPU, 512 MB RAM and 512 Mb/s bandwidth connection.

The servers are hosted on the following IP addresses -

- **Node** - 46.101.216.250
- **PHP** - 46.101.172.41
- **Django** - 46.101.169.55

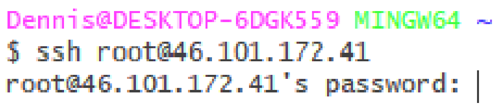
Connection to the remote servers is done via SSH in the terminal in the following way -

- In the terminal we establish an SSH connection to the remote server through 'root' user which is predefined when the VPS is created



```
Dennis@DESKTOP-6DGK559 MINGW64 ~
$ ssh root@46.101.172.41
```

- It will prompt for the password



```
Dennis@DESKTOP-6DGK559 MINGW64 ~
$ ssh root@46.101.172.41
root@46.101.172.41's password:
```

On successful login, connection to the remote server will be established and we can configure the server as per our requirements.

Transferring files to remote servers

There are two ways in which we can transfer our files to the remote servers. One is by using Git and the other is a file transfer software called Filezilla.

Git

Git, a distributed version control system that syncs repositories across different machines and Github to track code remotely. Creating a git repository is relatively easy. In this project, the Node remote server is setup using Git. In order to transfer files from a local machine, it needs to be located inside a directory with git initialized and synced with Github. This can be done by running the following command in the terminal -

```
git init
```

This command creates a git repository, with a .git directory with subdirectories for objects, refs/tags, refs/heads, template files etc. Now, any change that is made inside the directory is watched by git. It maintains a compared list between new versions and old version of the directory. The command 'git status' gives us the current status of the directory compared to the previous saved state.

In order to sync new changes to the repository, the command 'git commit' is used. A commit is a record of what files have been changed since the last commit or initial commit(git init). Commits help us to control the state of our code allowing us to rollback to a certain commit if required.

For the purpose of sharing code between the local and remote servers, Github enables developers to do so. By uploading the local git repository to Github we can share our code. This is done in the following way -

- First we add a remote git repository, this is done in the following way -

  ```
  git remote add origin https://github.com/username/repositoryname.git
  ```

  This tells the local git repository to sync with the remote repository on Github located in the specified URL.

- Once the remote repository is saved, push the current local repository and all it's files to the Github repository -

  ```
  git push -u origin branchname
  ```

  origin indicates the remote repository to push to and branchname is the branch we wish to upload.

- The code is now available on the remote Github repository and can be shared with any other computer.

Once the code is uploaded to the remote Github repository, it is time to download the code into the remote server.

- Login to the remote server via SSH using the appropriate IP address, username and password.

- In order to clone the repository into our remote server machine, we use the 'git clone' command

  ```
  git clone https://github.com/username/repositoryname.git
  ```

  This clones the git repository into the directory in the remote server. Now we can use the code the same way we did in the local machine.

Filezilla

Filezilla is a cross-platform FTP, SFTP and FTPS client with an intuitive graphical user interface. It allows transfer of files from one machine to another view the file transfer protocol. The server files for PHP and Django are transferred to their respective remote servers using the Filezilla client.

The main features of Filezilla are:

- Allows transfer of files in FTP, SFTP and also encrypted protocols of FTPS and SFTP

- Supports IPv6, the latest internet protocol version

- Directory comparison to compare local files and server files within the same directory. It also provides highlighting of missing or mismatched files

- Ability to configure network settings

- Support for remote file editing

- HTTP/1.1, SOCKS5 and FTP-Proxy support

Compared to git, transferring files via Filezilla is relatively easier and quicker to achieve. The user can directly login to the remote server within Filezilla using the IP address, username and password of the remote server, on success browse the directories in the server and make file transfers.

3.3 Testing tool

Apachebench is used to load test both the local and the remote servers. It provides most of the key metrics required to make our comparison of the performance of the various web technologies discussed in this project. The nature of benchmark testing is http based and Apachebench provides the functionalities of making requests with multiple users as well as with concurrency. Apachebench generates a flood of queries to the specified URL and returns a variety of metrics. These metrics can be used to compare the performance of various web technologies in our project. Some of the important metrics are requests/second, time per request, and transfer rate. The tool allows testers to load test a URL or server by replicating number of users and number of concurrent requests per second.

Apachebench comes bundled with the XAMPP installation required for PHP. In order to run Apachebench we need to navigate to the folder /xampp/apache/bin where the ab.exe file is located. Next, we load up the terminal and execute the command

```
ab -n numberOfUsers -c concurrency http://siteToBenchmark.com/
```

where numberOfUsers is an integer value which takes the number of users to benchmark the URL with and concurrency indicates the number of simultaneous users making the request to the server.

4 Test Methodology

The experiment evaluates the results of the server by performing benchmark tests. In all the tests, we follow the one-factor-at-a-time experimental design [**?**] to ensure the accuracy and effectiveness of tests.

4.1 Performance Testing

Performance Testing is a type of testing that ensures software applications perform well under certain expected workload. The Features and Functionality supported by a software system is not the only thing to be concerned about. An application's performance parameters such as its response time and reliability matters. The goal of Performance Testing is to eliminate performance bottlenecks.

The goal of Performance Testing is to check a software program's:

- Speed - Determines whether the application responds quickly
- Scalability - Determines maximum user load the software application can handle.
- Stability - Determines if the application is stable under varying loads

Performance testing also known as as "Perf Testing" is a subclass of performance engineering. It is done to provide developers information about the application regarding speed, stability and scalability. Further, performance testing helps to discover what areas need to be improved before the application goes live. In the absence performance testing, applications are likely to face issues such as: bottlenecks while several users use it simultaneously, inconsistent results across different operating systems and low quality of usability. Performance testing determines whether or not the application meets the requirements of speed, scalability and stability under expected workloads. Applications deployed on the internet with poor performance numbers due to none or poor performance test are likely to gain a bad reputation and fail to meet expected sales goals. Also, applications with high critical usage like space programs or medical tools should be tested throughly to ensure that they function for long period of time without high accuracy.

Common Performance Problems

Most performance problems focus on speed, response time and load time. Speed is usually one of the most important factors of an application. An application that runs slow will lose a lot of potential users. Performance testing is carried out in order to make sure an application runs fast to keep a user's attention and interest. The following is list of common performance problems highlighting how speed is a common factor in many of them:

- Long load time - Load time is usually the initial time it takes for an application to start up. This should generally be kept as low as possible. While it is impossible to make load time for certain applications in under a minute, it should be kept at least under a few seconds.

- Poor response time - Response time is the time taken from when a user enters data into the application to when the application sends a response to that particular input. Usually this should be very fast.
- Poor scalability - An application suffers from poor scalability when it is unable handle an expected number of users or when it cannot accommodate a wide enough range of users. Performance Testing should be carried out to be sure that the application can handle the expected number of users.
- Bottlenecking - Bottlenecks are obstacles in the application that lowers overall system performance. It occurs when either coding errors or hardware problems results in a decrease of requests per second under certain loads. Bottlenecking is often caused by one faulty section of code. The key to fixing a bottlenecking issue is to find the section of code that is causing the slow down and try to fix it there. Bottle necking is generally fixed by either fixing poor running processes or adding additional Hardware. Some common performance bottlenecks are CPU utilization, Memory utilization, Network utilization, Operating System limitations, Disk usage.

Performance Testing Process

The philosophy undertaken for performance testing can differ widely but the goal for performance tests remain the same as before. It can help to show that your product's system meets certain pre-characterized performance criteria. Also, it can help compare performance of two software programs. Likewise, it can also help distinguish parts of the software system which lowers the quality of its performance.

Below is a generic performance testing process -

- **Identify your testing environment** - Familiarize with the physical test environment, production environment and which testing equipments are accessible. Comprehend the details of the hardware, software and network settings utilized during testing before you start the testing procedure. It will help testers create more proficient tests. It will also help in identifying the possible difficulties that testers may face during the performance testing process.
- **Identify the performance acceptance criteria** - This incorporates objectives and limitations for throughput, response and resource allocations. It is also important to distinguish project success criteria beyond these goals and constraints. Testers should be enabled to set performance criteria and objectives because frequently the project definition will not include a wide enough quantity of performance benchmarks. It is possible that sometimes there may be none at all. If possible, finding a similar application to compare to is an acceptable method to set performance goals.

- **Plan and design performance tests** - Decide how the usage is probably to fluctuate among end users and distinguish key situations to test for all possible use cases. It is necessary to simulate different kinds of end users, design performance test data and outline what metrics will be collected.
- **Configuring the test environment** - Prepare the testing environment before execution. Also, arrange tools and other resources.
- **Implement test design** - Create the performance tests according to the design.
- **Run the tests** - Execute and monitor the tests.
- **Analyze, tune and retest** - Combine, investigate and share test outcomes. Then adjust and test again to check if there is an change in performance. Since improvements usually grow smaller with each test, stop when the CPU reaches bottleneck.

4.2 Benchmark Test Methodology

Benchmark Testing is a kind of testing done to give repeatable set of quantifiable outcomes from which present and future application releases for particular functionality can be baselined or thought on. It is a procedure used to analyze the performance of software systems also known as SUT (System Under Test).

A benchmark must be repeatable. For example, with each instance of load test, if the response times fluctuates too much, system performance needs to be benchmarked. Response time should be steady amongst various load conditions.

A benchmark must be quantifiable. For instance, user experience cannot be evaluated in numbers, however time a user spends on a website due to good user interface can be quantified.

Importance of benchmark testing

At business level, benchmark testing can be helpful to determine -

- How well a web-based application is performing with respect to the competitors
- It ensures that websites complies with standards and best practices
- It enables to evaluate third- party service providers prior to making a contracting decision
- Allows to figure out the mistakes to be avoided
- How different types of customers experience the response time and availability of a site.

4.3 Phases of benchmark testing

Planning Phase

- Identifying and organize standards and requirements
- Decide benchmark criteria
- Define benchmark test procedures

Analysis Phase

- Identify root cause of error to enhance quality
- Setting goals for test process

Integration Phase

- Share results with concerned individuals and get approval
- Establish functional objectives
- Define benchmark test process

Action Phase

- Design test plan and documentation
- Implement actions determined in previous phases and monitor advances
- Continously run the process

5 Experimental Results and Analysis

5.1 Test setup

The benchmark test were performed in three separate scenarios - CPU intensive task on local server, CPU intensive task on remote server and a SELECT database query on local server with a local database setup.

According to one-factor-at-a-time experimental design, we make three fundamental tests - CPU intensive calculation of fibonacci value on local server and remote server, select query to local database in local server. The calculate Fibonacci module calculates some value of Fibonacci and evaluates the performance under compute-intensive tests. Select Operation from local DB module compares different performance through querying some data of DB in an IO-intensive situation. Under all benchmark tests, we keep number of requests at 200 with a concurrency of 1. TABLE 1 summarizes the factors in our experiments.

Table 1: Benchmark test factors

Users	100, 200, 300, 400, 500
Requests	200
Benchmark test module	Calculate fibonacci local & remote server, SELECT query local DB
Web technologies tested	Node, PHP, Python-Django

In order to achieve fair results, the server machines are rebooted after each test in order to ensure no additional resources are being used up by other running processes or previously running processes.

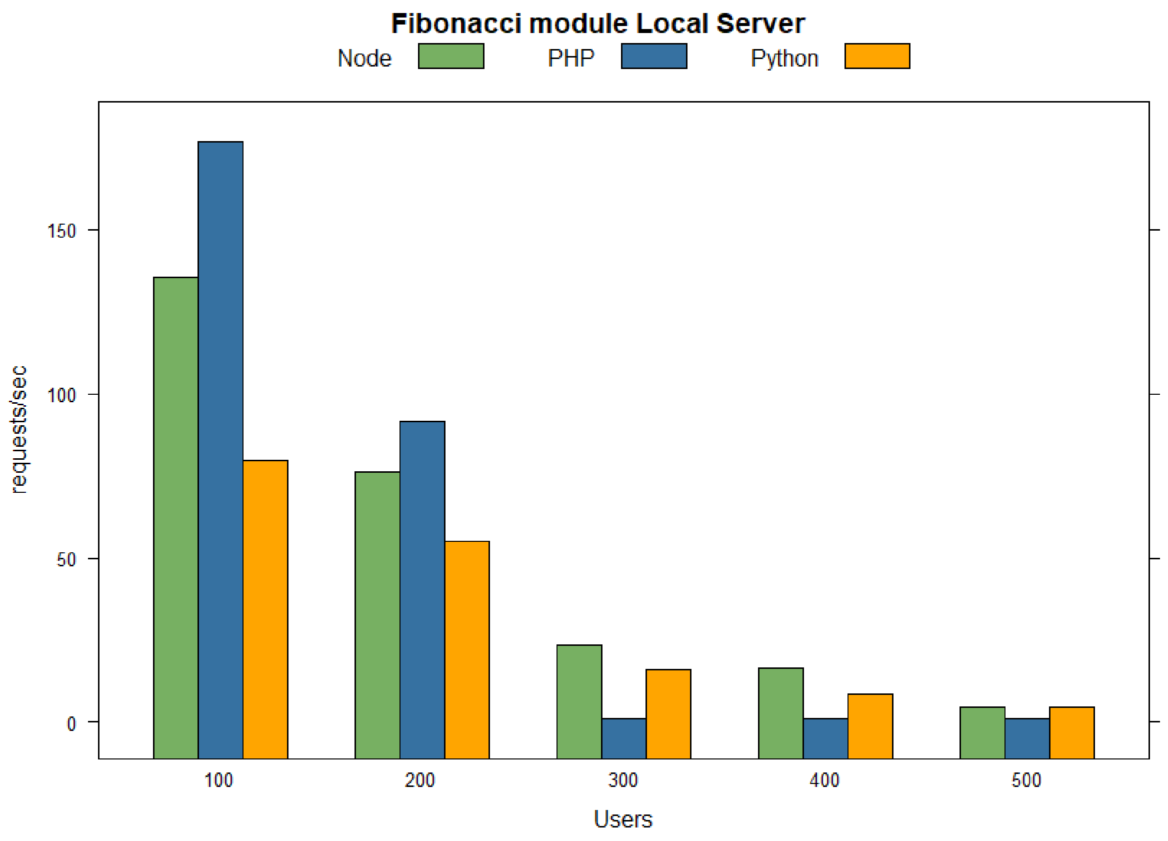


Figure 1: Throughput, Fibonacci module on local server

Table 2: Tabular results for fibonacci module on local server

Users	Node req/sec	PHP req/sec	Python req/sec
100	135.37	176.60	79.58
200	76.06	91.72	55.18
300	23.35	0.99	16.08
400	16.35	0.99	8.37
500	4.70	0.89	4.64

5. EXPERIMENTAL RESULTS AND ANALYSIS

The graph was obtained by testing the Fibonacci module on the local server machine on the three specified web technologies. It consists of a grouped column chart with the number of Users(requests) on the x-axis and Throughput on the y-axis. The readings are taken at intervals of 100, 200, 300, 400 and 500 users.

From the graph, we can observe that with the growth of users, the performance of the three technologies shows a decreasing trend. This is expected since as the number of users increases, it increases the load on the server thus slowing down the response time. The throughput is highest in the case of PHP for 100 users with 176.60 requests per second. It is followed by Node which comes in at 135.37 requests per second and then Django at 79.58 requests per second.

As the number of users increases there is a downward trend in the value of throughput. At 200 users we observe a similar result with PHP outperforming Node and Django with 91.72 requests per second, Node with 76.06 and Django with 55.18 requests per second.

However, as we reach the 300 users mark, the result is quite different for the case of PHP. While Node and Django show consistent results, as in decreasing throughput, PHP completely crumbles at this threshold and manages to only get 0.99 requests per second. This suggests that the server is saturated and cannot handle such large number of requests. With 400 and 500 users we observe a similar situation, with PHP server being completely saturated with 0.99 and 0.89 req/sec respectively, while Node and Django even though have low response rates, have not saturated their servers.

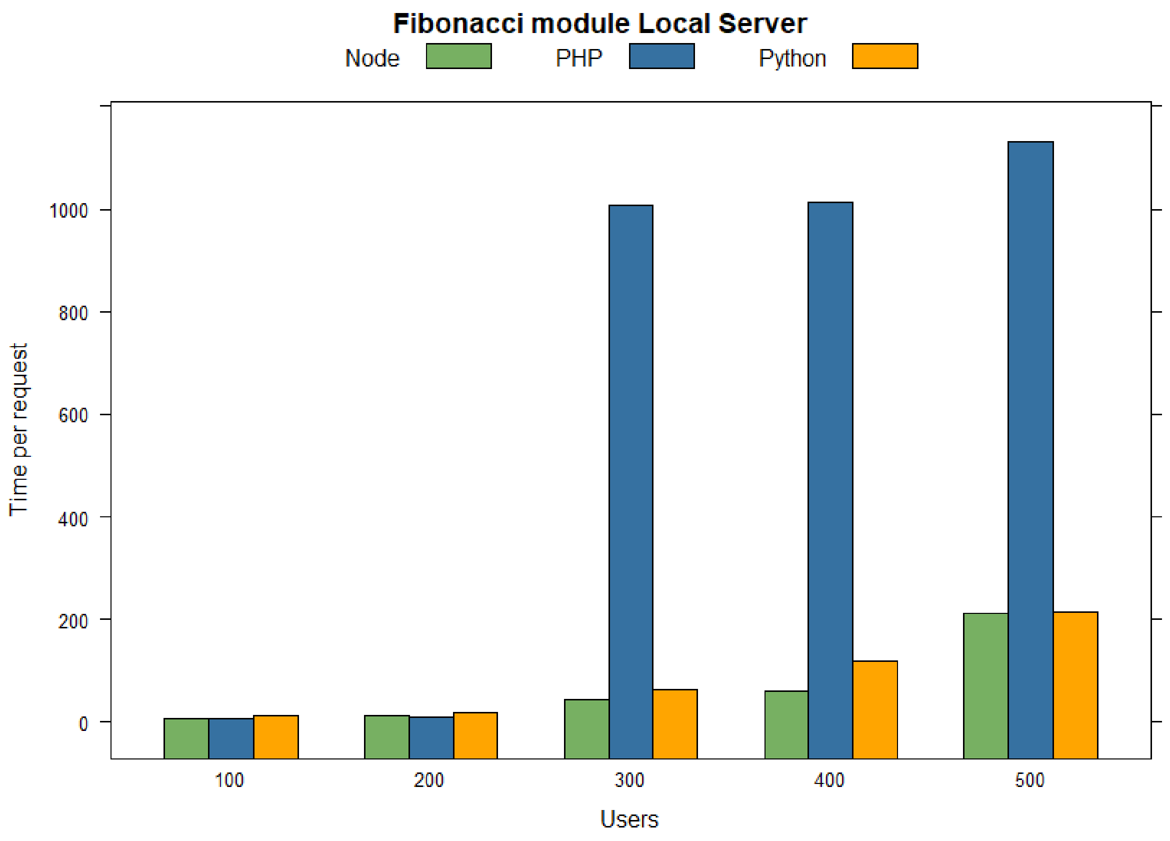


Figure 2: Time per request, Fibonacci module on local server

Table 3: Tabular results for fibonacci module on local server

Users	Node	PHP	Python
100	7.38 ms	5.66 ms	12.56 ms
200	13.14 ms	10.90 ms	18.12 ms
300	42.83 ms	1008.46 ms	62.19 ms
400	61.17 ms	1011.46 ms	119.07 ms
500	212.94 ms	1129.60 ms	215.63 ms

In the above graph we observe the results of the time taken for each request to be completed by testing the Fibonacci module on the local server machine. It consists of a grouped column chart with Users(requests) on the x-axis and time per request on the y-axis.

From the graph, it is evident that as the number of users increases, the time required to complete each request increases and there is a clear correlation between this graph and the graph for request per second. As the number of users increases, the processing time for each request also increases thus taking more time to complete each request. The time per request is lowest in case of PHP for 100 users with 5.66 ms per request. It is followed by Node with 7.38 ms per request and Python 12.56 ms per request.

With growing number of users there is a similar observation at 200 users per second with PHP again outperforming Node and Python with 10.90 ms per request against Node's 13.14 and Python's 18.12 ms per request.

However, as expected from the graph of request per second, when the users is at 300 and beyond, the PHP server gets saturated and the time taken to complete each request skyrockets in comparison to Node and Python which have more consistent results as expected. At this point, PHP takes 1008.46 ms to complete each request which indicates a saturation point in the server, while Node and Python remain unsaturated and clock in times of 42.83 ms and 62.19 ms per request respectively.

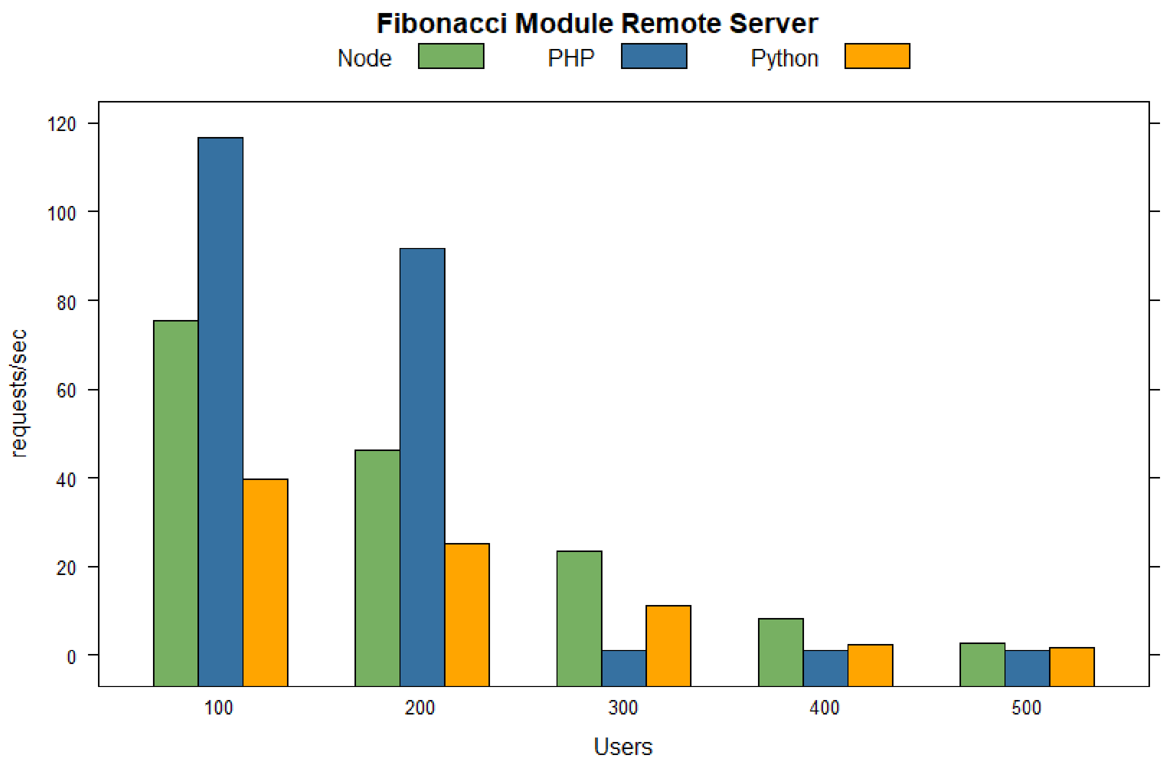


Figure 3: Throughput, Fibonacci module on remote server

Table 4: Tabular results for fibonacci module on remote server

Users	Node req/sec	PHP req/sec	Python req/sec
100	75.37	116.60	39.58
200	46.06	91.72	23.23
300	21.67	0.99	11.88
400	8.78	0.99	2.54
500	2.76	0.99	1.49

5. EXPERIMENTAL RESULTS AND ANALYSIS

The graph was obtained by testing the Fibonacci module on the remote server machine hosted on DigitalOcean virtual private network to simulate real-world request type scenario over the Internet. It is a grouped column chart with number of users on x-axis and throughput on the y-axis.

As seen previously PHP again performs much better when the number of users is relatively low at 100 and 200 users. The graphs shows a similar downtrend as the number of users increases progressively to 500. PHP processes 116.60 requests per second at 100 users while Node comes in at second with 75.37 req/sec and Python with 39.58 requests per second. A similar result is obtained when the number of users increases to 200 with PHP at 91.72 req/sec, Node 46.06 req/sec and Python at 23.23 req/sec.

When the number of users increases to 300 and beyond, we obtain a repeat of what we observed with the servers set up on the local machine. The requests processed per second by PHP plummets to 0.99 req/sec indicating signs of saturation while Node and Python maintain their gradual decrease in the number of requests processed per second.

It is important to note that even though the graph trend on the remote server is very identical to the graphs obtained on the local machine, the overall requests processed per second in the case of the remote server is lower. The most likely cause of this is the bandwidth and distance between the client making the request and the server.

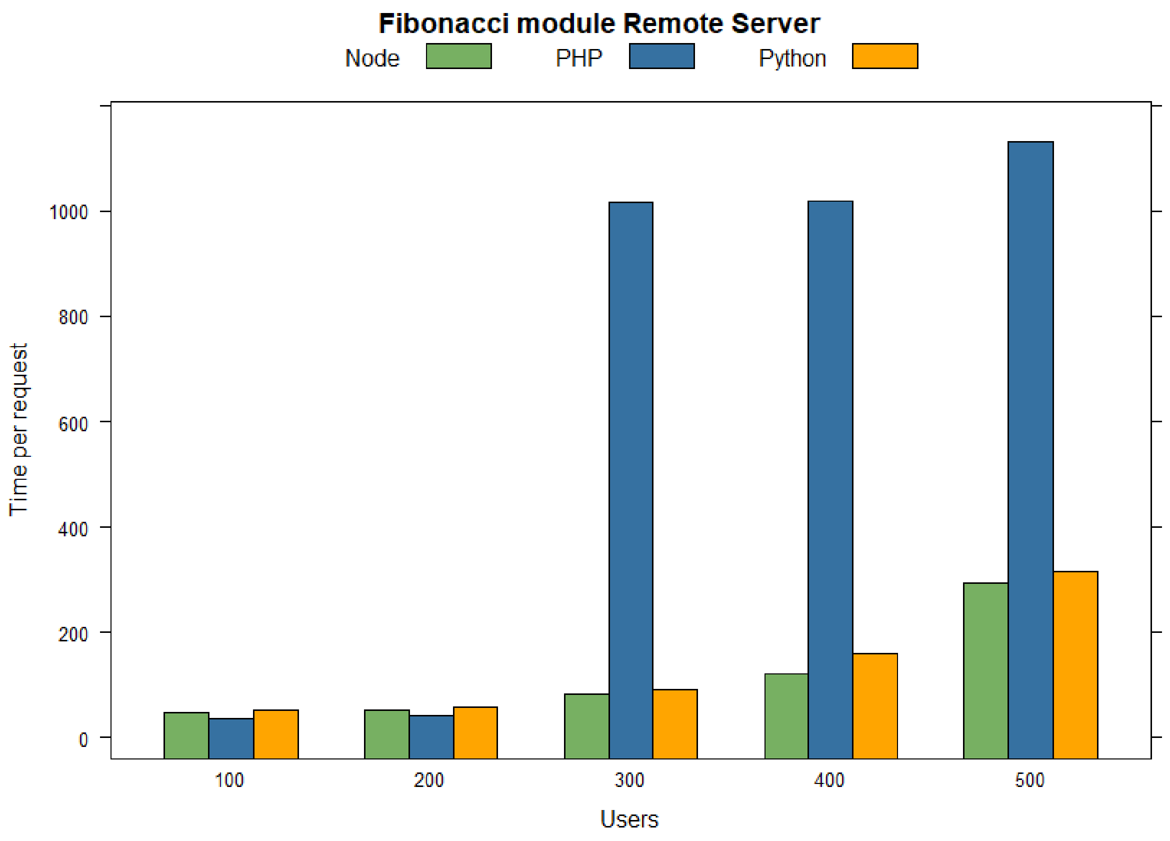


Figure 4: Time per request, Fibonacci module on remote server

Table 5: Tabular results for fibonacci module on remote server

Users	Node	PHP	Python
100	47.23 ms	35.24 ms	52.76 ms
200	53.15 ms	40.84 ms	58.01 ms
300	82.46 ms	1015.34 ms	92.13 ms
400	121.13 ms	1018.64 ms	159.12 ms
500	291.41 ms	1129.14 ms	315.22 ms

The graph was obtained by testing the Fibonacci module on the remote server machine hosted on DigitalOcean virtual private network to simulate real-world request type scenario over the Internet. It is a grouped column chart with number of users on x-axis and time per request (in ms) on the y-axis.

From the graph, we observe that as in the case with the local server, we obtain similar results. As the number of users increases, the time required to complete each request increases and again there is a clear correlation between this graph and the graph for request per second. As the number of users increases, the processing time for each request also increases thus taking more time to complete each request. The time per request is lowest in case of PHP for 100 users with 35.24 ms per request. It is followed by Node with 47.23 ms per request and Python 52.76 ms per request.

With growing number of users there is a similar observation at 200 users per second with PHP again outperforming Node and Python with 40.84 ms per request against Node's 53.15 and Python's 58.01 ms per request.

However, as expected from the graph of request per second, when the users is at 300 and beyond, the PHP server gets saturated and the time taken to complete each request rapidly increases in comparison to Node and Python which have more consistent results as expected. At this point, PHP takes 1015.34 ms to complete each request which indicates a saturation point in the server, while Node and Python remain unsaturated and clock in times of 82.46 ms and 92.13 ms per request respectively.

We observe that both the cases for request per second and time per request, once the PHP server is saturated there is not much difference in their results. This is an interesting outcome as it indicates that the bandwidth causes a smaller effect on the performance of the server once the server is saturated.

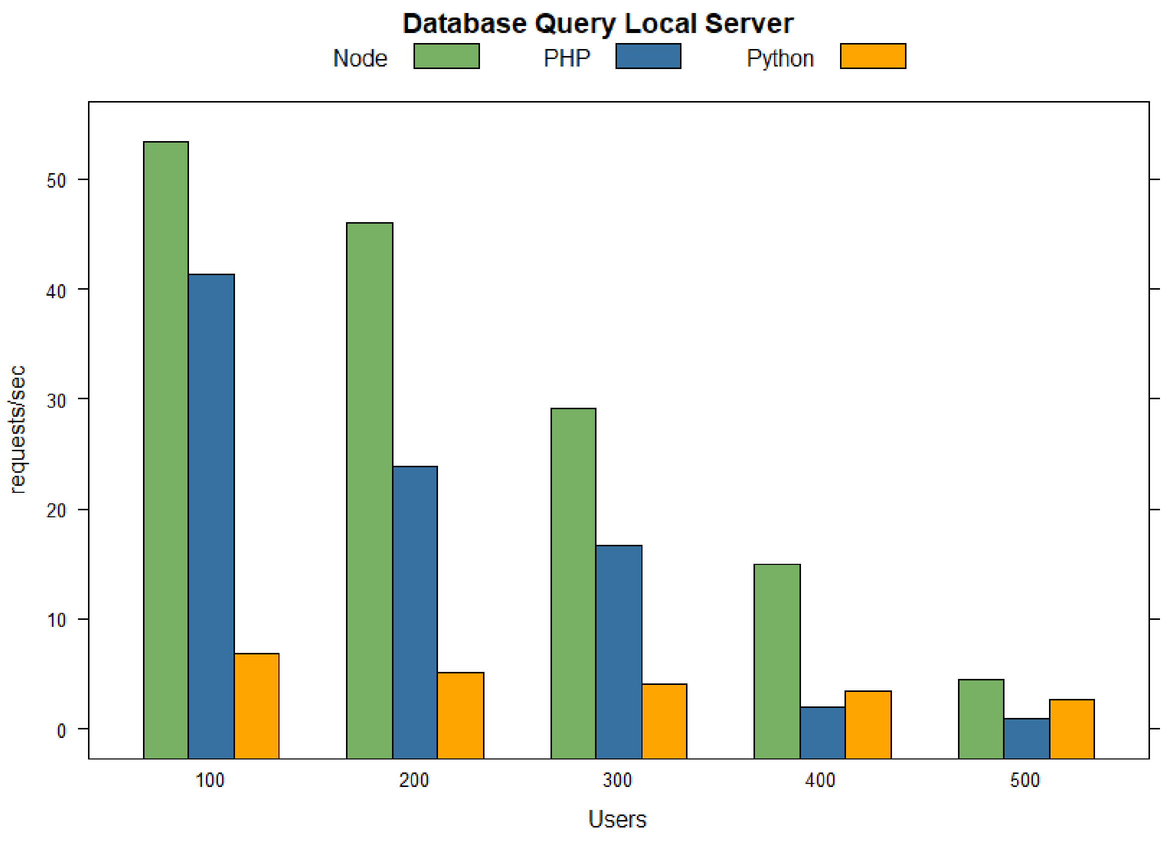


Figure 5: Throughput, Database query on local server

Table 6: Tabular results for Database query on local server

Users	Node req/sec	PHP req/sec	Python req/sec
100	53.22	41.35	6.86
200	46.06	23.83	5.12
300	29.18	16.07	4.10
400	14.99	1.96	3.42
500	4.5	0.89	2.52

The graph was obtained by testing the database query on the local server machine on the three specified web technologies. It consists of a grouped column chart with the number of Users(requests) on the x-axis and throughput on the y-axis. The data is stored in a relational database using PostgreSQL.

We obtain a very interesting result in this test. Unlike the previous results on the fibonacci module, PHP does not perform as well as it did previously. We observe a more consistent result among all the three technologies at least up until 300 users. Node performs the best out of the three with 52.22 req/sec at 100 users followed by PHP at 41.35 req/sec and then Python with 6.86 req/sec. A similar trend is observed at 200 and 300 users with each technology showing consistently decreasing results.

However, at 400 and beyond, the PHP server again saturates, causing a drop in the throughput, even below that of Python's. At 400 users, Node comes in at 14.99 req/sec with Python in second at 3.42 req/sec and then PHP with 1.96 req/sec. There is a drastic fall in the req/sec processed by PHP indicating that the server has saturated.

What we observe in the three technologies in this scenario is that Node performs better in an I/O intensive operation rather than a CPU intensive operation. PHP is able to handle more users before reaching it's saturation point in case of I/O operations, but once it reaches saturation, it again drop to 1 req/sec range. Python overall has considerably low throughput value, indicating that it may not be the best option for I/O operations, however, it still shows consistent readings and does not reach saturation as easily as PHP does.

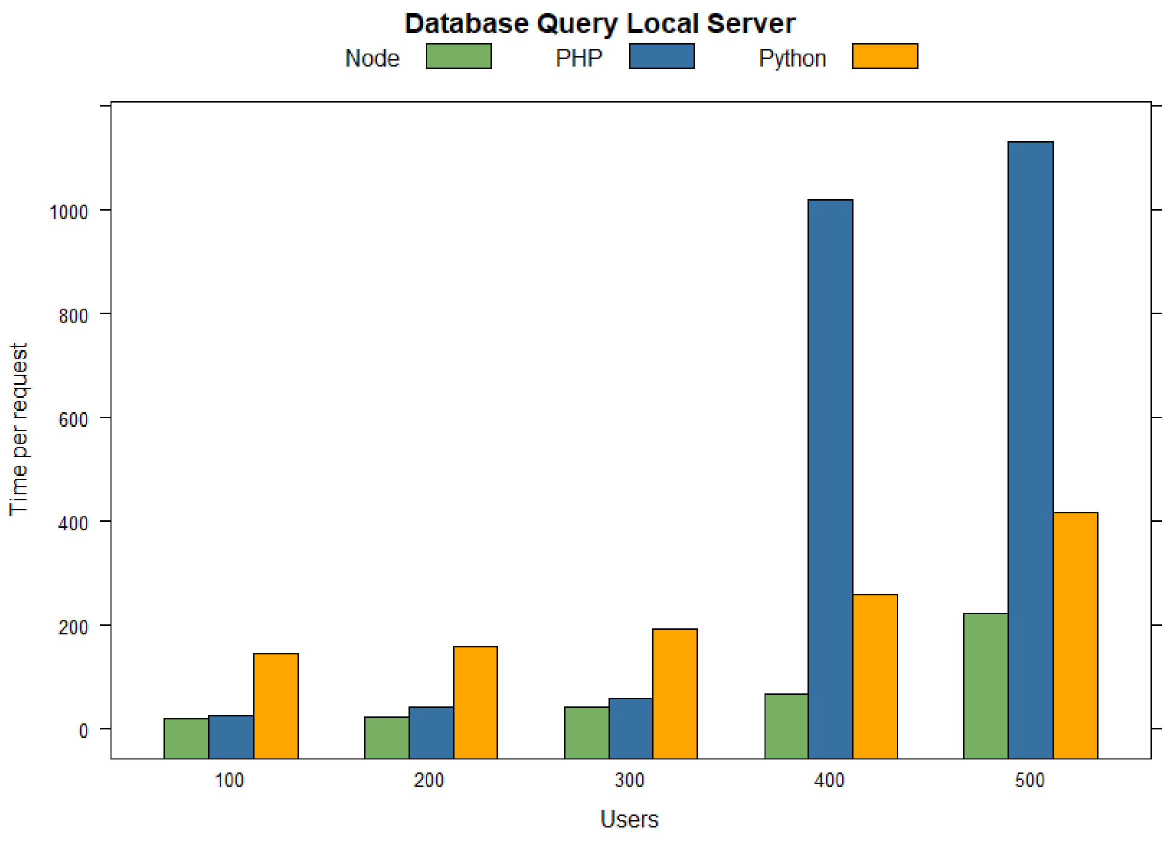


Figure 6: Time per request, Database query on local server

Table 7: Tabular results for fibonacci module on remote server

Users	Node	PHP	Python
100	18.74 ms	24.20 ms	145.86 ms
200	21.51 ms	41.43 ms	158.12 ms
300	42.83 ms	59.54 ms	192.42 ms
400	66.70 ms	1018.46 ms	259.24 ms
500	222.94 ms	1129.60 ms	415.23 ms

The graph was obtained by testing the database query on the local server machine on the three specified web technologies. It consists of a grouped column chart with the number of Users(requests) on the x-axis and time per request on the y-axis. The data is stored in a relational database using PostgreSQL.

As seen in the above test for database query of users vs. req/sec, we observe a strong correlation between the two graphs. Node has the lowest time taken to complete a request coming in at 18.74 ms per request followed by PHP at 24.20 ms per request and eventually Python at 145.86 ms per request.

At 400 and beyond, the PHP server again saturates, causing a spike in the time per request, even over that of Python's. At 400 users, Node comes in at 66.70 ms per request with Python in second at 259.24 ms per request and then PHP with 1018.46 ms per request. There is a drastic increase in the time per request processed by PHP indicating that the server has saturated.

6 Conclusion and Future Work

The research carried out in the project considers the study of performance characteristics of three web technologies, namely Node, PHP and Python-Django. The study was performed with object systematic benchmark tests in two aspects of wbe requests scenarios - CPU intensive task and I/O operation task. The CPU intensive task was to perform the caluclation of a large Fibonacci value and return it to the user. The I/O intensive task involved a query to the database.

As per the experiments carried out, the results obtained in the CPU intensive task vs I/O intensive task were quite contrasting. PHP performed very well in CPU intensive tasks when the volume of users was relatively low. However, as the number of users crossed beyond the 300 threshold mark, the PHP server began to bottleneck on both the local and remote machines, while Node and Django managed to perform in a consistent manner as expected with increasing number of users. This suggests that Node and Django are able to handle a higher number of requests before experiencing server bottlenecking like PHP. Further if we compare the performance of Node and Django, Node outperforms Django at the lower user numbers but as the number of users increases the difference between their performances decreases and we see similar results for both the technologies.

In the I/O intensive task of performing a database query, we observe Node performing much better than PHP and Django in all cases of increasing users. This justifies the I/O asynchronous nature of Node which uses callbacks to return to a function when finished so it can continue executing other requests. PHP seems to follow the same pattern as with the CPU intensive task, performing well for low number of users but experiencing failures when the number of users exceeds a threshold. However, in the case of I/O intensive task, PHP's threshold is slightly higher than that to the CPU intensive task. Django however has the lowest performance when it comes to I/O intensive task.

Summarizing the research on these three technologies, we conclude with the best use case for each of them as follows -

- Node is suitable for applications that have high volume of short message requests that require low computing power, for example real-time applications like livechat, instant-messaging apps etc. will perform well when run on Node compared to the other technologies, since data syncing between the client and server can be achieved very fast. Due to it's asynchronous nature, Node is able to process many requests with low response times.
- PHP being one of the oldest web technologies, still performs well under certain situations. For small-scale applications, PHP would give very good performance since it is able to handle relatively smaller number of requests better.
- Django lies somewhere between Node and PHP when it comes to performance. In all of the

tests performed, Django remained consistent with its results across the range of users. Thus Django is built to be robust and with further developement of this relatively new technology we may see improvements in its performance in the future, making it suitable for both CPU intensive and I/O intensive tasks.

References

[1] Node Documentation
https://nodejs.org/en/about

[2] Joseph Ottinger - What is a web server?
http://www.theserverside.com/news/1363671/What-is-an-App-Server

[3] Number of JavaScript repositories on Github
https://github.com/search?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] Node.js w/1M concurrent connections
http://blog.caustik.com/2012/08/19/node-js-w1m-concurrent-connections/

[14] Laurent Orsini, *What You Need To Know About Node.js*

[15] Aleksander Kasiuk, *On problems with threads in node.js - Future Processing*, April 2015.

REFERENCES

[16] Express Wikipedia
https://en.wikipedia.org/wiki/Express.js

[17] PHP Documentation
https://secure.php.net/manual/en/intro-whatis.php

[18] .NET Core Documentation
https://docs.microsoft.com/en-us/dotnet/framework/get-started/net-core-and-open-source

[19] Apachebench
https://httpd.apache.org/docs/current/getting-started.html

[20] JMeter
https://www.tutorialspoint.com/jmeter/