School of Business: Faculty Scholarship   School of Business

2023

Node.js or PHP? Determining the Better Website Server Backend Scripting Language

Qozeem Odeniran
Georgia Southern University

Haydeen Wimmer
Georgia Southern University

Carl Rebman
University of San Diego, carlr@sandiego.edu

Node.js or PhP? Determining the better website server backend scripting language

Qozeem Odeniran, *Georgia Southern University, qo00109@georgiasouthern.edu*
Hayden Wimmer, *Georgia Southern University, hwimmer@georgiasouthern.edu*
Carl M. Rebman, Jr., *University of San Diego, carlr@sandiego.edu*

Abstract

Most people interact with websites expecting them to perform quick results and provide quick responses to their requests and many do not realize the performance is due to server side or backend programming. There are several types of backend web framework/scripting technologies. Programmers and developers often debate over which is the technologies is the better solution. Most debates are based on various dimensions such as performance, scalability, and architecture. The most common factor for settling the debate or choosing the most appropriate frameworks tends to be the performance dimension. This study assesses the performance of both Node.js and PHP by implementing well-known algorithms of binary, bubble, and quick sort along with Heap's algorithm for permutations. These algorithms were selected for their increasing time complexities which allows us to observe the performance differences between the backend framework/scripting. By comparing the performance of these two backend scripting technologies, one can gain a better understanding of the circumstances when migrating from PHP to Node.js would be beneficial. Our results showed that a significant difference occurs in the performance of PHP and Node.js and specifically, Node.js outperformed PHP in terms of latency and other performance metrics. This study provides valuable information for software engineers, developers, and managers who are seeking the best framework for their web applications.
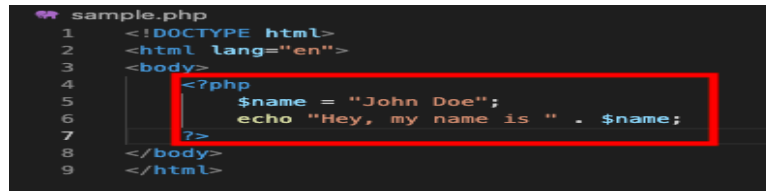
Keywords: Node.js, PHP, backend scripting technologies, sorting algorithms, performance, Apache JMeter

Introduction

Back-end scripting technologies are crucial in web development, as they greatly enhance the development of web applications and services. The popularity of backend scripting technologies is constantly evolving, with new backend scripting technologies emerging on a regular basis. Some popular backend platforms are PHP, node.js, Django, Flask. Ruby, PERL, and ASP.net. PHP is one of the nost popular development platform and is one of the oldest web development platforms. PHP is a powerful language even though modern applications are transitioning away from PHP. PHP most likely will remain a viable development platform due to the amount of legacy code that is powering the interweb. For performance and scalability, the current language of choice is JavaScript, and it is the fastest growing web development platform referred to as node.js.

PHP, otherwise known to many as personal home page (although it really stands for PHP Hypertext Processor) has been around since 1996 and is considered quite easy to use (https://www.softwareengineerinsider.com/programming-languages/php.html). PHP evolved over a need to be able to make more responsive and interactive web pages than what regular static HTML can offer.
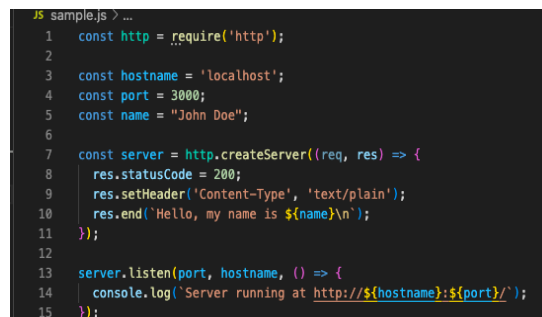
PHP is a general-purpose scripting language that works well with web development and is mainly used for dynamic webpages, while Node.js developed on chrome's JavaScript runtime for developing fast and scalable network apps (Lei et al., 2014). It supports object-oriented programming, is open source, simple to learn, and integrates well with HTML. Even now, PHP is still used as the back-end scripting technology for many millions of websites. To execute it successfully, a server and PHP editor are required. The most recent version is PHP-8.

```
sample.php
1    <!DOCTYPE html>
2    <html lang="en">
3    <body>
4        <?php
5            $name = "John Doe";
6            echo "Hey, my name is " . $name;
7        ?>
8    </body>
9    </html>
```

Figure 1: A basic PHP code inside HTML page.

Node.js is a younger language than PHP and like PHP it was developed to help make more interactive web pages that could collect data from webforms (https://www.section.io/engineering-education/history-of-nodejs/). Node.js adopts an event-driven, non-blocking I/O style which efficient handling of large concurrent connections and is based on Chrome's JavaScript runtime. Whenever a request is received by Node.js server, the event is placed on a queue while node.js event loop picks up the event and handles it asynchronously. This makes node.js to simultaneously handle multiple requests without event-loop blockage or performance issue. On the other hand, when PHP executes, a PHP interpreter processes it on the server-side which generates a Hypertext Markup Language (HTML), or other output that can be sent to web browser's client (Prayogi et al., 2020). With over 800,000 GitHub repositories, businesses are seeing the advantages of Node.js more and more. On GitHub, JavaScript is now the most widely used language.

```
JS sample.js > ...
1    const http = require('http');
2
3    const hostname = 'localhost';
4    const port = 3000;
5    const name = "John Doe";
6
7    const server = http.createServer((req, res) => {
8      res.statusCode = 200;
9      res.setHeader('Content-Type', 'text/plain');
10     res.end(`Hello, my name is ${name}\n`);
11   });
12
13   server.listen(port, hostname, () => {
14     console.log(`Server running at http://${hostname}:${port}/`);
15   });
```

Figure 2: A basic node.js code

Determining the best backend web framework is an issue that concerns developers and managers alike. One of the challenges is that there are many different types of technologies out there and it is not always easy to know which is best for your application. Another factor is that some technologies either cannot work on some platforms, or they do not perform well. In fact, making the wrong decision can cause the application to fail. Node.js and PHP are two popular backend scripting technologies that many developers have adopted for building web applications. Both technologies dominate the internet because they work well on many platforms.

This study compares both PHP and Node.js in terms of performance using common sorting algorithms and a heap algorithm to generate all possible permutations. The goal is to provide a comprehensive analysis of the performance of these two backend scripting technologies in handling complex algorithms and processing large data. Understanding and analyzing their performance characteristics is critical to the

success of one's application and both technologies have unique features. This understanding can lead management and developers to select the framework that suits their development needs.

This study aims to provide insights that can help organizations and software engineers make a well-informed choice on which backend scripting technologies to adopt for their web development projects. The rest of this paper is structured as follows. First, is a discussion of the relevant literature, followed by our methodology and test results. Results indicate that Node is a more efficient backend web framework but there are additional considerations which must be reviewed by developers and managers to select the most appropriate framework for the application.

Literature Review

Information systems play crucial roles in business processes in educational institutions, from student and course registration to end-of-semester/session evaluations and graduation processes (Prayogi et al., 2020). Prayogi et al. (2020) was interested in the fact that data exchange within those systems becomes increasingly important and their research study was to prototype the development of a REST API. This API was specifically for academic information systems and meant to analyze its performance by implementing it using two different server technologies: PHP and Node.js. The prototype that Prayogi et al. (2020) implemented was developed using a database with one sample table representing employees in a college, and two different endpoints were created for the implemented REST API. In the experiment, Apache JMeter was used to simulate a thousand concurrent requests on the database .Node.js consistently outperformed PHP in the experiment, achieving a 100% throughput for all concurrent 1000 requests, while PHP recorded a throughput of only 48.70% under the same conditions (Prayogi et al., 2020).

Chaniotis et al. (2015) discussed the implications of developing end-to-end web applications in the social web era and examines a distributed architecture suitable for modern web application development. The aim of their study was to find the most efficient and scalable technology stack for web application development in the current social web era. They conducted stress tests on popular server-side technologies, including PHP/Apache stack, Nginx, and Node.js, to determine their efficiency and scalability. The study found that the PHP/Apache stack was not efficient in handling increasing demand in network traffic, while Nginx was more than 2.5 times faster in I/O operations than Apache. Node.js outperformed both in I/O operations and resource utilization but lacked in serving static files. The study concluded that building cross-platform applications using web technologies is feasible and productive, and Node.js is an excellent tool for developing fast, scalable network applications that offer client-server development integration and aid code reusability (Chaniotis et al., 2015).

Raharjo (2014) compared Node.js and the PHP/Nginx web development stack and analyzed their performance and scalability. The study assessed and compared the performance and scalability of Node.js and PHP/Nginx web applications using a load generator. The study also designed mock applications based on the Dijkstra Algorithm and utilized the load generator to simulate concurrent user requests and assessed the performance and scalability of Node.js and PHP/Nginx web applications. The study determined that Node.js applications performed better and were more scalable than PHP/Nginx applications (Raharjo, 2014).

Lei et al. (2014) highlighted the significance of large-scale, high-concurrency, and data-intensive web applications in the latest generation of websites. Node.js has become popular for building such applications. Lei et al. (2014) were interested in comparing the performance of Node.js, Python-Web, and PHP in constructing data-intensive web applications using benchmark and scenario tests. Lei et al. (2014)

conducted benchmark and scenario tests to compare the performance of Node.js, Python-Web, and PHP in creating data-intensive web applications. The benchmark tests assessed performance data, while the scenario tests simulated realistic user behavior. The study results revealed that Node.js has a much higher capacity than PHP and Python-Web for handling requests within a specific time frame. The authors concluded that Node.js is lightweight and efficient, making it an excellent choice for I/O intensive websites. PHP is only suitable for small and medium-scale applications, while Python-Web is developer-friendly and suitable for large web architectures. This study is the first to assess these web programming technologies using both objective systematic tests and realistic user behavior tests, with Node.js being the primary focus of discussion (Lei et al., 2014).

Odeh (2019) compared the quantity of websites on the internet between 1995 and 2018 using the C# programming language. The study offered a critical analysis and useful advice on how to use a web programming language to create a high-quality product. The study considered web developers' experiences and views on PHP, ASP, and web development as well as other comparison criteria such as cost, performance, readability, understanding, maintainability, editing & deployment tools, platform, database, webservers, core-language, synthetic character, webpage structure. The knowledge, application domain, platform being used, and other considerations all play a role in the decision between ASP and PHP. While ASP is more dependable and effective than PHP, PHP is better suited for developers who are more familiar with Microsoft products. Both are appropriate, but C# is a safer language in terms of server-side web application development (Odeh, 2019).

The readability, writability, and dependability of six regularly used programming languages are compared in Ahmed et al. (2021) study: C, C++, Java, Python, JavaScript, and R. The analyzed the competitive advantage and tradeoffs in the different languages use in various applications. A survey was used to collect data, and a theoretical comparison was performed to analyze the value of these criteria and their impact on the decision-making process of selecting a programming language. Ahmed et al. (2021)'s research devised a novel method for comparing the six programming languages by using a metric evaluation system to evaluate each language by using the categories 'Bad,' 'Moderate,' and 'Good.' A poll was also undertaken to validate the appraisal of this metric system, with a group of people answering a series of questions. The most readable and writeable languages were found to be Python and R, with Java having an advantage in terms of reliability (Ahmed et al., 2021). According to a poll, Python is the programming language of choice for beginners and non-programmers, although Java is favored by seasoned programmers due to its dependability. Theoretical analysis and survey findings were complementary, with Python excelling in readability and writability and Java excelling in reliability. Expert programmers might, however, favor the language in which they feel the most at ease.

Adebukola and Kazeem (2014) researched the evolution of web scripting languages from CGI to PHP and ASP.NET . Open source, PHP is a commonly used server-side scripting language that can be integrated into HTML. Microsoft's ASP.NET web development platform enables programmers to create dynamic web apps using compiled languages like VB.NET and C#. Scripting languages are progressing on many fronts, but businesses still have the difficult but important task of comparing options to determine which one best suits their particular requirements (Adebukola & Kazeem, 2014). The focus of the research study was to analyze and contrast the effectiveness of PHP and ASP.NET, the two most widely used dynamic scripting languages for online development. The study noted that WAPT software is used to evaluate PHP and ASP.NET for online application development. They focused on measuring the response time, which is the amount of time it takes for a client to send and receive a request, to assess the performance of online applications created using PHP and ASP.NET. Performance tests were run to determine the average, minimum, and maximum reaction times while the two technologies remained the same during the development of the program. The most significant performance indicator is thought to be response time.

PHP has a faster reaction time under stress and endurance tests, making it a superior choice for online applications (Adebukola & Kazeem, 2014).

For online applications that require lots of data, Node.js is gaining popularity (Brar et al., 2021). Due to its event-driven, non-blocking I/O approach, Node.js is growing in popularity, and Brar et al. (2021)'s study employs objective benchmark testing and accurate user behavior testing to assess its performance. Brar et al. (2021) utilized benchmark tests and scenario tests to evaluate the performance of Node.js, Python-Web, and PHP. The test results provide some insightful enforcement data, demonstrating that Node.js can handle far more requests in a certain period than PHP and Python-Web can. Three key tests were run in Brar et al. (2021)'s experiment to gauge how well Node.js, Python Web, and PHP performed. Node.js fared better than Python Web and PHP, according to the results, in terms of average requests per second and latency per request. Python-Web, Node.js, and PHP are mature frameworks for large-scale websites (Brar et al., 2021).

Challapalli et al. (2021) examined the process of creating websites in the past and present, the evolution of content delivery over time, the uses of websites, websites for mobile devices, and a performance comparison of the two most popular web backend development languages, namely Node.js and Python. The study found that Node JS outpaced Python at processing requests, processing about 250 times as many requests as Python in a 30-second period. As the number of users increased, Node JS's requests per second increased rapidly, whereas Python's increase was steady and gradual. The failure rates in both situations were 0%. Python's average response time was roughly 2040ms, compared to 7ms for Node JS. While Python's latency climbed dramatically as the number of users increased, plateauing at 14000ms near the end of the test, with an average delay of 7187ms, Node JS had an average latency of 1.04ms (Challapalli et al., 2021).
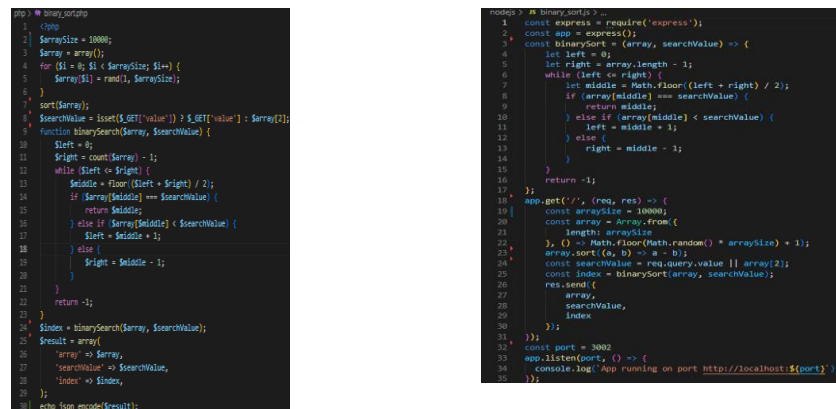
Methodology

This study compares the efficiency of two backend programming languages and technologies, PHP (old) and Node.js (new) using a four types of sorting algorithms; binary sort, bubble sort, quick sort and heap algorithm. The research goal is to identify the most optimal backend language or scripting technology that can provide a maximum performance with regards to latency which is defined as time taken for a request to reach the server and receive a response.

Sorting Algorithms

Sorting algorithms are procedures for sorting items according to a specific order. Unsorted items are reordered based on a specific criterion, such as alphabetical or numerical. Sorting algorithms are classified based on their space and time complexities, stability, comparison, or non-comparison-based nature. The following is a quick description of the four algorithms used in this study.

Binary Sort is a simple sorting algorithm that works by comparing each element with the elements that precede it and swapping them if they are in the wrong order. This algorithm uses a divide-and-conquer style whereby it divides the array into two equal halves, sorts each half individually, and merges them back together. Elements in the left and right sub-arrays are compared by the merge operation, which then merges them into sorted order. Until the full array is sorted, this procedure is repeated. . If the array is in a sorted order, binary sort has an $O(n2)$ worst-case time complexity and a best-case time complexity of $O(n \log n)$. Binary sort is advantageous in situations when the input data is mostly sorted or for tiny lists.
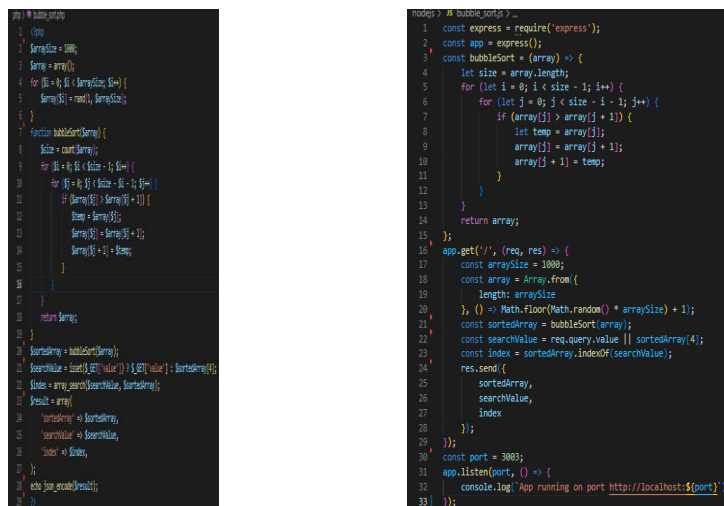
Figure 3: Binary Sort in PHP (left frame) & Node.js (right frame)

Bubble Sort is an algorithm repeatedly swaps adjacent elements in a list until the entire list is sorted.. This sorting algorithm analyses adjacent elements and swaps them if they are in the wrong order as it iteratively moves through the list to be sorted. Until the list is sorted, this trip through the list is repeated. The smaller components "bubble" is passed to the top of the list with each run, hence this technique is known as bubble sort. The worst-case and average time complexity of bubble sort, where n is the total number of elements to be sorted, is O(n2). It has a worst-case time complexity of $O(n^2)$ and a best-case time complexity of O(n) when the input data is already sorted. Bubble sort is not very efficient and is generally only used for small lists

Figure 4: Bubble Sort in PHP (left frame) & Node.js (right frame)

Quick Sort is a divide-and-conquer algorithm that works by selecting a "pivot" element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. This sorting technique divides an array or list of elements into two sub-arrays, one of which contains elements less than a pivot value and the other of which contains elements greater than or equal to the pivot value. The sub-arrays are then sorted recursively. It has a worst-case time complexity of $O(n^2)$ but typically performs much better, with an average time complexity of O(n log n). Quick sort is widely used and is often the preferred algorithm for large data sets.
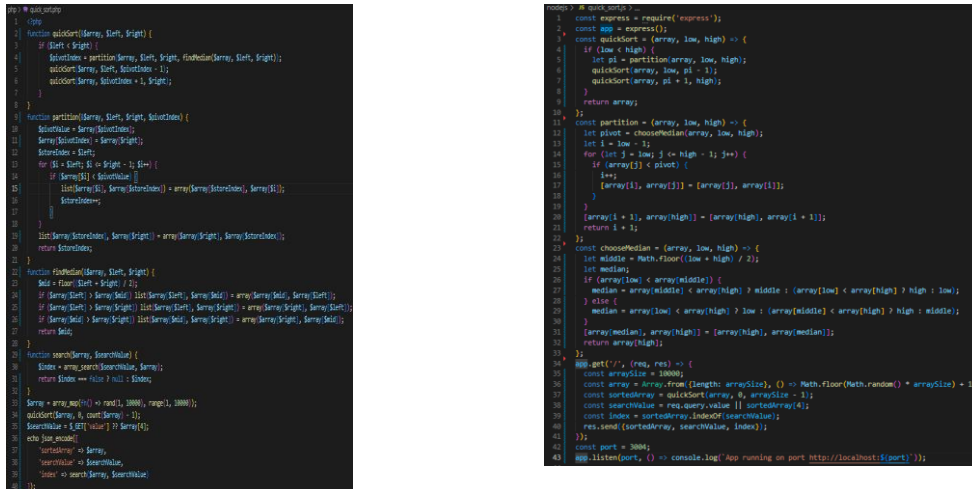
Figure 5: Quick Sort in PHP (left frame) & Node.js (right frame)

Heap Algorithm is a recursive algorithm that generates all possible permutations of a given set. It works by swapping elements in the set and recursively generating permutations of the remaining elements. The for loop iterates over each element of the array, and for each element it calls heapPermutation with a reduced array size of $size-1. The swap operation is performed depending on whether $size is odd or even. If $size is odd, the first element of the array is swapped with the last element, otherwise the $ith element is swapped with the last element. It has a worst-case time complexity of O(n!), where n is the number of elements in the list. Heap algorithm for permutation is useful for cases where all possible permutations of a list are required.
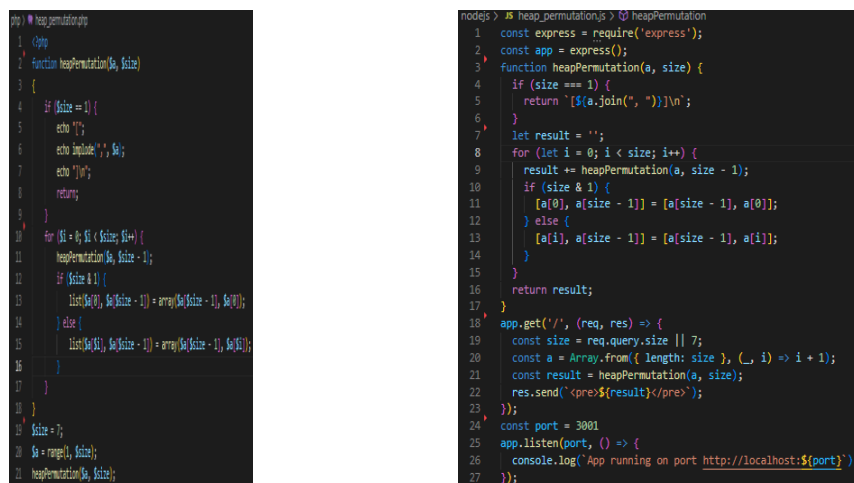
Figure 6: Heap Algorithm in PHP (left frame) & Node.js (right frame)

Algorithmic Complexity

A measure of an algorithm's effectiveness or efficiency is its algorithmic complexity. It is typically stated in terms of the number of steps needed to run the algorithm in relation to the size of the input. Time complexity and spatial complexity are the two most popular metrics for algorithmic complexity. Time complexity measures the number of computational steps required to execute an algorithm as a function of the size of the input data. The most common notations used to express time complexity are O, Ω, and Θ.

Big O notation is the most used notation to describe time complexity, and it represents an upper bound on the number of steps required by the algorithm. Ω notation represents a lower bound on the number of steps, and Θ notation represents an average or tight bound on the number of steps. Space complexity, on the other hand, measures the amount of memory required by an algorithm to execute as a function of the size of the input data.

The analysis of algorithmic complexity is important because it helps us understand how an algorithm will perform on different input sizes and allows us to make informed decisions about which algorithm to use for a particular task. In general, we prefer algorithms with lower time and space complexity because they are more efficient and faster. However, the choice of algorithm also depends on other factors, such as the problem domain, the input size, and the available resources.

Table 1: Time Complexity Table of Sorting Algorithms

Algorithm / Time Complexity	Best Case	Average Case	Worst Case
Quick Sort	O(n log n)	O(n log n)	O(n2)
Binary Sort	O(1)	O(n log n)	O(n log n)
Bubble Sort	O(n)	O(n2)	O(n2)

Performance Goal, Statement of Research Objectives and Research Questions

Performance Goal:

There is a significant difference between the performance of PHP (old) and Node.js (new) backend scripting technologies.

Research Objectives:

⇒ To compare the performance of PHP and Node.js using binary sort, bubble sort, quick sort, and heap algorithm.
⇒ To determine which algorithm performs better in PHP and Node.js
⇒ To provide recommendations for choosing the appropriate backend scripting technology.

Research Questions:

⇒ RQ 1: Is node.js faster than PHP when it comes to Sorting Algorithms?
⇒ RQ 2: Is node.js is faster than PHP when it comes to heap algorithm?
⇒ RQ 3: In overall, is node.js faster than PHP for backend scripting?

Description of Research Design, Apache JMeter, PHP & Node.js, Sorting Algorithms, and Performance Metrics.

Our design for this research involved comparing the performance of two prominent backend scripting technologies – PHP (old) and Node.js (new) using four sorting algorithms (bubble, binary, quick, heap) to sort data as we exponentially increased the array size, and heap algorithm to generate all possible permutations of the elements of the array. Each of the algorithms were run thirty (30) times for both PHP and Node.js.

Performance metrics were collected using the Apache JMeter software – an open-source load testing tool used to simulate different types of load testing scenarios for web applications. The selection of these algorithms was based on popularity, relevance and effectiveness in data sorting as used in the field of Computer Science. The key performance metric used in this research was latency and the following is a detailed description of the research design.

Apache JMeter & Settings:

⇒ **Ramp Up** time taken by JMeter to start all the virtual users specified in the test plan. It determines the rate at which users are added during the test execution. We used a ramp-up time of 10.
⇒ **Thread** – Thread refers to a single virtual user that simulates user activity on the application being tested. Each thread executes the test script independently of all other threads. We simulated 100 virtual users.

Although, there is no one-size-fits-all answer to what the standard settings for ramp-up and threads (number of simulated users) should be in Apache JMeter. The ideal values depend on various factors such as the size and complexity of the application being tested, the available hardware resources, and the performance goals. However, as a rule of thumb, the ramp-up time should be set to a value that allows a gradual increase in the number of virtual users, instead of an abrupt spike. This helps simulate real-world scenarios where traffic gradually increases over time. A good starting place for ramp-up time could be 5-10 seconds. The number of threads, or virtual users, should also be chosen based on the hardware resources available for the load testing.

In general, it is recommended to start with a low number of threads and gradually increase it to find the optimal number that the system can handle without performance degradation or failures. A starting number could be 50-100 threads, and then gradually increase to several hundred or even thousands depending on the available resources. A ramp-up of 10 seconds is a reasonable choice for a 100 number of threads and a typical test scenario as allows the load to be gradually increased (our array size), giving the server time to warm up and stabilize before reaching the maximum load. It also provides enough time for JMeter to start all the threads and for the server to respond to the initial requests. In Apache JMeter, the Ramp-up period represents the time taken to spin up all the threads in the thread group.

For instance, if we have 100 threads and a ramp-up period of 10 seconds, then JMeter will create a new thread every 100 milliseconds (10,000 milliseconds / 100 threads). In other words, the ramp-up period determines how quickly the threads will be created and started. If the ramp-up period is short, like 1 second, then all 100 threads will be created almost simultaneously, which could overload the server being tested. On the other hand, if the ramp-up period is too long, like 100 seconds, then it will take too long to start all the threads, and the test may not be representative of real-world scenarios. Therefore, the ramp-up period of 10 seconds and 100 threads means that JMeter will create a new thread every 100 milliseconds over the course of 10 seconds, so all threads will be active after the ramp-up period. This setting strikes a balance between quickly starting the threads and not overloading the server with too many requests at once.
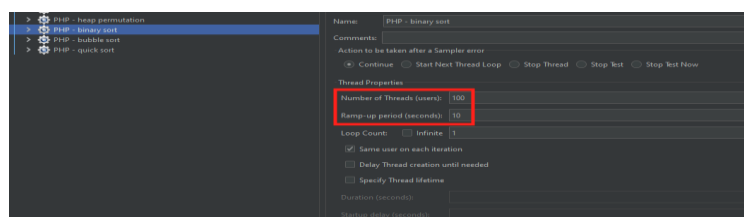

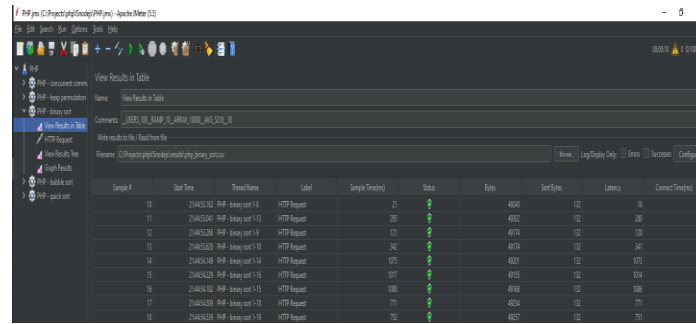**Figure 7a: Apache JMeter showing Number of Threads & Ramp-up period**

Figure 7b: Apache JMeter showing test run results

Performance Metrics:

Latency (milliseconds) - The term "latency" describes how long it takes a server, usually measured in milliseconds, to reply to a client request. It includes any processing time required on the server side as well as the time it takes for a request to go from the client to the server and back again. A server that is quick and responsive has a low latency, while one that is slow and unresponsive has a high latency.

Latency is an important metric for evaluating the performance of any system, especially in the context of computer networks and the internet. Latency refers to the time delay between a request for data and the response to that request. In other words, it is the time it takes for data to travel from its source to its destination.
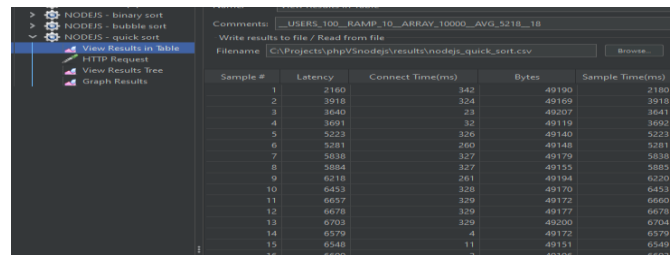
Low latency is important for several reasons. Firstly, it affects the user experience of interactive applications such as online gaming, video conferencing, and real-time communication tools. In these applications, high latency can cause delays and interruptions in communication, leading to a poor user experience. Secondly, latency is critical for high-speed trading applications, where even a small delay can have a significant impact on the outcome of a trade. In this context, low latency can give traders a competitive advantage by allowing them to make decisions and execute trades faster than their competitors.

- Sample Time (milliseconds) - Sample time is the duration of time for which a test run is executed. It can help identify whether the system can provide consistent performance over a period of time, especially when multiple requests are made.

- Connect Time (milliseconds) - Connect time is the time taken to establish a connection between the client and the server. A low connect time is crucial for applications that require frequent connections, such as real-time applications.

- Sent Bytes (bytes) - Bytes refer to the amount of data transferred during a request/response cycle. The amount of data transferred can have a significant impact on the overall performance of a system, especially when handling large volumes of data.

Data Collection Process and Testing Procedures

As Apache JMeter provides several ways to collect data during load testing, our adopted collection process involves the use of TABLE LISTENER – which provides listener to collect data in real-time while the load test is running. With the table listener, we were able to capture a wide range of data including latency, connect times, bytes, sample time etc. Data was collected during load tests of different array sizes. We

Apache JMeter enables us to export this table as a csv file and we calculated latency of each simulated user for 30 runs.

Figure 8: Apache JMeter showing the Table Listener

Procedure:

To ensure that we are only running targeted load test, our testing procedure involves first right-clicking on the intended algorithm, then clicking on 'Start' to begin the test load run.
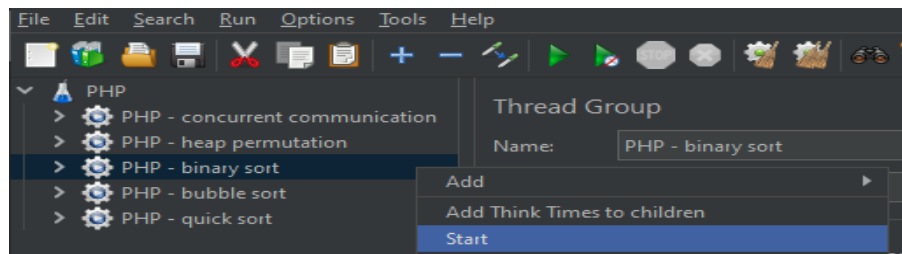
Figure 9: Running targeted algorithm in Apache JMeter

Results

Our research question for this study involves determining if there is difference in performance between PHP (old backend technology) and Node.js (new backend technology) and was evaluated by testing performance with different algorithms. After writing the programs to implement various sorting algorithms, and the heap algorithm to generate all possible permutations, we gathered our data from the performance test we ran on Apache JMeter. Tables 10[a to l] show the t-test results of study which indicated that a significant difference occurs between the performance of PHP and Node.js in all four of the algorithms tested.

Table 10a: T-test Result for Bubble Sort of 100 Elements

T-Test Bubble Sort Array Size 100		
	PHP	*NODE.JS*
Mean	15.35	8.78
Variance	9.59	2.72
Observations	30	30
Pooled Variance	6.16	
df	58	
t Stat	10.26	
P(T<=t) one-tail	0	

Table 10b: T-test Result for Bubble Sort of 100

T-Test Bubble Sort Array Size 1000		
	PHP	*NODE.JS*
Mean	4435.34	837.66
Variance	637917.69	97517.36
Observations	30	30
Pooled Variance	367717.53	
df	58	
t Stat	22.98	
P(T<=t) one-tail	0	

Table 10c: T-test Result for Bubble Sort of 10000 Elements

T-Test Bubble Sort Array Size 10000		
	PHP	*NODE.JS*
Mean	373562.1	33011.73
Variance	2034154626	75620299.33
Observations	30	30
Pooled Variance	1054887463	
df	58	
t Stat	40.61	
P(T<=t) one-tail	0	

Table 10d: T-test Result for Binary Sort of 100 Elements

T-Test Binary Sort Array Size 100		
	PHP	*NODE.JS*
Mean	13.8	6.86
Variance	9.54	2.21
Observations	30	30
Pooled Variance	5.87	
df	58	
t Stat	11.09	
P(T<=t) one-tail	0	

Table 10i: T-test Result for Quick Sort of 10000 Elements

T-Test Quick Sort Array Size 10000		
	PHP	*NODE.JS*
Mean	53.16	13.37
Variance	35.52	2.67
Observations	30	30
Pooled Variance	19.1	
df	58	
t Stat	35.27	
P(T<=t) one-tail	0	

Table 10j: T-test Result for Heap of 5 Elements

T-Test Heap Array Size 5		
	PHP	*NODE.JS*
Mean	8.84	5.78
Variance	3.24	0.39
Observations	30	30
Pooled Variance	1.82	
df	58	
t Stat	8.8	
P(T<=t) one-tail	0	

Table 10k: T-test Result for Heap of 7 Elements

T-Test Heap Array Size 7		
	PHP	*NODE.JS*
Mean	51.45	25.6
Variance	705.67	220.4
Observations	30	30
Pooled Variance	463.04	
df	58	
t Stat	4.65	
P(T<=t) one-tail	0.00001	

Table 10l: T-test Result for Heap of 9 Elements

T-Test Heap Array Size 9		
	PHP	*NODE.JS*
Mean	51627.94	39741.03
Variance	185368533	200767573.4
Observations	30	30
Pooled Variance	193068053	
df	58	
t Stat	3.31	
P(T<=t) one-tail	0.0008	

Table 10e: T-test Result for Bubble Sort of 1000 Elements

T-Test Binary Sort Array Size 1000		
	PHP	*NODE.JS*
Mean	17.83	10.64
Variance	31.14	1.87
Observations	30	30
Pooled Variance	16.5	
df	58	
t Stat	6.85	
P(T<=t) one-tail	0	

Table 10f: T-test Result for Binary Sort of 10000 Elements

T-Test Binary Sort Array Size 10000		
	PHP	*NODE.JS*
Mean	47.55	24.93
Variance	115.75	4.24
Observations	30	30
Pooled Variance	59.99	
df	58	
t Stat	11.31	
P(T<=t) one-tail	0	

Table 10g: T-test Result for Quick Sort of 100 Elements

T-Test Quick Sort Array Size 100		
	PHP	*NODE.JS*
Mean	9.42	6.63
Variance	4.43	0.94
Observations	30	30
Pooled Variance	2.68	
df	58	
t Stat	6.59	
P(T<=t) one-tail	0	

Table 10h: T-test Result for Quick Sort of 1000 Elements

T-Test Quick Sort Array Size 1000		
	PHP	*NODE.JS*
Mean	15.19	6.37
Variance	10.16	0.76
Observations	30	30
Pooled Variance	5.46	
df	58	
t Stat	14.62	
P(T<=t) one-tail	0	

Limitations and Future Research

There are some limitations that should be addressed. First, for the binary, bubble, and quick sort we limited our array to 100, 1000, and 10000. For the Heap algorithm, our array size was limited to 5, 7 and 9. As such our variability in sorting algorithm performance might be replicated for other larger or smaller array sizes. In both PHP and Node.js cases, the implemented algorithms behaved differently as we varied the array size. Secondly, we limited the scope of the study to the performance of sorting and heap algorithms which may not be generalizable to other types of software or applications. Lastly, we limited the performance metric to latency which may not capture all aspects of system performance.

Regarding our completed results and limitations of the study, there are several areas for future research. One possibility would be to consider adding additional sorting algorithms and permutation generation algorithms to determine performance difference in PHP and Node.js. Other backend technologies such as Django, Ruby, and Perl could be considered and compare their performance to that of PHP and Node.js . There is the potential that different hardware configurations could have an impact on performance. Lastly, other factors such as development time, scalability, and maintainability might have an impact on performance and should be investigated.

Conclusion

In conclusion, our research study examined the performance comparison between PHP and Node.js, because they were two of the more prominent backend technologies. We implemented various complex

sorting algorithms and a heap algorithm to generate all possible permutations with varying array sizes. Our tests were run using Apache JMeter and the T-Test statistical method was used in analyzing the collected data. Our results showed that Node.js performed better than PHP in all the tests, with notable statistically significant differences between the two backend scripting technologies. This study contributes by providing evidence that newer technologies such as Node.js does have superior performance compared to the older technologies such as PHP. Future researchers are advised to expand on this study by considering more algorithms and a larger sample size. This study provides valuable information for software engineers, developers, and managers who are seeking the best framework for their web applications.

References

Adebukola, O. M., & Kazeem, O. B. (2014). Performance Comparison of dynamic Web scripting Language: A case Study of PHP and ASP .NET. *International Journal of Scientific & Engineering Research*, 661-668.

Ahmed, Z., Kinjol, F. J., & Ananya, I. J. (2021). *Comparative Analysis of Six Programming Languages Based on Readability, Writability, and Reliability* 2021 24th International Conference on Computer and Information Technology (ICCIT),

Brar, H., Kaur, T., & Rajoria, Y. (2021). The better comparison between PHP, python-web & Node. js. *Int. J. Res. Eng. Sci., 9*(7), 29-37.

Challapalli, S. S. N., Kaushik, P., Suman, S., Shivahare, B. D., Bibhu, V., & Gupta, A. D. (2021). *Web Development and performance comparison of Web Development Technologies in Node. js and Python* 2021 International Conference on Technological Advancements and Innovations (ICTAI),

Chaniotis, I. K., Kyriakou, K.-I. D., & Tselikas, N. D. (2015). Is Node. js a viable option for building modern web applications? A performance evaluation study. *COMPUTING, 97*, 1023-1044.

Lei, K., Ma, Y., & Tan, Z. (2014). *Performance comparison and evaluation of web development technologies in php, python, and node. js* 2014 IEEE 17th international conference on computational science and engineering,

Odeh, A. H. (2019). Analytical and Comparison Study of Main Web Programming Languages–ASP and PHP. *TEM Journal, 8*(4), 1517-1522.

Prayogi, A., Niswar, M., & Rijal, M. (2020). *Design and implementation of REST API for academic information system* IOP Conference Series: Materials Science and Engineering,

Raharjo, W. S. (2014). Performance and Scalability Analysis of Node. js and PHP/Nginx Web Application. *Informatika: Jurnal Teknologi Komputer dan Informatika, 9*(2), 67762.