This thesis is submitted to the School of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.

Performance Evaluation of PHP Frameworks (CakePHP and CodeIgniter) in relation to the Object-Relational Mapping, with respect to Load Testing

Ali Raza Fayyaz & Madiha Munir

Ali Raza Fayyaz
Address: lindblomsvagen Ronneby
E-mail: ali.raza368@gmail.com

Madiha Munir
Address: lindblomsvagen Ronneby
E-mail: mmadihamunir@gmail.com

University advisor(s):
Bengt Carlsson, PHD
School of Computing

School of Computing
Blekinge Institute of Technology
SE – 371 79 Karlskrona
Sweden

Internet : www.bth.se/com
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Context
Information technology is playing an important role in creating innovation in business. Due to increase in demand of information technology, web development has become an important field. PHP is an open source language, which is widely used in web development. PHP is used to develop dynamic web pages and it has the ability to connect with database. PHP has some good features i.e. cross platform compatibility, scalability, efficient execution and is an open source technology. These features make it a good choice for developers to choose PHP for web development. The maintenance of an application becomes difficult and performance being considerably reduced, if PHP is to be used without using its frameworks. To resolve these issues, different frameworks have been introduced by web development communities on the internet. These frameworks are based on Model, View, Controller design pattern. These frameworks provide, different common functionalities and classes in the form of helpers, components, and plug-in to reduce the development time. Due to these features like robustness, scalability, maintainability and performance, these frameworks are mostly used for web development in PHP, with performance being considered the most important factor.

Objectives
The objective of this thesis is to compare and analyze the affect of data abstraction layer (ORM) on the performance of two PHP frameworks. These two frameworks are CakePHP and CodeIgniter. CAKEPHP has built-in support of object-relational mapping (ORM) but CodeIgniter has no built-in support of object-relational mapping (ORM). We considered load testing and stress testing to measure the performance of these two frameworks.

Methods
We performed the experiment to show the performance of CakePHP (with ORM) and CodeIgniter (no ORM) frameworks. We developed two applications in both the PHP frameworks, with the same scope and design and measured the performance of these applications, with respect to load testing, with automated testing tool. The results have been obtained by testing the performance of both the applications on local and live servers.

Results
After analyzing the null hypothesis with T-test, the results showed that there is not much of a difference, as per as the performance of CAKEPHP and CodeIgniter was concerned, with the respect to response time on a live server. CodeIgniter had better performance with respect to throughput on live server. On local server CodeIgniter performed clearly better than CAKEPHP framework with respect to response time and throughput.

Conclusions
After analyzing the results we concluded that CodeIgniter is useful for small and medium sized applications. But CAKEPHP is good for large and enterprise level applications, as in stress conditions the CAKEPHP performed better than CodeIgniter on both local and live environment.

Keywords: Model View and Control, PHP, Page Load Time, Stress Testing,

ACKNOWLEDGMENTS

We would like to thank all those who were involved in our thesis work and contributed with advice and helpful insight.

We would like to thank to our supervisor Bengt Carlsson. He has been very kind and provided us continuous guidelines and support throughout the study. He arranged weekly meetings in the start and gave continuous feedback and comments which was useful to improve the quality of our thesis document.

Special thanks to our parents and families because their prayers and love was a source of inspiration, support and relief for us. Thanks to shayan solutions who provided us live server access to host our web applications and test it.

Ali Raza and Madiha

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION

Nowadays internet has become more popular due to the increase in number of users, for its usage in every business. Internet has become the need of every field of life and our day to day activities are dependent upon the internet. Internet is used in the form of web applications and these applications are used to pay utility bills, social networking, emails, communication, online shopping, online transactions etc. These web applications are developed in different languages like EJB (Enterprise Java Beans), PHP (Hypertext preprocessor), ASP.NET and Ruby on Rails[33].

Due to the rapid growth of information technology and its importance, for almost every kind of business, the web development has become a popular field. Increase in demand of web development has brought the high demand of efficiency, reliability, maintainability and scalability. PHP has features of intuitive, scalability, efficient execution, open source, cross platform compatibility and its supports for SQL, to connect and manipulate data in databases. PHP is widely used for the web application development [1].

PHP is a server side scripting language for web development, which is used for making dynamic and interactive web pages. During the development with simple PHP, business logic is mixed with database queries and presentation markups. Due to mixture of this development mode, the maintenance and scalability of the application becomes difficult [1]. PHP has brought different development frameworks (CAKEPHP, CodeIgniter, Yii, Symfony) to solve this issue [18]. These PHP frameworks are based on Model, View and Controller (MVC) design pattern. MVC separates the application into three different layers, Model, View and Controller. In MVC design pattern, model is data access layer and this layer communicates with a database, which obtains data from databases. View is a presentation layer and this layer is responsible for rendering information on the front end (browser) and Controller is the central part which communicates with both Model and View. The Controller takes the data from the Model, process this data and sends it to the View to render it on to the browser. The Controller contains the business logic [1] [2] [3] [4] [5]. MVC is used to increase the reusability of the code and decrease the coupling of the data description. Using MVC the code maintainability and scalability becomes easy [11].

Due to increasing popularity of web applications, the performance of web application is considered the most important factor. Because the users are no longer passive web consumers but they are active contributors of web, so a poor performance of web application distract the users [34]. Basically PHP performance is based on PHP version in which version application is developed, web server environment and system coding complexity [33]. It means by web-server configurations and adopting better programming practices, we can increase the performance of the web application. But, the main focus of PHP frameworks is to make the development of websites faster and make them perform better. This functionality makes these frameworks popular among the developers, who want to make a fast demo page for a new project.

1.1 Technical Terminologies and Definitions

In this section we will describe technical terms and definitions. This will help the reader to understand the ongoing discussion about the topic.

1.1.1 Server-side Scripting VS. Client-side scripting languages

Server-side scripting is about "programming" the behavior of the server. This is called server-side scripting or server scripting [37].

Server-side script can do:

- Dynamically edit, change or add any content to a Web page
- Respond to user queries or data submitted from HTML forms
- Access any data or databases and return the result to a browser
- Customize a Web page to make it more useful for individual users
- Provide security since your server code cannot be viewed from a browser

Client-side scripting is about "programming" the behavior of the browser. This is called client-side scripting or client scripting. This programming language executed on client-side by the browser. Client-side scripts are usually embedded in HTML files or in separate files. These scripts contain instructions for the browser to perform actions according to user's input. Client-side scripting is used to provide validation, detect visitor's browsers, create cookies etc [36].

Basically client (browser) sends a request to the server for an html file and server returns the file to the browser. If the file contains the server-side script, then initially script is executed on server and server returns the file in the plain html form [37].

1.1.2 PHP

PHP stands for PHP: Hypertext Preprocessor and was born in 1995. PHP is an HTML-embedded scripting language. Most of its syntax is borrowed from C, Java and Perl with a couple of unique PHP-specific features [22]. One of the PHP feature is the ease for developers to connect and manipulate the database. There are many advantages of PHP language, for example performance, scalability, open source, portability etc. [2]. The goal of language is to allow web developers to write dynamically generated web pages quickly. The founder of PHP, Rasmus Lerdorf, used Perl to create PHP, because of massive amount of code needed to code in Perl. The biggest advantage of PHP over Perl is that PHP was designed for scripting for web, while Perl was designed to do a lot more. PHP is also easier to integrate into existing HTML than Perl.

Flow of PHP is shown in the following figure (Fig 1.1):

1. Client sends requests to Script by typing a URL.
2. Script processes the data and sends request to the database directly.
3. Script receives the output from the database and processes the data.
4. Scripts produce the output and forward the data to the client.

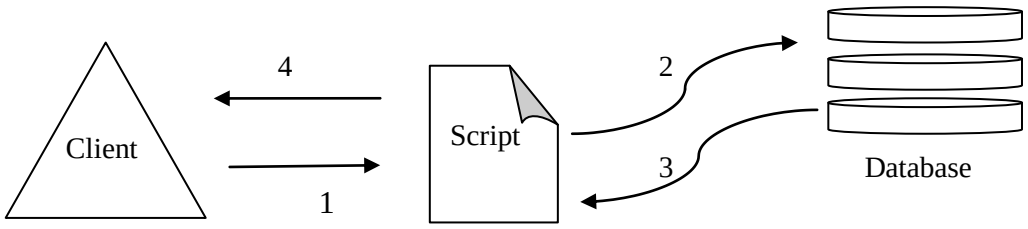

Figure 1.1: Typical Flow of PHP

1.1.3 Content Management Systems

There are also some Content Management Systems (CMSs) available in PHP language. CMSs are software applications which are used for creating, editing, publishing and managing content. These contents are saved in database. CMSs are usually used by new and media organizations, e-commerce systems, broadcasting and film industry, where contents are more important [12]. CMS takes care of almost all the contents of the sites. It separates content from presentation layer. Contents are fetched from database and presented on browser with some styles [6].

1.1.4 Design Patterns

A design pattern is the description of the solution to the common problem. It can be seen as a template to solve a problem which occurs in different situations. A design pattern can be reused in different applications. It is not a code reuse although code can be created from design pattern. Design pattern specify the relationship and interaction between the classes or objects. Design pattern describes the solution for the particular type of problem [39].

Design pattern consists of following three parts.

Problem/Requirement

To create a design pattern, we need to go through analysis design of the application. In this section the requirements are gathered for the problem which we want to solve. This problem is common problem which can occur in more than one application.

Forces

In this section technologies are defined, which helps and guides to create the solution.

Solution

In this section the design is created for design pattern. This section describes how to write the code to solve the problem. This section may contains class diagrams, sequence diagrams or whatever is needed to describe, how to write the code to solve the problem.

1.1.5 Model View Controller (MVC)

Model View Controller (MVC) is the type of design pattern. Most of the PHP frameworks and content management systems are based on MVC design pattern. MVC separates the application into three different layers, Model, View and Controller which makes the application very light. New features can easily be added in MVC based application. The layout of application can be easily changed. Due to modular and separate design, the developers and designer can work simultaneously on the application. It provides the facility to user to make changes in one part without affecting other parts [38].

MVC design pattern can be explained with the following image.

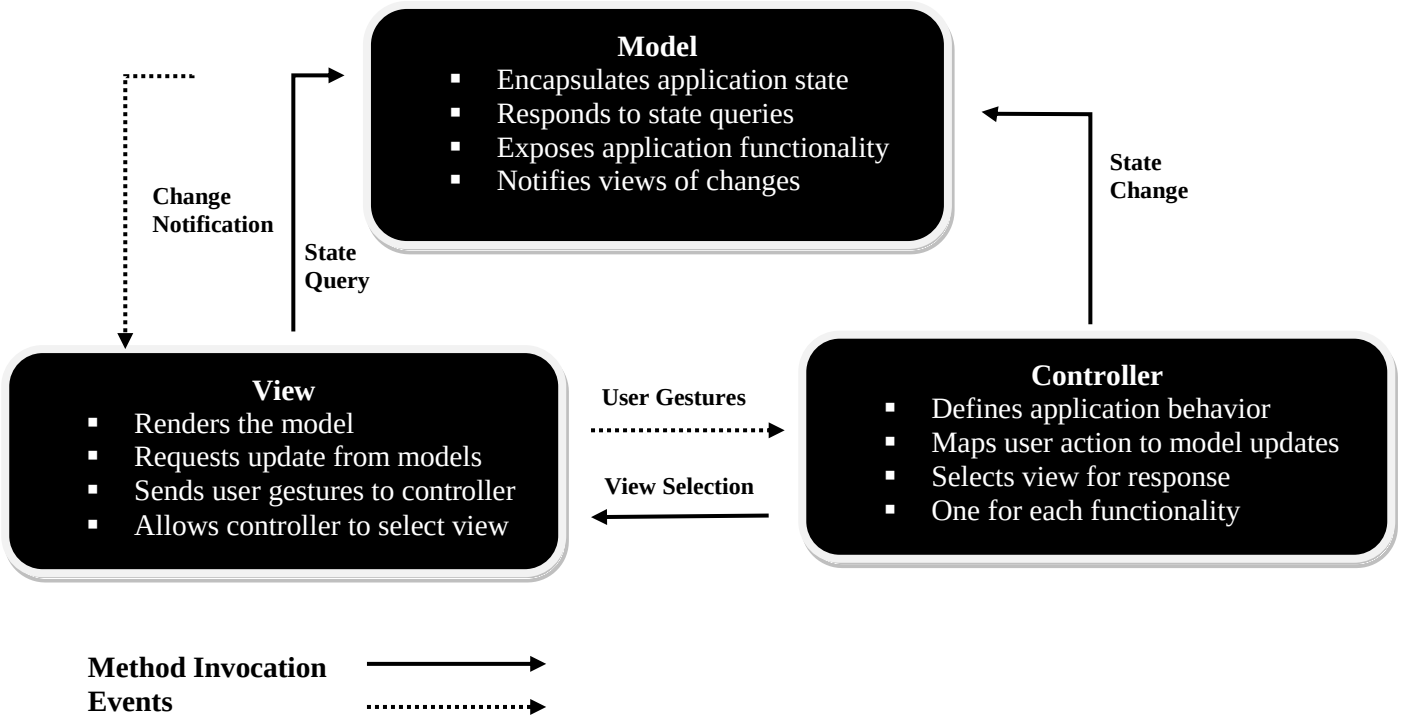

Figure 1.2: MVC

1.1.6 CakePHP

CakePHP is a MVC based framework and was released in 2005, it is written in PHP but inspired by Ruby on Rails. Ruby is a cross-platform interpreted language. Rails is a web development framework which runs on the Ruby programming language. Ruby on rails also uses MVC design pattern. CakePHP uses well known software engineering concepts and software design patterns as convention over configuration, Model View Controller, active records, association data mapping and front controller [23].

CakePHP has some good features i.e. no configurations, clean MVC conventions, code generation and scaffolding features, security for SQL injections preventions, input data validation and form tampering protection which helps to keep applications clean, safe and secure [8]. For database security, CakePHP has a built in database abstraction layer. This layer is called object-relational mapping (ORM), which makes database interaction very easy and secure.

CakePHP enforces the MVC structure for your web application. Basically it effectively separates typical operations into specific areas. Models are used for only database interactions. Views are used only for rendering output to browser. Controllers for all the commands, scripts for input and program flow. In CakePHP the client request is processed as follows.

1. The client sends a page request to the application, either by typing a URL or by clicking a link of some kind. By convention, a typical URL is usually structured like this:http://{Domain}.com/{Application}/{Controller}/{Action}/{Parameter 1, etc.}

2. The dispatcher script parses the URL structure and determines which controller to execute. It also passes parameters along any actions to the controller.

3. The function in the controller may need to handle more data than just the parameters forwarded by the dispatcher. It will send database requests to the model script.

4. The model script determines how to interact with the database using the requests submitted by the controller. It may run queries with the database and do all sorts of handy data-sorting instructions.

5. Database inserts or pulls data from database tables by running SQL queries sent by the model and make data available for model.

6. Once the model has pulled any data from or sent data to the database, it returns its output to the controller.

7. The controller processes the data and outputs to the view file.

8. The view adds any design or display data to the controller output and sends its output to the client's browser.

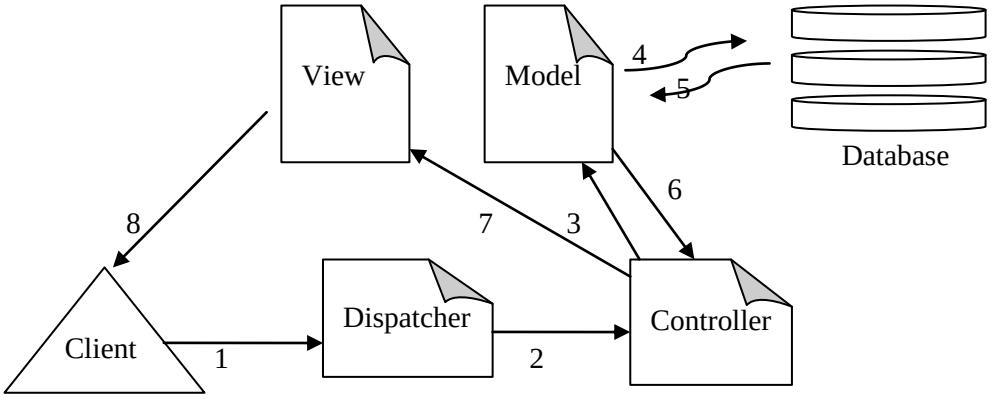

Figure 1.3: How CakePHP makes use of MVC structure

1.1.7 CodeIgniter

CodeIgniter is also MVC based PHP framework, was written by Rick Ellis. CodeIgniter framework has some distinct features i.e. no restrictive coding rules, no need to learn template language, small but comprehensive libraries and thorough documentation. These features are suitable for small and medium sized application. In CodeIgniter, there is no database abstraction layer like object-relational mapping (ORM) in CAKEPHP. Due to absence of ORM in CodeIgniter framework, the database communication becomes complex and insecure [9][21]. In CodeIgniter the client request is processed is as follows.

1. The index.php serves as the front controller, initializing the base resources needed to run CodeIgniter.

2. The Router examines the HTTP request to determine what should be done with it.

3. If a cache file exists, it is sent directly to the browser, bypassing the normal system execution.

4. Security. Before the application controller is loaded, the HTTP request and any user submitted data is filtered for security.

5. The Controller loads the model, core libraries, helpers, and any other resources needed to process the specific request.

6. The finalized View is rendered then sent to the web browser to be seen. If caching is enabled, the view is cached first so that on subsequent requests it can be served.

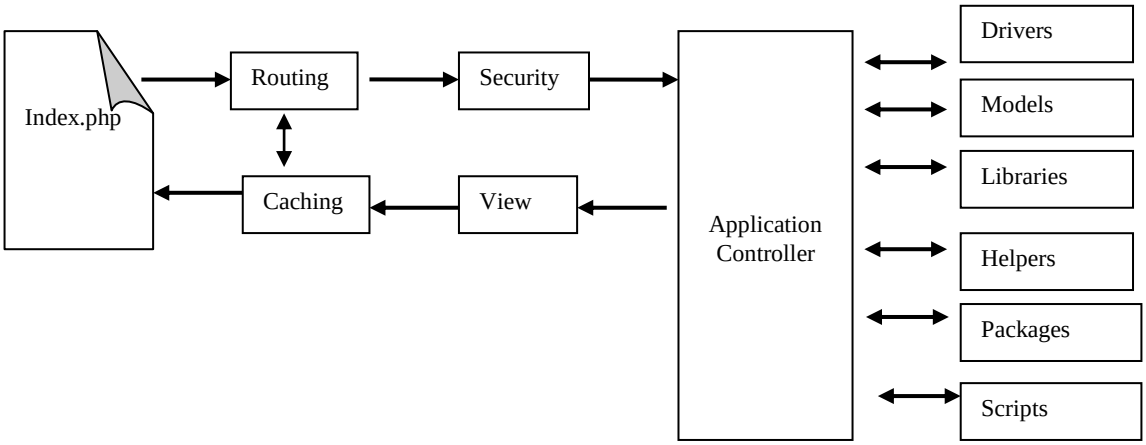

Figure 1.4: How the data flows in the system of CodeIgniter

1.1.8 Object-relational Mapping

Object-relational mapping (ORM) in computer software is a programming technique for converting data between incompatible type systems in object-oriented programming languages [40]. The key feature of ORM is to bind an object to its data in the database by using mapping. Mapping defines the relationship of an object and its properties and behavior with one or more tables in the database, in such a way that object does not need to know anything about the database and database does not need to know anything about the data structured in the application. ORM takes care of the conversion from object to relation and relation to object.

In relational database it is hard to maintain relationship between different tables even for mid-sized PHP applications, particularly multiple level of relationships are involved among database tables. Because complex SQL queries are needed to fetch, insert, update and delete the data. Due to ORM, the database relationships are handled very easily, because ORM helps to define relations among the database tables through association. These associations are defined in the models, according to the relationships among the tables in the database. After defining these associations ORM helps to retrieve, save, update and delete data from different database tables with ease and simplicity. There is no need to write complex SQL queries to join tables. We just need to call the main table; the data from related tables will be fetched automatically. ORM also apply SQL injections in SQL queries and protect database from string attacks. ORM makes application communication with database very easy and simple. It reduces the development time and makes code clean and simple. There are some benefits of using ORM in the framework.

1) ORM increases the overall productivity of the application. It automatically generates the code for the data access depending upon the data model defined

in the application. It automatically applies the security for data access. It significantly reduces the development time and the code to access database.

2) The use of ORM increases the code reusability. The code written for data access for one application can be reused to other application.

3) ORM increases the application maintainability, because the code generated by ORM for data access need not to be tested. We just need to test that code is working according to our requirements. It is presumed to be stable and well tested.

Beside the above benefits of the ORM in the frameworks, there are some drawbacks of it. The data access code generated by ORM is more complex than simple SQL queries, because ORM is designed to handle different data usage scenarios and apply security on the code. It automatically generates complex un-necessary code for some scenarios, which can affect the performance the application. Because the execution of complex code takes more time, which can affect the performance for some scenarios. The frameworks without ORM, has to write complex join queries to fetch data from different related tables and data access code is protected manually, which increases the code size and complexity. It is difficult to maintain application without using ORM. Beside these drawbacks, the application does not need to execute un-necessary extra code to generate code for data access.

2 BACKGROUND AND RELATED WORK

As mentioned above, the performance of the web application is considered very important. Similar work has been done to measure the performance of web applications in different studies.

S.Kaur el al. [16] presented methodology for attaining high availability to the demands of the web clients. In order to improve in response time of web services, during peak hours dynamic allocation of host nodes has been used in this research work. To achieve this objective, LAMP platform is used which are Linux, Apache, MySQL, and PHP. LAMP is used to increase the quality of product by using open source software. In this paper homogenous node has been considered for simulation environment.

Maria del Pilar Salas Zarate el al. [14] discussed the best practices in different frameworks like JSF, Ruby on Rails, Struts, CakePHP, Lift. CakePHP also showed the same performance like other frameworks but Lift showed best results for web development. Analysis presented in this paper revealed that Lift is a new and powerful Framework. And Lift offers more features for developing Web applications than others frameworks. With the usage of these best practices, Web applications were developed in an interactive, intuitive and secure way, improving the development effort and reducing the development time

Raúl Peña-Ortiz el al. [35] measured the performance of the web server. They explored that how web server's performance changes with the dynamic workload on web application. They developed typical e-commerce website and reproduced different user's dynamic behaviors on website. They calculated web server performance using two different parameters i.e. response time and CPU utilization and compared it with traditional workload approaches. The results showed that dynamic workload put more stress on the web server rather than the using traditional workload. Results showed that CPU utilization increased 20% and response time decreased 40% to 50%, when different user's dynamic behaviors have reproduced on the website.

Pushpendra Kumar Singh el al. [33] has created web application i.e. online books mart by using the frameworks of PHP 5.3.0 and ASP.NET 3.5. They calculated and analyzed the web application performance by performing different automated test cases and concluded that ASP.NET performed slightly better than PHP in small scale application and there were mixed results of performance between PHP and ASP.NET in large scale applications. PHP showed better performance in database access.

Savan K Patel el al. [6] has compared the performance of different PHP content management systems (Joomla. Drupal and WordPress). To evaluate the performance of these Content Management Systems (CMS) pages had been hosted on local and live server. From the results of local server he concluded that Drupal is the best choice for informative websites, Joomla is good for fast response and multiple objects and WordPress is good for caching pages. In live server WordPress performed best in all areas but Joomla performed better after caching the page and Drupal performed better for informative websites.

In these studies the performance of PHP content management systems, web server performance by producing dynamic workload on web site developed in PHP and the performance of PHP website, has been compared with ASP.NET website in general. In these studies, performance has been measured on the basis of the response time of web page and page load time, but so far the work relating to PHP framework's performance, has not been done as of yet. There are different PHP MVC based frameworks ,that are available but CAKEPHP and CodeIgniter frameworks are more widely used, for web application

development, due to some notable features [18][19]. We want to measure the performance of these two PHP frameworks by considering two performance parameters i.e. load and stress testing. We will also investigate that how built-in support of data abstraction layer (ORM) in CAKEPHP affect its performance. We did not find any scientific paper related to the performance of CAKEPHP and CodeIgniter. But some web pages have also evaluated the performance of both frameworks (CakePHP and CodeIgniter) and draw some conclusions regarding the performance and choice of frameworks (CakePHP and CodeIgniter). But they did not provide any information regarding their research methodology. The data is also not available on these web pages regarding conclusions [18] [19] [20].

PHP frameworks are using in industry as well e.g. Viaplay AB (viaplay.se) was using Drupal PHP content management systems (CMS) for their product. But soon they realized that drupal is not suitable for their business as their business is growing and there are a lot of users are coming on their product (viaplay.se) so they decided to change the technology. They did some research on different PHP frameworks and they found Yii PHP framework suitable for their business. The online system of Linasmatkasse (linasmatkasse.se) was developed in simple customize PHP. After growing their business they realized that they should developed their system in PHP framework for performance purposes and they decided to choose symfony framework for their business after some research. So we decided to study PHP frameworks and evaluated the performance of two PHP frameworks (CAKEPHP and CodeIgniter) with the respect to load and stress testing because these frameworks are widely used in the market. We will calculate performance of both frameworks on local and live server by developing web pages of different complexities in both frameworks to make results more realistic.

2.1 Purpose

The purpose of this thesis is to evaluate the performance of two PHP frameworks i.e. CakePHP and CodeIgniter with respect to load testing, in order to know which framework performs better than the other.

2.2 Problem Domain

This is an era of information technology and the usage of web is increasing in the form different web applications which facilitate us in our daily lives. Due to increase in usage of the web, the users are also very active contributors to the web ,so that is why the performance of the website has become important. The poor performance of a website will detract the users. To keep users interest on the website it is important to load the web page quickly [34].

When a web developer decides to develop a website, in a PHP framework, where the performance of the website is critical, a need arises for the developer to choose suitable PHP framework, which is time consuming task. There are different PHP frameworks available for the development. We want to calculate and compare, the performance of two PHP frameworks i.e. CAKEPHP and CodeIgniter, which will help small and medium sized companies, to choose the PHP framework for performance critical web applications. We will calculate and compare the performance of these two PHP frameworks by considering the performance parameters i.e. load and stress testing.

2.3 Aims and Objectives

Aim of this research is to evaluate the performance of PHP frameworks, with respect to load testing and investigate, how the presence of object-relational mapping (ORM) affect the performance of PHP frameworks. We chose CAKEPHP and CodeIgniter frameworks for

experiment, because CAKEPHP has built-in implementation of ORM but CodeIgniter has no built-in implementation of ORM.
The following are the objectives of the research that are fulfilled to achieve the aim:

- To investigate the effect of object-relational mapping (ORM) on PHP framework's performance.
- To evaluate the performance of the CodeIgniter (no ORM) framework with respect to load testing.
- To evaluate the performance of the CAKEPHP (with ORM) framework with respect to load testing.
- To analyze and compare the performance of both the frameworks (CakePHP and CodeIgniter).

2.4 Research Questions

Following are our research questions:

RQ. Does a database abstraction layer (Object-relational mapping) affect the performance of PHP frameworks with respect to load testing?

The following two research questions will be the sub questions of our first research question which will more narrow down our research.

RQ1. In relation to object-relational mapping, which framework (CakePHP or CodeIgniter) has better performance with the perspective of load testing on local server?

RQ2. In relation to object-relational mapping, which framework (CakePHP or CodeIgniter) has better performance with the perspective of load testing on live server?

2.5 Research Methodology

In this section we will present research methodology, which we will use to answer our research questions. To answer our main research question RQ, we will answer RQ1 and RQ2. To answer RQ1 and RQ2, we will develop blog application with the same scope and design in both the frameworks (CAKEPHP and CodeIgniter) and measure the performance of both the applications with respect to load testing by running same test case with automated testing tool. We will perform experiment in both local and live server. After analyzing data from local and live server, we will draw conclusion, which framework has better performance on local server and live server.

Our main objective is to investigate the effect of object-relational mapping on the performance of the PHP frameworks. We chose CAKEPHP and CodeIgniter framework, because CodeIgniter has no built-in implementation of ORM but CAKEPHP has built-in implementation of ORM. So by comparing the performance of CAKEPHP and CodeIgniter we will be able to analyze that how ORM affects the performance of the framework. The experiment will be performed by the guidelines of Wohlin [17].

3 THEORETICAL WORK

In this chapter we will explain in detail the relevant terms about performance testing. We will describe types of performance testing and benefits associated with these types of performance testing.

3.1 Performance

Achieving optimal performance of the application requires planning in application design and understanding of best practices. There are different performance requirements which need to determine, whether the application fulfill all those requirements or not. While testing the performance of the application it is assumed that application is functioning stable and robust.

It is important to eliminate the variables as much as possible from the tests. If there are bugs in the code then those bugs can create a performance problem. To check either application is functioning correctly or not, it is important to retest the application. The application must pass its functional tests before testing its performance. If there is any change in the application then unexpected changes can occur in hardware, network traffic, software configuration, and system services.

To measure the performance, it is necessary to maintain the accurate and complete records of test pass. The records should include following steps.

1. The exact system configuration, especially changes from previous test passes
2. Both the raw data and the calculated results from performance monitoring tools

These records not only show either application meets the performance goals but also identify causes of future performance problems. In each test pass it is necessary to run same set of performance tests, it is not possible to determine different results are due to changes in tests rather than changes in the application.

There is different factors impact the results of performance tests, for example, time duration of application (for how long the application runs before the test begins). In addition to these, some more factors which affect the performance and those are firmware design, applications that load at the system startup, memory, I/O components, built in capabilities of system components. To help the system designers and manufacturers, there are tools and information for design and tuning of systems for best performance [25].

3.2 Performance Testing

"Performance testing is defined as the technical investigation done to determine or validate the speed, scalability, and/or stability characteristics of the product under test. Performance-related activities, such as testing and tuning, are concerned with achieving response times, throughput, and resource-utilization levels that meet the performance objectives for the application under test" [27].

3.3 Types of performance testing

There are three types of performance testing.

- Load testing
- Stress testing
- Capacity testing

3.3.1 Load testing

Load testing is used to verify that application can meet desired performance objectives. These performance objectives are often specified in a service level agreement. A load test is used to measure response times, throughput rates, and resource-utilization levels. This is used to identify application's breaking point [27].

3.3.1.1 Load test benefits

- It determines the throughput requires to support the anticipated peak production load.
- It determines the adequacy of hardware environment.
- Evaluates the adequacy of a load balancer.
- Detects concurrency issues.
- Detects functionality errors under load.
- Collects data for scalability and capacity planning purposes.
- Helps to determine number of users the application can handle before performance is compromised.
- It also determines the load for the hardware can handle [27].

3.3.2 Stress testing

The objective of stress testing is to reveal application bugs that becomes visible only under high load conditions. These bugs can include such things as synchronization issues, race conditions, and memory leaks. Stress testing enables to identify application's weak points, and shows how the application behaves under extreme load conditions [27].

3.3.2.1 Stress test benefits

- It determines either the data can be corrupted by making more stress on the system.
- It provides an estimate to show how far target load an application can go before causing failures and errors in addition to slowness.
- It allows establishing the application monitoring the triggers to warn of coming failures.
- It also ensures that security vulnerabilities are not opened by stressful conditions.
- It also determines what sorts of failures are most valuable to plan for [27].

3.3.3 Capacity testing

Capacity testing is conducted in conjunction with capacity planning. Capacity testing is related to future growth to accommodate future load of users. We estimate that how many additional resources (such as processor capacity, memory usage, disk capacity, or network bandwidth) will be necessary to support future usage levels [27].

3.3.3.1 Capacity testing benefits

- Provides information about how workload can be handled to meet business requirements.
- Provides actual data that capacity planners can use to validate or enhance their models and/or predictions.
- Enables you to conduct various tests to compare capacity-planning models and/or predictions.
- Determines the current usage and capacity of the existing system to aid in capacity planning.
- Provides the usage and capacity trends of the existing system to aid in capacity planning [27].

4 EMPIRICAL EVALUATION

In this chapter we present our experiment design with its execution with the guide lines of Wohlin [17]. We will determine independent and dependent variables for our experiment and then we will formulate hypothesis. These hypotheses will be accepted or rejected on the basis of experiment findings. The performance of applications implemented in PHP frameworks are measured with the help of automated test tools. The objective of this study is to compare the performance of the two PHP frameworks i.e. CAKEPHP and CodeIgniter. In the end of this chapter we will present the data from the experiment.

4.1 Experiment Definition

Analyze the CAKEPHP and Condeigniter frameworks to evaluate the performance of both frameworks with respect to the effectiveness of the load testing on local and live server.

4.1.1 Experiment Explanation

To perform the experiment we developed applications in both PHP frameworks i.e. CAKEPHP and CodeIgniter. As we were intended to compare the performance of both frameworks with respect to load and stress testing so we followed the following precautions to develop applications in both frameworks so that we can have good comparison of performance of both frameworks.

- The scope of both applications was the same.
- The HTML and CSS design of both applications were the same.
- We strictly followed the documentation in the development of applications in both frameworks i.e. CAKEPHP [8] and CodeIgniter [9] so that we can use the power and best practices of both frameworks.
- We tested applications on the same local and live environments.
- We tested both applications with the same automated testing tool.

To compare two frameworks it is necessary that applications developed in both frameworks should have the same scope, same design and all the best practices and guidelines of the respective frameworks should be followed which are necessary to increase the performance of the application. So that's why we developed blog application in both frameworks. Application has the following functionalities for user.

- User can login and logout
- Visitor user can view different posts and comments on the posts
- Visitor and logged in User can perform textual search
- Every logged in user has its own admin area
- Every logged in user can add posts and comment to the posts
- Every logged in user can delete its posts from the admin area.

We used MySQL database to store application data. We used HTML and CSS for design purposes. We deployed both applications in local and live server. We used laptop as our local environment and in local machine we installed wamp (Windows, Apache, MySQL and PHP) package and deployed both applications in wamp. To deploy both applications on live environment we bought web hosting with all the necessary tools installed on live environment. We also bought domain name (shayansolutions.com) to access our application on live server through internet. We created two sub domains i.e. http://cake.shayansolutions.com/,

http://CodeIgniter.shayansolutions.com/ to deploy CAKEPHP and CodeIgniter applications respectively. Both blog applications look same.

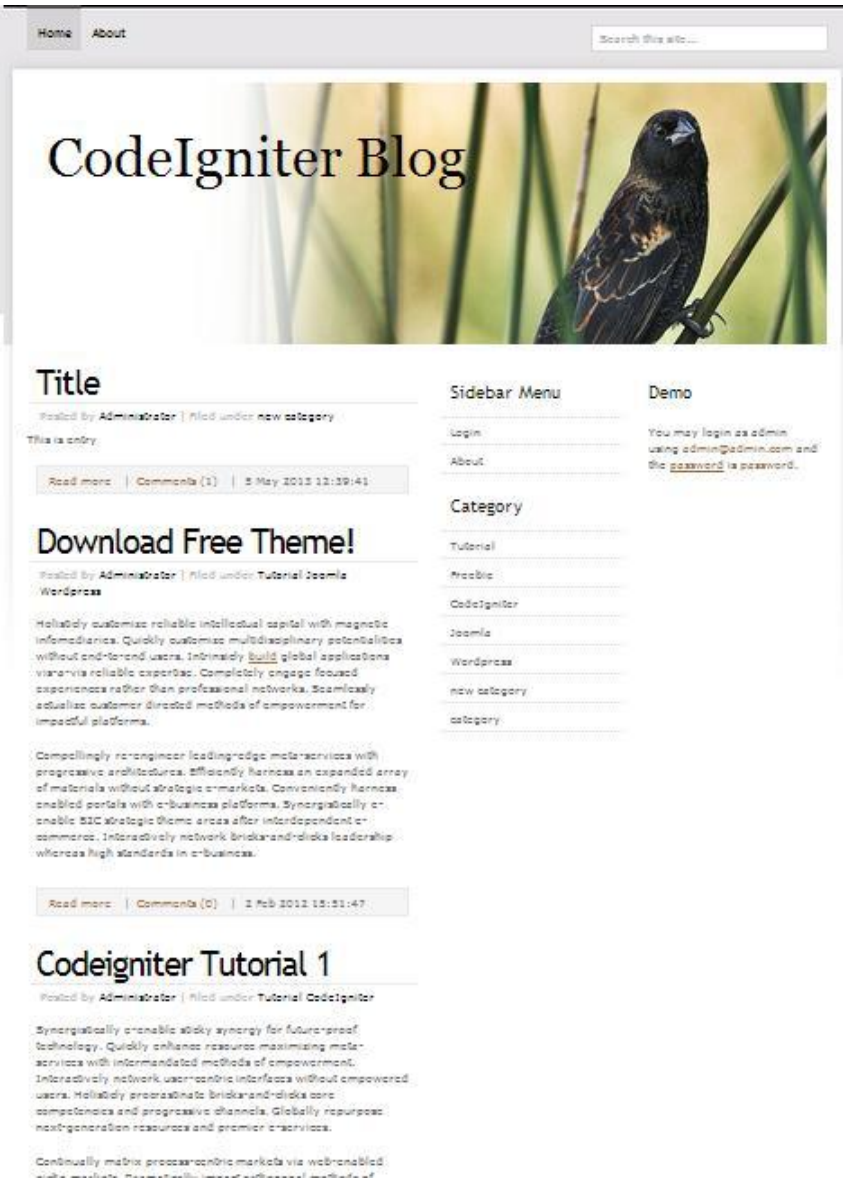

Figure 4.1: CodeIgniter Blog

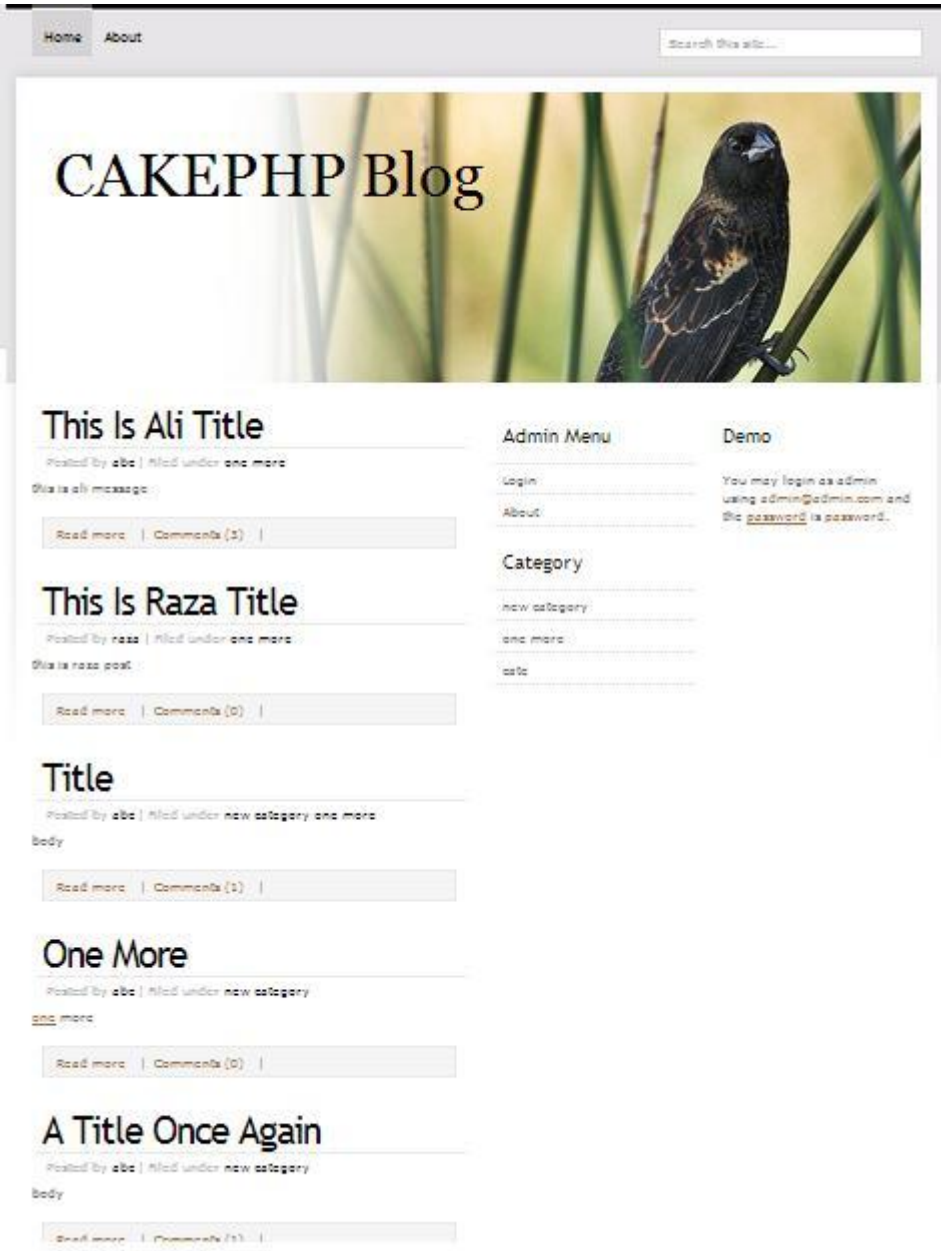

Figure 4.2: CakePHP Blog

After developing and deploying both the blog applications on local and live servers the next step was to measure the performance of these both blog applications. As we were intended to measure the performance of these both applications with respect to load and stress testing so we decides to do the load testing using automatic test tool i.e. JMeter.

4.1.1.1 JMeter

JMeter is an open source software designed to measure the performance of the web application and it is widely used in Software application developer community to measure the performance. JMeter is used to test the performance of both static and dynamic resources. It is used to simulate heavy load on a server, network or object to test its strength and overall performance under different load types. It can also be used to make analysis of performance graphically. It has the ability to test the performance of the web application with respect to load testing. It can be used to load and performance test for different server types i.e. Web-HTTP, HTTPS, Database via JDBC, LDAP etc. It provides the functionality

to choose different load statistics with pluggable timers. Jmeter is not a web browser but for web services and remote services it looks like web browser. JMeter does not perform all the functions like normal browser, for example it does not execute javascript found in the HTML page. It also does not render HTML page like normal browser but response can be seen as HTML but timings are not included in the samples. To perform load testing in Jmeter, first we need to design the test plan. Test plan describes which functionalities of the application, for which we want to perform load test. JMeter has some feature elements which we use to create a test plan in JMeter.

4.1.1.1.1 Building Test Plan

In JMeter test plan describes the series of steps that JMeter will execute when test plan is run. A test plan consists of different elements and these elements are Thread Groups, logic controllers, sample generating controllers, listeners, timers, assertions, and configuration elements.

4.1.1.1.2 Thread Group

Thread group is the beginning point of every test plan. All the controllers are samplers resides under thread group. Listeners reside directly under test plan. Thread group basically controls the number of thread that JMeter will use to execute the test. Thread group provides three options to set for running the test.

- Set the Number of threads
- Set the Ramp up time period
- Set the Number of times to execute the test for each thread

Every thread will execute test plan independent to the other threads. Multiple threads are used in load test to simulate concurrent connections to the server application.

Ramp-up time tells the JMeter how long to delay between starting each thread.

OR

The ramp-up period tells JMeter how long to take to "ramp-up" to the full number of threads chosen. Ramp-up time should be long enough that there should be small work load of the threads at the start of the test on the server.

Example: If 10 threads are used, and the ramp-up period is 100 seconds, then JMeter will take 100 seconds to get all 10 threads up and running. Each thread will start 10 (100/10) seconds after the previous thread was begun.

4.1.1.1.3 HTTP Request Sampler

This HTTP request sampler tells JMeter to send HTTP request to server and wait for the response. This sampler is used to send multiple HTTP request to the same server.

4.1.1.1.4 Recording Controller:

Recording controller is used to record the script of all the functionalities which we want to test. We perform the functionalities in the test plan manually in the browser and Jmeter record those functionalities. In other words the recording controller record all the test plan and we run this recorded test plan for different number of thread and different ram-up times.

4.1.1.1.5 Listeners

Listeners are used to access the information which JMeter gathers about the test plan. This information includes different statistics i.e. Number of threads, Response time, Throughput

etc. These statistics about the test plan are gathered in the csv files or in the form of graph. We used Summary report and aggregate graphs listeners to gather information.

The following figure 4.3 shows the test plan and its different elements discussed above.

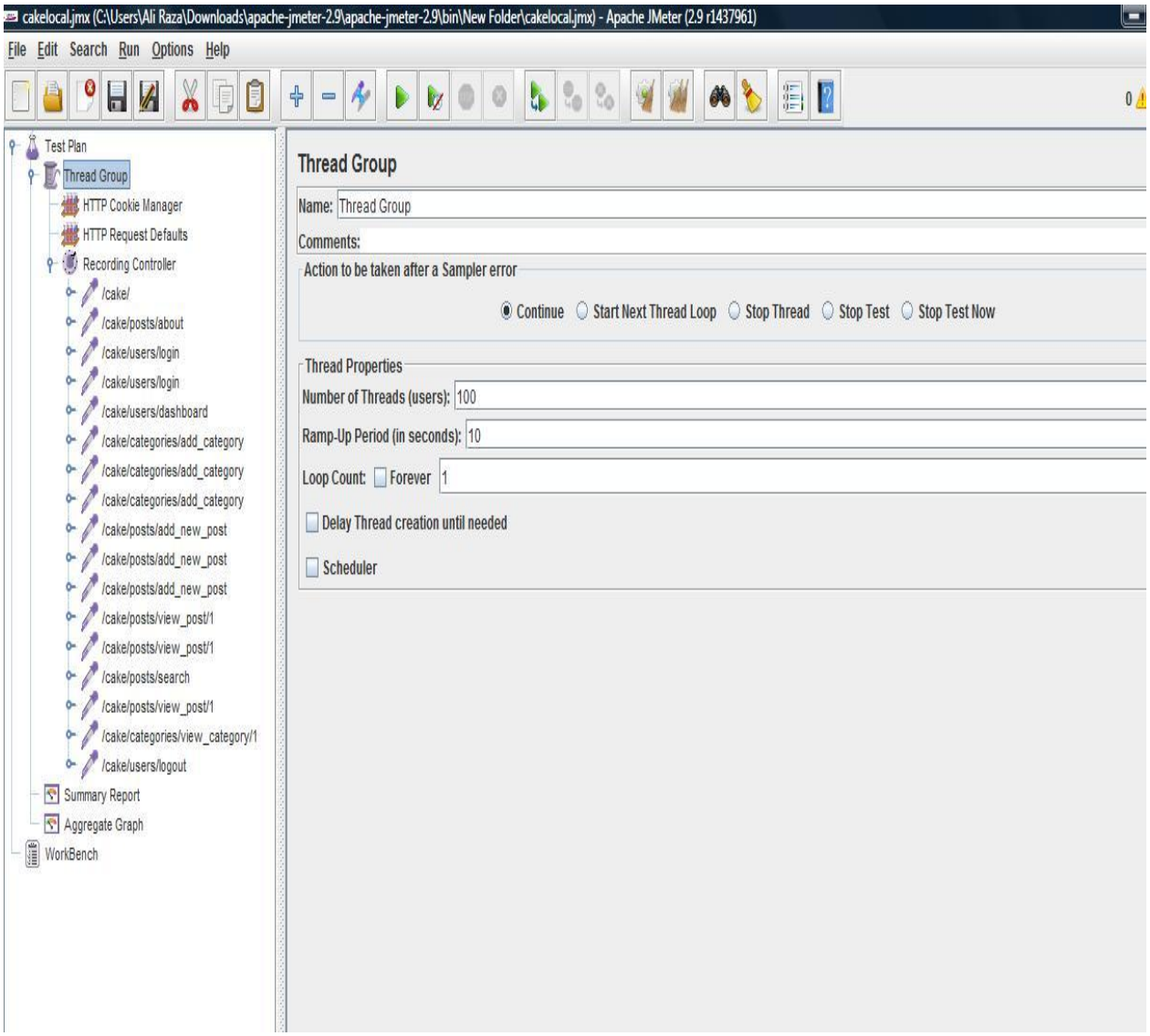

Figure 4.3: Test Plan in JMETER

In the above figure the test plan is shown. In the left window pane the test plan, thread group, HTTP request sampler, recording controller and listeners (Summary report and Aggregate graphs) are showing. Under the recording controller all the recording of the script can be seen. Recording is available in the form of application paths of the functionalities which we performed during recording. In the right window pane three important fields number of users, ramp-up time and loop count. We can specify number of threads (users) for which we want to test application with the test plan recorded in the left window pane under recording controller. The ramp-up time will specify the time for which all the threads should be up and running.

We tested both applications in jMeter with load testing on both local and live server. We created the test plan in jmeter by using all the important functionalities of the applications. The test plan was same for both the applications. We tested both applications for load testing

by increasing load (users). We started testing with 100 users and keeps increasing number of users to check which application respond better by increasing load on the application. Both the applications were tested with same test plan and same load on local server. In live server we used the same test plan to test the applications but the load was different from the local server because the live server cannot support huge load of users. The following is the test plan which we used to test applications for load testing on local and live server.

Test Plan

1	1 Root
2	1 Login
3	1 Dashboard
4	1 Add new category
5	1 Add new post
6	1 Search
7	3 Visited category
8	3 Visited post
9	1 Comment on the post
10	1 About
11	1 Logout

Table 4.1: Test Plan to test application

The above test plan showing that in which order the user performed different actions on the application. The user performed the following actions.

- User entered in the application
- User performed login
- User visited the dashboard after login
- User added one new category into the database
- User added one new post and assigned category to the post
- User performed text search only once and visited three different categories and posts which came in the result of text search.
- User commented on the post only once
- User visited the about page and at the end user logout

We divided our experiment testing in two parts. In first part we deployed both applications on local server and tested both the applications on local server and in second part we deployed both the applications on live server and tested both the applications on live server.
On local server, we tested application in different iterations. We started iteration from 100 threads (users) and tested both applications up to 1100 thread (users). During testing with 1100 and more threads (users) on local server for both applications the server were producing connection errors. Because apache server allows limited number of socket connections for clients. In local environment apache server can only process 150 clients (threads) simultaneously. We increased the ramp-up time during different iterations to get maximum results. But after 1100 users the server was throwing connection errors on local server for both applications. Testing with 1000 users (threads) the server was running with its maximum capacity and with 1100 users (threads) server was running in stress conditions. On live server we started testing with 100 threads and were only able to test up to 600 threads because live server was sending connection refused error with more than 600 users. But the error rate on live server was very low because the live server had capacity to process more number of users simultaneously. In this way we performed testing on both local and live server for both applications with different number of threads and with different ramp-up times and we gathered results in the form of csv files and graphs. JMeter measures the

response time, throughput and standard deviation to measure the performance of the application.

Throughput

The Throughput shows the amount of throughput on the server during each second of the scenario run. Throughput is measured in kilobytes and represents the amount of data that the users received from the server at any given second.

Response Time

The Response Time is the time interval between the receipt of the end of transmission of an inquiry message and the beginning of the transmission of a response message to the inquiry source.

4.2 Experiment Planning

4.2.1 Hypothesis Formulation

A hypothesis is formulated before the design and execution of the experiment. Hypothesis tells what we want to achieve from the experiment. This hypothesis is accepted or rejected on the basis of data collected from experiment. As we will measure the performance of PHP frameworks (CAKEPHP, CodeIgniter) on both local and live server so we will have null and alternative hypothesis for local and live server separately.

Hypothesis for local server:

1. Null Hypothesis, H0: There is no difference of performance between CAKEPHP and CodeIgniter with respect to load testing (response time, throughput) on local server.

2. Alternative Hypothesis, H1: There is a difference of performance between CAKEPHP and CodeIgniter with respect to load testing (response time, throughput) on local server.

Hypothesis for live server:

1. Null Hypothesis, H0: There is no difference of performance between CAKEPHP and CodeIgniter with respect to load testing (response time, throughput) on live server.

2. Alternative Hypothesis, H1: There is a difference of performance between CAKEPHP and CodeIgniter with respect to load testing (response time, throughput) on live server.

4.2.2 Experiment Variables

Before the experiment design we need to find out independent and dependent variables.

4.2.2.1 Independent Variables

Independent variables are those variables which are controllable and changeable in the experiment. Independent variables affect on dependent variables. In our experiment the **independent variables are Frameworks (CakePHP and CodeIgniter).**

4.2.2.2 Dependent variables

The effect of the independent variables measured in dependent variable. Usually there is one dependent variable and it is derived from the hypothesis. The dependent variable is not directly measureable but this is measured indirectly. In our experiment the **dependent variable is performance (load test i.e. response time and throughput).**

4.2.2.3 Participants/Subjects

As we measured the performance of the PHP frameworks (CAKEPHP and CodeIgniter) with automated test tools. We measured the performance of applications with Jmeter tool which puts the load on the server with virtual users (threads).So these virtual users are participants/subjects who performed the experiment.

4.3 Design Experiment

The objective of the experiment is to draw conclusion about the problem which is going to solve. To draw the conclusion we need to collect data from experiment and apply some statistical analysis on the data. Experiment design is very important to draw conclusion about experiment. Experiment design tells which statistical technique we should apply on data to draw conclusion [17].

4.3.1 General Design Principles

The general design principles are randomization, blocking and balancing. In our study we used all the three principles i.e. randomization, blocking and balancing.

4.3.1.1 Randomization

Randomization is the most important design principles. The randomization is used for the allocation of the objects, subjects and in which order the tests are performed. In our experiment we tested performance of applications with different number of virtual users to randomize the selection of the subjects.

4.3.1.2 Blocking

In the experiment some factors have effect on the response and we do not want to consider that effect on the response so we eliminate that undesired effect of the factor on the response. This design principle is called blocking. In our experiment we measured the performance of PHP frameworks (CAKEPHP and CodeIgniter) so there is chance that performance of one framework will be different on local server than on live server with the same number of virtual users. So that's why we measured the performance in two blocks. In one block we measured the performance in local server and other block we measured performance on live server.

4.3.1.3 Balancing

We used balanced deign principle for our experiment because for every treatment we used the same number of subjects (Virtual users). We tested both frameworks with equal number of users (load) in every iteration.

4.3.2 Experiment Design Type

To determine our experiment design type we need to find out factors and treatments in our experiment. In our experiment we are measuring the performance of the PHP framework (CAKEPHP and CodeIgniter) so frameworks is the factor in our experiment and CakePHP and CodeIgniter are the treatments.

Factor: Framework

Treatment1: CAKEPHP
Treatment2: CodeIgniter

The following table show that how subjects (virtual users) are assigned to the treatments. The following table shows that each treatment is assigned with the same number of virtual users to test the performance.

Subjects (virtual users)	Treatment1	Treatment2
100	CAKEPHP	CodeIgniter
200	CAKEPHP	CodeIgniter
300	CAKEPHP	CodeIgniter
400	CAKEPHP	CodeIgniter
500	CAKEPHP	CodeIgniter

Table 4.2: Treatments to test performance

4.3.3 Experiment Instrumentation

As we performed experiment on local and live server so that's why we had different instrumentation and different server configurations on local and live server.

4.3.3.1 Instrumentation on local Machine

- Processor: Intel(R) Core(TM) 2 Duo CPU T5800 @2.00GHz
- Installed memory: 3.00 GB
- System Type: 32 Bit operating System
- Operating System: Windows 7
- Web Server: Wamp (Windows apache MySQL PHP)
- JMeter for load testing

4.3.3.2 Instrumentation on live Server

- Operating system: Linux
- Server Name: shayansolutions.com
- MySQL Disk Space: 0.24 MB
- LAMP(Linux Apache MySQL PHP)
- Apache version: 2.2.24
- MySQL version: 5.1.70-cll
- PHP version: 5.3.23
- Architecture: i686

4.3.4 Experiment validity evaluation

Validity evaluation concerns about the validity of the results. It consider how the results are valid which are drawn from the experiment. It is important that the results should valid for the population of interest. We are considering three levels of validity about our experiment results.

4.3.4.1 Internal Validity

In the internal validity the relationship between treatment and outcome is observed. In other words we must make sure that treatment causes outcome. In our experiment we repeated the experiment several time so that treatment should causes the outcome. We performed experiment on local and live server so during the experiment we make sure that everything is properly installed and working properly on both local and live environment so that the experiment should be performed properly and results should not be affected. JMeter automatically replicate the test plan and there was some database communication involved in test plan so we also make sure that jmeter is testing applications correctly and inserting the entries in the database accordingly.

4.3.4.2 Conclusion validity

In this validity the relationship between treatment and outcome is concerned to make sure that there is statistical relationship between treatment and outcome. As mentioned above in the experiment design that we have one factor and two treatments in our experiment. So that we choose right statistical test i.e. T-test. We performed the T-test on the dataset collected

from the experiment to draw proper conclusion about the experiment. T-test shows how treatments and outcome are statistically related with each other.

4.3.4.3 External Validity

This validity is concerned about the generalization of the results. The results gathered from our study can generalize our study? We developed only one application in both the PHP frameworks (CAKEPHP and CodeIgniter) and we strictly followed development documentation and the best coding practices of both frameworks respective recommended for respective frameworks so that we can develop applications in both frameworks by using their maximum power. In this way we can be able to generalize our results. We developed medium sized applications in both the frameworks (CAKEPHP and CodeIgniter) which cover all the necessary functionalities which modern website should provide. But there is possibility that these frameworks (CAKEPHP and CodeIgniter) may have different performance with different complexity level. We only measured the performance of the application with respect to load testing. But the performance of these frameworks can measured with other factors for examples which of these two frameworks (CAKEPHP and CodeIgniter) has low development time or which framework provides better reusability etc.

4.3.5 Data Sets

As we measured the performance of both the applications on both local and live server so we gathered data for both applications from local and live server separately with different iterations. We presented datasets for local and live server separately in the appendix. The data sets presented in the appendix have 5 columns.

Sampler_label:
In the label section you will able to see the entire recorded http request, during test run or after test run.

Sample:
Samples denote to the number of http requests executed for a given thread. For example if we have one HTTP request in our test plan and we run it with 100 users, than the number of samples will be 100x1=100. If we have five HTTP requests in our test plan and we run it with 100 users, than the number of samples will be 100x5=500. It means that 500 HTTP requests were executed in by running test plan with 100 users.

Average:
Average is the average response time taken to execute the complete test plan with one or more than one users. This response time is recorded in millisecond.

Std.Deviation:
This shows how many exceptional cases were found which were deviating from the average value of the receiving time. The lesser this value more consistent the time pattern is assumed.

Throughput:
Throughput is measured in kilobytes and represents the amount of data that the users received from the server at any given second.

4.3.5.1 Data Sets on local server

The following table shows that how much ramp-up we used in each test run with different number of users on local server.

Iterations	Number of threads (users)	Ramp up time
1	100	100
2	200	200
3	300	300
4	400	400
5	500	500
6	600	600
7	700	700
8	800	800
9	900	900
10	1000	1000
11	1100	1100

Table 4.3: Datasets on local server

4.3.5.2 Data Sets on live server

The following table shows that how much ramp-up we used in each test run with different number of users on live server.

Iterations	Number of users	Ramp up time
1	100	100
2	200	200
3	300	300
4	400	400
5	500	500
6	600	600

Table 4.4: Datasets for live server

5 RESULTS AND ANALYSIS

In this chapter we will present the data collected during the execution of the experiment in more formatted form. We will present and analyze the results. We will interpret results using descriptive statistics and hypothesis testing.

5.1 Descriptive statistics

We will use descriptive analysis to present and visualize the data in tabular and graphics form. We will present data separately for local and live server using descriptive analysis.

5.1.1 Data analysis on local server

We will present data collected from local server during experiment of both applications developed in both frameworks (CAKEPHP and CodeIgniter). This test plan has run with different number of threads (users) ranging from 100 to 1100 for both applications. We ran test plan starting with 100 users and added 100 users in every successive test plan. Both the applications have been tested up to 1000 threads (users) but after exceeding from 1000 threads (users) the server were sending connection timeout errors. Because on local server apache allow the limited number of clients (threads) to be connected at a time.

The columns in below tables are representing the statistics related to performance of the applications. The columns have the same meaning in the tables below which we described in previous chapter's section 4.3.5 data sets. But the data presented in the below tables is the total data for the whole test plan. In the previous chapter we presented data of every HTTP request included in the whole test plan but in this below table we are showing the sample, average response time, standard deviation and throughput of the whole test plan. In other words the below tables are showing the total sample. total average, total standard deviation and total throughput presented in the previous chapter data sets.

5.1.1.1 CAKEPHP data analysis on local server

The following table shows the data set of the CAKEPHP framework on local server. The table shows the average response time, standard deviation, through put and error percentage of the test plan run with different number of users.

sampler_label	sample	Average (sec)	Stddev (sec)	error%	Throughput (kb/sec)
100 users	1700	11.5	12.2	0	39.9
200 users	3400	36.4	86.2	0	39.0
300 users	5100	55.3	171.0	4	38.3
400 users	6800	44.1	160.1	31	35.5
500 users	8500	45.6	180.4	38	35.3
600 users	10200	41.0	174.3	46	36.7
700 users	11900	36.8	172.1	54	31.5
800 users	13600	36.8	182.0	60	41.2
900 users	15300	27.4	143.5	64	39.7
1000 users	17000	29.4	155.4	64	37.5
1100 users	17085	26.0	143.6	68	36.6

Table 5.1: Results of CakePHP on local server

5.1.1.2 CodeIgniter data analysis on local server

The following table shows the data set of the CodeIgniter framework on local server. The table shows the average response time, standard deviation, through put and error percentage of the test plan run with different number of users.

sampler_label	Sample	Average (sec)	Stddev (sec)	error%	Throughput (kb/sec)
100 users	1700	0.5	0.7	0	161.8
200 users	3400	10.4	21.3	0	127.6
300 users	5100	30.1	88.1	0	118.8
400 users	6800	33.4	113.8	12	128.9
500 users	8500	26.9	102.8	30	126.5
600 users	10200	25.4	108.2	41	120.9
700 users	11900	29.9	133.2	43	121.1
800 users	13600	26.2	121.5	49	126.5
900 users	15300	22.9	112.4	55	128.0
1000 users	17000	21.4	112.9	62	118.6
1100 users	17034	23.7	120.2	56	134.6

Table 5.2: Results of CodeIgniter on local server

5.1.1.3 Cross comparison of response time between CAKEPHP and CodeIgniter

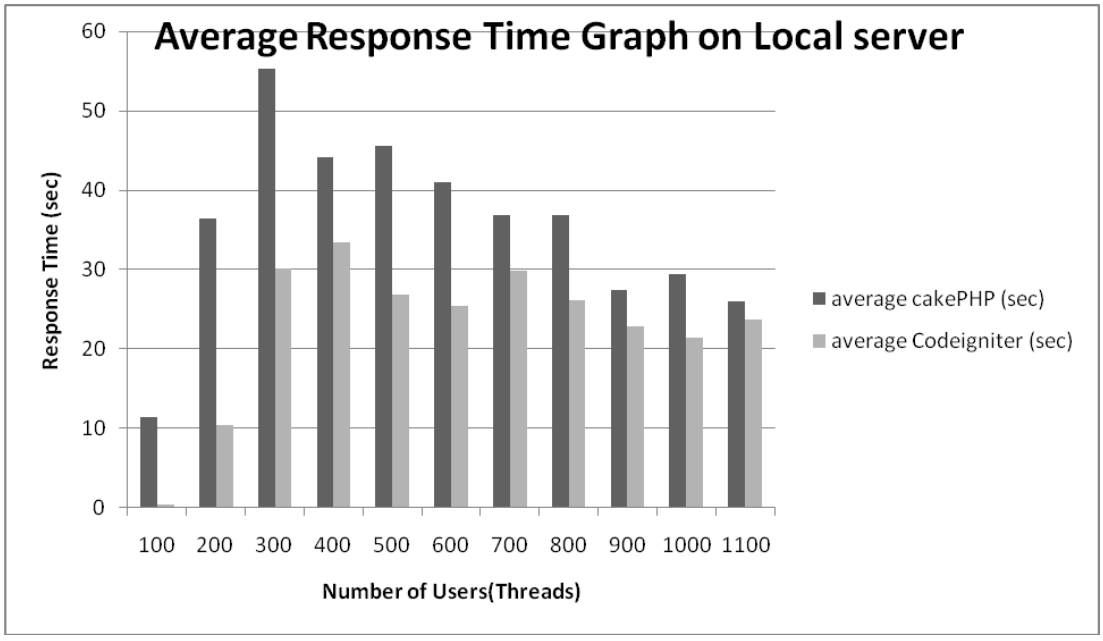

Figure 5.1: Average response time between CakePHP and CodeIgniter on local server

The above graph is showing the comparison of average response time between CAKEPHP and CodeIgniter applications. The response time has been recorded by running test plan with different number of users for both applications. The graph is showing that the response time of both the applications is increasing with the increasing number of users in every test run but after 300 users the response time of both the applications is decreasing but actually the response time should be increased. This decreasing trend in response time on increasing load on applications is due to error percentage in HTTP requests. When test plan is run with more concurrent users, the HTTP requests also increases on the local server and when these HTTP requests cross the limit of the local server's capacity then all those HTTP requests become failed because local server allows limited number of clients to be connected with server. It can be seen in the graph that the response time of CodeIgniter application is better than CAKEPHP application in all the iterations. CodeIgniter performed good in all the load conditions (threads) on local server. CodeIgniter performance was better than CAKEPHP with the perspective of response time. Here in the graph we are showing the results until thousand users, but application breaks at the 1100 users and stress comes on the application on 1100 users. There is a little difference between the 1000 users and 1100 users for the response time.

5.1.1.4 Cross comparison of throughput between CAKEPHP and CodeIgniter

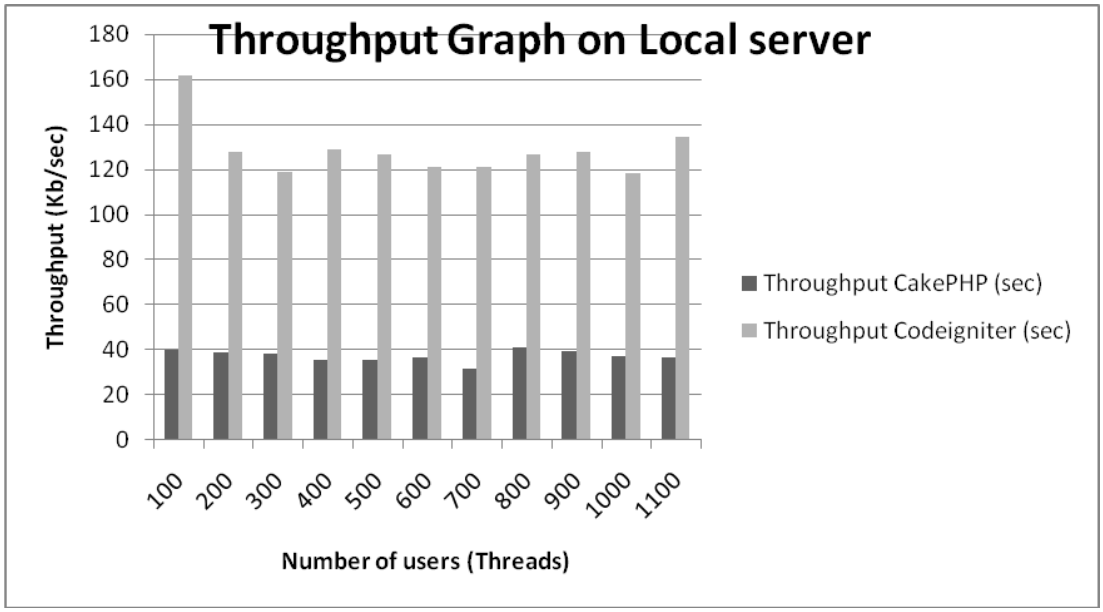

Figure 5.2: Throughput between CakePHP and CodeIgniter on local server

The above graph is showing the comparison of throughputs between CAKEPHP and CodeIgniter. The throughput has been recorded as kilobytes per seconds. As it can be seen in the above graph that throughput of applications is not consistent. This is due to increase in error percentage in every test plan run with increasing number of users. It can be seen that throughput of CodeIgniter application is significantly better than CAKEPHP in all the load conditions.

5.1 Hypothesis Testing

Hypothesis testing is performed on data collected from experiment. As we have hypothesis for different local and live servers so here we are analyzing data of local server so here we test local server hypothesis. Our hypothesis is:

Hypothesis, H0: There is no difference of performance between CAKEPHP and CodeIgniter with respect to load testing (response time, throughput) on local server.

We are intended to measure the performance of the CAKEPHP and CodeIgniter with respect to load testing on live server. After testing the applications on live server the Jmeter provided us the two performance statistics i.e. response time and throughput, so we have two depended variables (response time and throughput) here and we will apply T-tests on response time and throughput data of both applications separately. We calculated response time and throughput of both the applications by applying load on both the application with different number of threads ranging from 100 to 1100 threads. The testing with the load from 100 users to 1000 users showing load testing and testing with the load of 1100 users is showing stress testing. We applied T-test on the response time and throughout of both the applications with load range from 100 to 1000 users, which is the load testing.

5.1.1 T-test on response time of CAKEPHP and CodeIgniter

To test NULL hypothesis we applied T-test on the response time of CAKEPHP and CodeIgniter. The T-test is performed by considering un-equal variances of response time of both the frameworks i.e. CAKEPHP and CodeIgniter. By calculating the mean value of the response time of both frameworks, the mean value of response time (22.71 sec) of CodeIgniter is less than the mean value of response time (36.43 sec) of CAKEPHP. This shows that response time of CodeIgniter is better than CAKEPHP. In other words that CodeIgniter has better performance with perspective of response time. To know more about our experiment data we will analyze the values of t-test.

According to our t-test values, t value 2.78 is greater than t critical values with level of significance 0.01. So we can reject null hypothesis. There is significant difference between the response time values of both the applications. So our null hypothesis is rejected i.e. "There is no difference of performance between CAKEPHP and CodeIgniter with respect to load testing (response time) on local server" and we accept our alternative hypothesis i.e. "There is a difference of performance between CAKEPHP and CodeIgniter with respect to load testing (response time) on live server". So after rejecting null hypothesis we can conclude that there is difference between the performance (response time) of CodeIgniter and CAKEPHP framework. As mean of response time (22.71 sec) of CodeIgniter is less than the mean of response time (36.43 sec) of CAKEPHP. So performance (response time) of the CodeIgniter framework is better than the corresponding response time of CAKEPHP with the response time.

5.1.2 T-test on throughput of CAKEPHP and CodeIgniter

T-test applied on throughput values of CakePHP and CodeIgniter frameworks for 100 to 1000 users show that absolute t-value 22.21 is greater than t Critical values with Level of significance 0.001. There is significance difference between the throughput of CAKEPHP and CodeIgniter. We reject our null hypothesis. From T-test we found that mean value of throughput (127.87 kb/sec) of CodeIgniter is greater than mean value of throughput (37.46 kb/sec) of CAKEPHP. So CodeIgniter has better performance with the perspective of throughput.

5.1.3 Data analysis on live server

We will present data collected from live server during experiment of both applications developed in both frameworks (CAKEPHP and CodeIgniter). This test plan has run with different number of threads (users) ranging from 100 to 600 for both applications. We run test plan starting with 100 users and added 100 users in every successive iteration. Both the applications responded till 500 threads (users) but after exceeding from 500 threads (users) the server were sending connection timeout errors. Because on live server apache allow the limited number of clients (threads) to be connected at a time.

5.1.3.1 CAKEPHP data analysis on live server

The following table shows the data set of the CAKEPHP framework on live server. The table shows the average response time, standard deviation, through put and error percentage of the test plan run with different number of users.

sampler_label	Sample	Average (sec)	Stddev (sec)	Throughput (kb/sec)	error%
100 users	2000	0.9	5.5	128.1	0
200 users	4000	1.3	1.2	190.2	0.075
300 users	6000	1.5	1.3	279.7	0.05
400 users	8000	2.0	2.5	329.6	0.06
500 users	10000	3.3	4.1	285.2	20.1
600 users	11953	5.9	9.6	297.2	6.1

Table 5.3: Results of CakePHP on live server

5.1.3.2 CodeIgniter data analysis on live server

The following table shows the data set of the CodeIgniter framework on live server. The table shows the average response time, standard deviation, through put and error percentage of the test plan run with different number of users.

sampler_label	sample	average (sec)	stddev (sec)	Throughput (kb/sec)	error%
100 users	1900	0.8	0.7	208.4	0
200 users	3800	1.6	1.9	714.7	0
300 users	5700	1.3	1.5	615.2	0
400 users	7600	2.9	5.8	695.4	0.5
500 users	9500	4.8	8	833.7	0.3
600 users	11229	22.4	127.6	195.6	2.4

Table 5.4: Results of CodeIgniter on live server

5.1.3.3 Cross comparison of response time between CAKEPHP and CodeIgniter

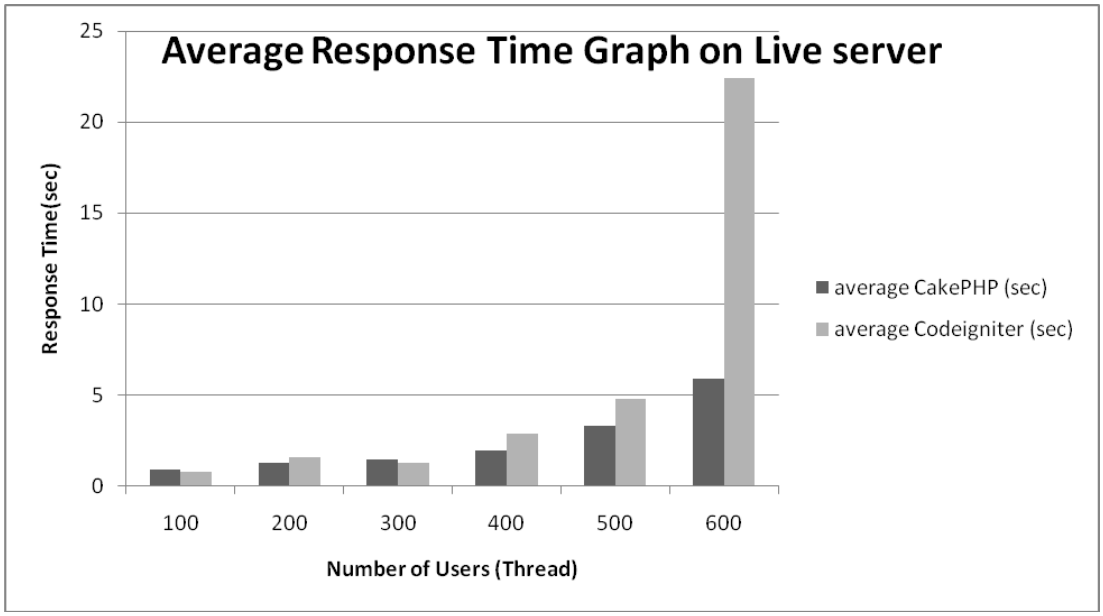

Figure 5.3: Average response Time between CakePHP and CodeIgniter on live server

The above graph is showing the comparison of average response time between CAKEPHP and CodeIgniter applications on live server. The response time has been recorded by running test plan with different number of users for both applications. The graph is showing that the

response time of both the applications is increasing with the increasing number of users in every test run respectively. It can be seen in the graph that the response time of CodeIgniter application is better during the initial test runs with little loads but when load is increased on both the applications then response time of CAKEPHP applications is better. On the extreme stress situation the CAKEPHP performed significantly better than the CodeIgniter.

5.1.3.4 Cross comparison of throughput between CAKEPHP and CodeIgniter

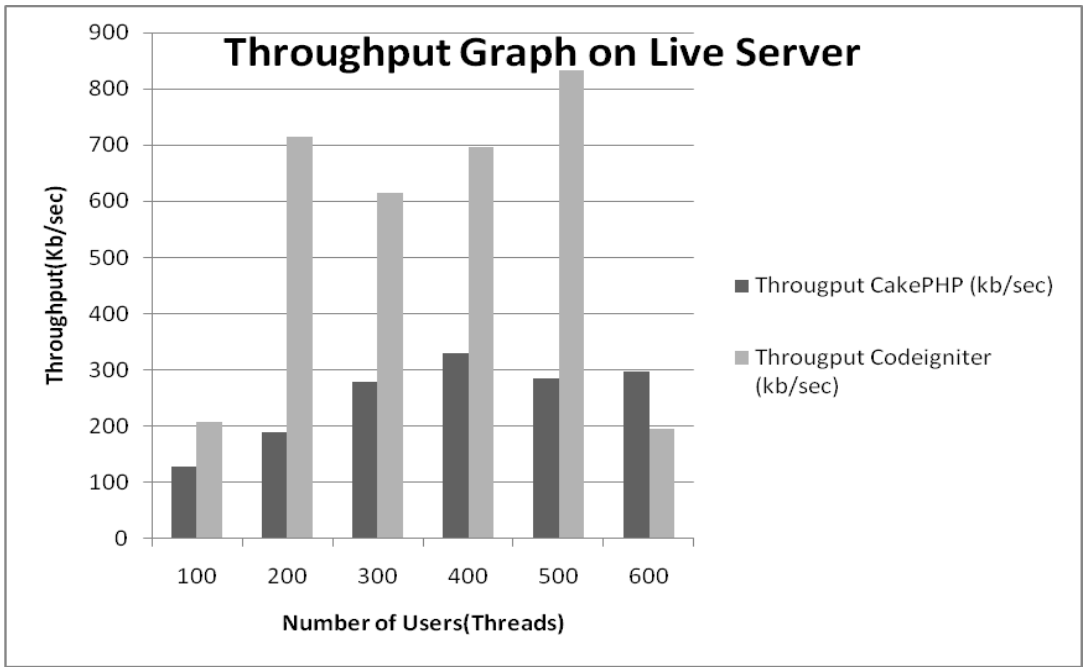

Figure 5.4: Throughput between CakePHP and CodeIgniter on live server

The above graph is showing the comparison of throughputs between CAKEPHP and CodeIgniter on live server. The throughput has been recorded as kilobytes per seconds. As it can be seen in the above graph that throughput of CodeIgniter application is significantly more than CAKEPHP on all the load conditions but on the extreme stress condition the throughput of CAKEPHP is better than the CodeIgniter framework.

5.2 Hypothesis Testing

The hypothesis testing is performed on data collected from experiment. As we have hypothesis for different local and live servers so here we are analyzing data of live server so here we test live server hypothesis. Our hypothesis is:

Hypothesis, H0: There is no difference of performance between CAKEPHP and CodeIgniter with respect to load testing (response time, throughput) on live server.

We are intended to measure the performance of the CAKEPHP and CodeIgniter with respect to load testing on live server. After testing the applications on live server the JMeter provided us the two performance statistics i.e. response time and throughput, so we have two depended variables (response time and throughput) here and we will apply T-tests on response time and throughput data of both applications separately. We calculated response time and throughput of both the applications by applying load on both the application with different number of threads ranging from 100 to 600 threads. The testing with the load from 100 users to 500 users showing load testing and testing with the load of 600 users is showing stress testing. We applied T-test on the response time and throughout values of both the applications with load range from 100 to 500 users, which is the load testing.

5.2.1 T-test on response time of CAKEPHP and CodeIgniter

The T-test applied on the response time of CAKEPHP and CodeIgniter on live server shows that absolute t value i.e. 0.57 is less than t critical value with level significance 0.05. There is no significance difference between response time values of both the frameworks. So we can not reject our Null hypothesis. i.e. "There is no difference of performance between CAKEPHP and CodeIgniter with respect to load testing (response time) on live server.

5.2.2 T-test on throughput of CAKEPHP and CodeIgniter

After applying T-test on throughput values of both the frameworks we found that absolute t-value i.e. 3.27 is greater than t-critical values with the level of significance 0.01. There is significant difference between the throughput values of both the frameworks. So we can reject our Null hypothesis. i.e. "There is no difference of performance between CAKEPHP and CodeIgniter with respect to load testing (throughput) on live server". We accept our alternative hypothesis i.e. "There is a difference of performance between CAKEPHP and CodeIgniter with respect to throughput on live server". T-test table shows that mean value of throughput 242.56 kb/sec of CAKEPHP is less than mean value of throughput 613.48kb/sec of CodeIgniter. So CodeIgniter has better performance with the perspective to throughput.

5.3 Results summary

The following tables are showing the results summary of both the frameworks for local and live servers. The result summary is presented below for normal load conditions. The normal load conditions for local server is the load testing with the range from 100 to 1000 users and normal load conditions for live server is the load testing with range of 100 to 500 users. The table shows that which framework performed better on local and live server. The plus (+) sign is used to show better performance and minus (-) sign is used to show bad relatively bad performance. The zero (0) sign shows that there was not significant difference between the performance of both the application.

Results comparison of load testing			
Environment		**CakePHP**	**CodeIgniter**
Local Server	**Response Time**	**-**	**+**
	Throughput	**-**	**+**
Live Server	**Response Time**	**0**	**0**
	Throughput	**-**	**+**

Table 5.5: Results comparison for Normal conditions

The result summary is presented below for stress conditions. The stress condition for local server is the load testing with 1100 users and the stress conditions for live server is the load testing with 600 users.

Results comparison of stress testing			
Environment		**CakePHP**	**CodeIgniter**
Local Server	**Response Time**	**-**	**+**
	Throughput	**-**	**+**
Live Server	**Response Time**	**+**	**-**
	Throughput	**+**	**-**

Table 5.6: Results comparison for stress conditions

6 DISCUSSION

The aim of the study was to measure the performance of the CAKEPHP and CodeIgniter frameworks on local and live server, with respect to load and stress testing and how object-relational mapping affect the performance of these frameworks. Whenever, small or medium sized companies decide to start develop website by using PHP framework, the first thing they have to consider that which framework should be used for development. Our study can provide some facts related to performance of two, widely used PHP frameworks i.e. CAKEPHP and CodeIgniter. We chose the topic because usually companies have to spend time and money to find out that which PHP framework they should choose for their business.

We gathered results from local and live server by testing both the frameworks. The statistical analysis of results from live server showed that overall there is no significant difference of average response time between the CAKEPHP and CodeIgniter. But if we compare the response time of both the frameworks in each iteration, we can see that with the load of 100 - 300 users, the response time of CodeIgniter is slightly better than CAKEPHP. But with load of 200, 400 and 500 users the response time of CAKEPHP is better than CodeIgniter.

On the stress conditions with the load of 600 users, the response time of CAKEPHP (5.9 sec) is significantly better than CodeIgniter (22.4 sec). On the stress conditions CAKEPHP executed 11953 HTTP requests and CodeIgniter executed 11229 HTTP requests. CAKEPHP executed more HTTP requests than CodeIgniter on stress conditions and its response time was also better than CodeIgniter. After careful analysis of the response time of individual HTTP requests involved in the test plan, it was accrued that in the first request, in which the initialization of the framework takes place; CodeIgniter has longer response time than CAKEPHP, in the case of live server. The initialization of the CodeIgniter application took more time than CAKEPHP application initialization, in all the iterations, which affected the overall response time of CodeIgniter application, in complete test plan.

After statistically analyzing the throughput of both frameworks on live server, it became evident that the throughput of the CodeIgniter framework was significantly better, in all the normal load conditions from 100 - 500 users. The throughput of CAKEPHP increased gradually, as the number of users were increased from 100 to 400 (128.1- 329.6 kb/sec). It faced a decline as the number of users rose to 500 (285.2 kb/sec). This decrease in throughput, upon increasing load is due to the 20 % of error rate in HTTP requests processed by CAKEPHP. There was a slight improvement as the number of users grew to 600 (297.2 kb/sec). But on stress conditions with 600 users, CAKEPHP performed better than CodeIgniter with the perspective of throughput, because CAKEPHP executed more number of HTTP requests than CodeIgniter.

From the results acquired by testing applications on the local server showed that CodeIgniter performed much better than CAKEPHP framework with respect to response time and throughput.

The results further showed that the response time of both the CAKEPHP and CodeIgniter increased to a certain number of users (500 in the case of CAKEPHP and 400 in the case of CodeIgniter) but faced a decline as the number of users increased. The increase or decrease in response time was not constant though.

Conceptually the response time should increase with an increase in load, on both the applications. This decrease in response time is due to increase in error rate in HTTP requests, processed by both the applications. Some HTTP requests were declined by the local server due to limited number of socket connections. Local server can only process limited number

of HTTP requests simultaneously. But after statistically analyzing the, successfully processed HTTP requests in a test plan, we came to conclusion that CodeIgniter performed better than CAKEPHP with respect to response time and throughput, in both normal and stress conditions on local server,

Table 5.1 and 5.2 showed the response time and throughput of CAKEPHP and CodeIgniter framework on local server respectively. After comparing the response time and throughput of both the frameworks, it can be seen that due to the absence of object-relational mapping (ORM) implementation in CodeIgniter, it had overall better performance with respect to response time and throughput in both the normal load and stress conditions. Although built-in support of ORM in CAKEPHP provided facility, to define relationship between database tables in the model and automatically generated the complex code for data access, according to these relations defined in application model. But it created an extra overhead in the performance of the application. Because the instantiation of the ORM object consumed more memory and CPU utilization rather than the execution of the simple SQL queries in CodeIgniter. ORM has to take care for different data usage scenarios so it generated complex code automatically for data access, which took more time to execute. Due to this overhead in performance, CAKEPHP did not perform better than CodeIgniter framework.

Table 5.3 and 5.4 showed the response time and throughput of CAKEPHP and CodeIgniter framework on live server respectively. After comparing the response time and throughput of both the frameworks, it can be seen that CodeIgniter performed better with respect to response time and throughput in normal load (100 – 500 users) conditions. Due to built-in support of ORM in CAKEPHP created overhead in its performance and that's why CAKEPHP could not perform better than CodeIgniter in normal load conditions. But in stress (600 users) conditions CAKEPHP significantly performed better with respect to response time and throughput. In stress conditions, built-in support of ORM in CAKEPHP framework improved its performance. Because ORM is complete data abstraction layer and it handles wide range of different data usage scenarios so it has the ability to handle the concurrent database connections in stress conditions. Due to built-in support of ORM, CAKEPHP effectively handled the stress conditions and processed not only more HTTP requests (11953) than CodeIgniter (11229) but also efficiently processed these requests. Even though in normal load conditions, ORM created overhead in the performance of CAKEPHP but in stress conditions it applied positive effect in its performance. CodeIgniter was running simple SQL queries to communicate with database, which was effective in normal load conditions because it took less time to execute simple SQL queries. But in stress conditions, simples SQL queries in CodeIgniter did no manage stress condition effectively.

CodeIgniter framework does not follow strictly MVC design pattern and object oriented paradigm. CAKEPHP strictly follows the MVC design pattern and object oriented paradigm that's why CAKEPHP has to load a lot of libraries, helpers and components which also create the overhead in its performance.

7 CONCLUSION AND FUTURE WORK

After analyzing the results collected from local and live server. Overall CodeIgniter framework performed better on local with perspective of response time and throughput. But on live server we got mix results related to response time. In the start of load testing the CodeIgniter performed better than CAKEPHP but as the load increased on the application then CAKEPHP performed better than CodeIgniter with the perspective of response time. But if we compare both frameworks with the perspective of throughput the CodeIgniter performed better than CAKEPHP.

After analyzing the results it is clear that CodeIgniter performed better in normal load conditions on live server but in stress conditions CAKEPHP performed better. Moreover CodeIgniter is very light, it does not have huge libraries of functions, Authentication system and access control list like in CAKEPHP. So we can conclude that CodeIgniter framework is better for small or medium sized applications. CodeIgniter can handle less concurrent users. Due to the huge libraries of functions, helpers, object-relational mapping and components in CAKEPHP, it is good for big applications where there are a lot of concurrent users expected. It has the ability to perform better in the stress conditions.

The built-in support of object-relational mapping (ORM) in CAKEPHP reduced the development time. It provided the feature to define the relations of database tables in the model to fetch data from different tables. But it created overhead in the performance of the application developed in the CAKEPHP framework in normal and stress conditions on local server. On live server, in normal conditions ORM support in CAKEPHP created overhead in performance but in stress conditions, ORM support created a positive effect on the performance of the CAKEPHP that's why CAKEPHP performed better in stress conditions. We conclude that the existence of the ORM in the framework is not useful for the performance of the framework in normal load conditions but it is useful for the performance of the framework in the stress conditions.

The importance of ORM cannot be denied because it increases the productivity, maintainability and reusability of the code. So ORM supported frameworks should be used for large scale applications, where huge database communication involved.

In the future the performance of these frameworks (CAKEPHP and CodeIgniter) can be measured with other type of performance parameters i.e. Capacity testing. As more new PHP frameworks have been introduced in the web development industry so this study can be increased by including more PHP frameworks i.e. Yii, Symfony, Zend etc. The performance of CAKEPHP and CodeIgniter frameworks can be measured for load testing by testing different complexity applications to get better results. These applications should be developed by different developers who have different experience of web development in PHP frameworks (CAKEPHP and CodeIgniter). In future the performance of the PHP frameworks can be measure by using third party Object-relational mapping implementation instead of using built-in ORM support of the framework.

8 REFERENCES

[1] W.Cui, L.Huang, L.J.Liang, J.Li, "The Research of PHP Development Framework Based on MVC Pattern", Conference on Computer Sciences and Convergence Information Technology, IEEE Computer Society, 2009.

[2] C.Supaartagorn, "PHP Framework for database management based on MVC pattern", Department of Mathematics Statistics and Computer, Ubon Ratchathani University, Thailand, 2011.

[3] P.R.Morpeth, J.Ellman, "Some Security Issues for web based frameworks", School of Computing, Engineering and Information Sciences, Northumbria University, UK, IEEE, 2010.

[4] S.Drobi, "A New Era of Web Application Development", IEEE, 2012.

[5] Y.Zhang, "An Excellent Web Content Management System", Department of Education Science and Media Engineering Weifang University Weifang, China, IEEE, 2011.

[6] S.K.Patel, V.R. Rathod, S.Parikh, "Joomla Drupal and WordPress - A Statistical Comparison of Open Source CMS", IEEE, 2011.

[7] K. Hokamura, R. Naruse, M. Shiozuka, N. U. S.Nakajima, A.Iwai, "AOWP: Web-specific AOP framework for PHP", IEEE/ACM International Conference on Automated Software Engineering, 2009.

[8] "Cake PHP makes building web applications simpler, faster and require less code", available at, http://cakephp.org

[9] "Code Igniter is an open source web application framework that helps you write incredible PHP programs", available at, http://CodeIgniter.com

[10] E.Wells, L.G.Tassinary, "Standardized Performance Trajectory as a Measure of Usability", USA, 1998.

[11] G. Mak, "Spring MVC Framework," in Spring Recipes: A Problem-Solution Approach, Apress, 2010, pp. 321–393.

[12] G.B. Laleci, G. Aluc, A. Dogac, A.Sinaci, O. Kilic, F. Tuncer, "A semantic backend for content management systems", Knowledge-Based Systems 23, 2010.

[13] S. NAKAJIMA, K. HOKAMURA, N. UBAYASHI, "Aspect-Oriented Development of PHP-based Web Applications", IEEE Computer Software and Applications Conference Workshops, 2010.

[14] M. d. P. S. Zarate, G. A. Hernandez, A. R. Gonzalez, "Developing Lift-based Web Applications Using Best Practices", Procedia Technology 3, 2012, pp. 214 – 223

[15] R. L. T. Santos, P. A. Roberto, M. A. Goncalves, A. H. F. Laender, "A Web services-based framework for building componentized digital libraries", The Journal of Systems and Software 81, 2008, pp.809–822.

[16] S.Kaur, K. Kaur, D. Singh, " A framework for hosting web services in cloud computing environment with high availability", IEEE, 2012.

[17] C.Wohlin, P.Runeson, M.Host, M.C.Ohlsson, B.Regnell, A.Wesslen, "Experimentation in software engineering an introduction", USA, 2000.

[18] "PHP Review: 4 Most Common PHP Frameworks Reviewed – SevenL Networks", available at, http://blog.7l.com/php-review-4-most-common-php-frameworks-reviewed-sevenl-networks/

[19] "PHP frameworks, Part 1: Getting started with three popular frameworks Zend, symfony, CakePHP", available at, http://www.ibm.com/developerworks/library/os-php-fwk1/

[20] "CodeIgniter Vs CakePHP",[Online]. Available: http://www.dazines.co.uk/programming/CodeIgniter_vs_cakephp

[21] "CodeIgniter".[Online]. Available: http://ellislab.com/CodeIgniter/user-guide/

[22]The PHP Group. PHP faq what is PHP and what does it stand for http://php.net/manual/en/faq.general.php, June 2012.

[23] "CakePHP," Wikipedia, free encyclopedia, Last modified on 5 September 2013.[Online]. Available: http://en.wikipedia.org/wiki/CakePHP [Accessed: July 2013]
[24] "Stress Testing", Microsoft Developer Networks.[Online]. Available: http://msdn.microsoft.com/en-us/library/aa292187(v=VS.71).aspx#vxcontestingforperformanceanchorstresstesting [Accessed: June 2013]

[25] "Testing for Performance", Microsoft Developer Networks .[Online]. Available: http://msdn.microsoft.com/en-us/library/aa292187(v=VS.71).aspx#vxcontestingforperformanceanchormeasuringperformance [Accessed: June 2013]

[26] "Object Relational Mapping", Wikipedia, free encyclopedia, Last modified on 30 September 2013.[Online]. Available: http://en.wikipedia.org/wiki/Object-relational_mapping [Accessed: September 2013]

[27] "Load Testing Web Applications", J.D. Meier, Carlos Farre,Prashant Bansode, Scott Barber, and Dennis Rea, Microsoft Corporation, September 2007.[Online]. Available: http://msdn.microsoft.com/en-us/library/bb924372.aspx [Accessed: June 2013]

[28] "Cake Software Foundation. Intro to CakePHP, what is CakePHP", June 2012.[Online]. Available: http://book. cakephp.org/1.1/view/307/Introduction-to-CakePHP

[29] https://en.wikipedia.org/wiki/Client-side

[30] N. Håkan, "PHP Framework Performance for Web Development Between CodeIgniter and CakePHP", August 2012.
[31] D.Golding, "Beginning CakePHP From Novice to Professional", 2008.

[32] D.Upton, "CodeIgniter for rapid PHP application development", 2007.

[33] P.K.Singh, P.Gupta, S.S.Bedi, K.Singh, "Analyze the performance of New Edge Web Application's over N-Tiers Layer Architecture", pp. 299-305, 2011.

[34] R.P.Ortiz, J. A. Gil, J. Sahuquillo, A. Pont, “The impact of user’s dynamic behavior on web performance”, IEEE, 2012.

[35] R.P.Ortiz, J. A. Gil, J. Sahuquillo, A. Pont, “Analyzing web server performance under dynamic user workloads”,pp. 386-395, IEEE, 2013.

[36] “Client Side Scripting”, Wikipedia, free encyclopedia , Last modified on 20 September 2013.[Online]. Available: http://en.wikipedia.org/wiki/Client-side_scripting [Accessed: July 2013]

[37] “Server Side Scripting”, W3Schools .[Online]. Available: http://www.w3schools.com/web/web_scripting.asp [Accessed: June 2013]

[38] “Model View Controller”, CakeBook .[Online]. Available: http://book.cakephp.org/1.3/en/The-Manual/Beginning-With-CakePHP/Understanding-Model-View-Controller.html [Accessed: July 2013]

[39] “Java Programming / Design Patterns”, Wiki Books . Last modified on 3 September 2013[Online]. Available: http://en.wikibooks.org/wiki/Java_Programming/Design_Patterns [Accessed: August 2013]

9 APPENDIX

9.1 Source code of applications

We have uploaded our source code on github. We created public repository on github. The code both CAKEPHP and CodeIgniter can be accessed through the following links of github repositories.

9.1.1 Source code of CAKEPHP

The source code of CAKEPHP application can be accessed by the following link of the github repository for CAKEPHP.

https://github.com/aliraza368/CAKEPHPblog

We have deployed CAKEPHP application on the live server. So our CAKEPHP application can be accessed by clicking the following link.

http://cake.shayansolutions.com/

9.1.2 Source code of CodeIgniter

The source code of CodeIgniter application can be accessed by the following link of the github repository for CodeIgniter.

https://github.com/aliraza368/CodeIgniter

We have deployed CodeIgniter application on the live server. So our CodeIgniter application can be accessed by clicking the following link.

http://CodeIgniter.shayansolutions.com/

9.2 Test Results

We are presenting the results of load testing which JMeter provided us in the form of csv file. We will present results of CAKEPHP and CodeIgniter frameworks gathered from local and live server separately. We uploaded all the results on githum repository. These results can be accessed by clicking on the following link. This is public repository.

https://github.com/aliraza368/loadtestresults

9.2.1.1 Data Sets on local server

The following tables show the datasets on local testing.

9.2.1.1.1 CAKEPHP Data Sets on local server

The following tables showing the datasets of CAKEPHP load testing.

	100 Users				200 Users			
sampler_label	count	average	Stddev	Throughput	count	average	Stddev	Throughput
/cake/	100	20086	36093	8.27	200	266261	260539	11.095
/cake/posts/about	100	6608	5295	2.15	200	14316	8554	1.89
/cake/users/login	200	8696	8567	1.94	400	21220	13561	1.14
/cake/users/dashboard	100	7434	5114	0.78	200	15036	7321	0.47
/cake/categories/add_category	300	10256	7794	2.89	600	21406	12205	1.75
/cake/posts/add_new_post	300	12314	9118	12.25	600	23811	12449	11.87
/cake/posts/view_post/1	300	13266	9152	12.55	600	24694	12116	12.13
/cake/posts/search	100	10324	5401	2.05	200	20501	6365	1.70
/cake/categories/view_category/1	100	9601	5684	1.77	200	17415	6832	1.47
/cake/users/logout	100	16872	11153	0.91	200	33137	12818	0.52
TOTAL	1700	11519	12253	39.99	3400	36402	86229	39.03

	300 Users				400 Users			
sampler_label	count	average	Stddev	Throughput	count	average	Stddev	Throughput
/cake/	300	548123	486647	9.56	400	440056	515101	8.17
/cake/posts/about	300	17593	8192	1.70	400	13995	10657	1.53
/cake/users/login	600	25309	13797	0.86	800	19506	16218	1.02
/cake/users/dashboard	300	17075	7143	0.37	400	13035	9271	0.45
/cake/categories/add_category	900	23710	12684	1.38	1200	18383	15139	1.64
/cake/posts/add_new_post	900	26090	13379	11.43	1200	20402	16470	11.02
/cake/posts/view_post/1	900	26793	13092	11.58	1200	21592	17215	9.98
/cake/posts/search	300	23298	8042	1.25	400	18283	12289	1.30
/cake/categories/view_category/1	300	19511	8318	1.48	400	16288	11983	1.53
/cake/users/logout	300	35428	14174	0.44	400	28532	21282	0.5
TOTAL	5100	55378	171059	38.34	6800	44137	160133	35.52

		500 Users				600 Users		
sampler_label	count	average	Stddev	Throughput	count	average	Stddev	Throughput
/cake/	500	416510	542642	7.81	600	396800	559496	8.84
/cake/posts/about	500	15730	69126	1.61	600	13976	61585	1.87
/cake/users/login	1000	20503	69826	1.12	1200	18291	63171	1.26
/cake/users/dashboard	500	15216	67843	0.49	600	13555	63409	0.56
/cake/categories/add_category	1500	24293	104090	1.74	1800	17859	64505	1.91
/cake/posts/add_new_post	1500	25997	103555	11.21	1800	19416	62989	11.50
/cake/posts/view_post/1	1500	24002	79347	10.69	1800	22298	81517	9.73
/cake/posts/search	500	17385	12978	1.22	600	15596	13767	1.16
/cake/categories/view_category/1	500	21904	97000	1.47	600	16899	63769	1.57
/cake/users/logout	500	26205	20804	0.57	600	26397	67319	0.57
TOTAL	8500	45696	180434	35.36	10200	41090	174312	36.70

	700 Users				800 Users			
sampler_label	count	average	Stddev	Throughput	count	average	Stddev	Throughput
/cake/	700	353666	571189	6.66	800	349035	610953	9.77
/cake/posts/about	700	12829	64410.86	1.57	800	12150	67001	1.83
/cake/users/login	1400	17194	65973	1.30	1600	16121	68284	1.41
/cake/users/dashboard	700	12935	64023.64	0.59	800	11790	66325	0.64
/cake/categories/add_category	2100	17695	74427.67	1.95	2400	16616	77176	2.11
/cake/posts/add_new_post	2100	17099	63278.22	9.19	2400	17982	67779	12.39
/cake/posts/view_post/1	2100	17812	64491.48	9.25	2400	20771	86002	12.18
/cake/posts/search	700	15740	63253.98	1.11	800	16530	66693	1.72
/cake/categories/view_category/1	700	15717	64417.77	1.34	800	12718	15317	1.59
/cake/users/logout	700	23192	66790.21	0.57	800	26475	97123	0.66
TOTAL	11900	36840	172125.3	31.57	13600	36885	182009	41.24

		900 Users			1000 Users			
sampler_label	count	Average	Stddev	Throughput	count	average	Stddev	Throughput
/cake/	900	258511	492544	8.79	1000	272135	514392	7.22
/cake/posts/about	900	9596	53244	2.05	1000	13733	95423	1.92
/cake/users/login	1800	12956	54974	1.73	2000	13693	68881	1.66
/cake/users/dashboard	900	11371	72886	0.80	1000	12122	81385	0.78
/cake/categories/add_category	2700	12647	52870	2.57	3000	14106	73822	2.53
/cake/posts/add_new_post	2700	13090	53272	11.26	3000	14336	68632	11.12
/cake/posts/view_post/1	2700	14326	60954	11.21	3000	14471	63261	10.12
/cake/posts/search	900	12138	53838	1.367	1000	15013	80638	1.30
/cake/categories/view_category/1	900	9623	12514	1.70	1000	14175	82276	1.66
/cake/users/logout	900	18984	75618	0.79	1000	17507	52621	0.81
TOTAL	15300	27431	143571	39.73	17000	29459	155475	37.53

		1100 Users		
sampler_label	count	average	Stddev	Throughput
/cake/	1005	247658	496729	7.06
/cake/posts/about	1005	8914	50419	1.94
/cake/users/login	2010	11873	52142	1.83
/cake/users/dashboard	1005	9025	51111	0.86
/cake/categories/add_category	3015	12337	59634	2.73
/cake/posts/add_new_post	3015	12133	52212	10.25
/cake/posts/view_post/1	3015	13084	59321	10.19
/cake/posts/search	1005	11231	52104	1.30
/cake/categories/view_category/1	1005	11721	59113	1.61
/cake/users/logout	1005	17772	73431	0.87
TOTAL	17085	26043	143604	36.63

Table 9.1: CakePHP Dataset for local server

9.2.1.1.2 CodeIgniter Data Sets on local server

		100 Users			200 Users			
sampler label	count	average	Stddev	Throughput	count	average	Stddev	Throughput
/CodeIgniter-blog/	100	1383	1050	33.79	200	57567	69141	28.39
/CodeIgniter-blog/about	100	272	341	12.38	200	7233	6338	9.62
/CodeIgniter-blog/auth/login	200	527	694	3.86	400	7317	6148	1.82
/CodeIgniter-blog/dashboard	100	367	423	1.79	200	6266	4962	0.83
/CodeIgniter-blog/add-category	300	622	785	13.02	600	8549	7423	5.48
/CodeIgniter-blog/add-new-post	300	634	799	49.94	600	8241	6890	43.60
/CodeIgniter-blog/view_post/1	300	382	458	35.00	600	6640	4801	31.16
/CodeIgniter-blog/blog/search	100	332	402	2.70	200	6287	4859	1.12
/CodeIgniter-blog/category/as	100	493	504	14.73	200	7568	5696	13.84
/CodeIgniter-blog/auth/logout	100	351	461	1.99	200	7697	6248	0.82
TOTAL	1700	539	703	161.83	3400	10443	21368	127.63

		300 Users			400 Users			
sampler_label	count	average	Stddev	Throughput	count	average	Stddev	Throughput
/CodeIgniter-blog/	300	283601	249551	24.63	400	343100	328241	28.17
/CodeIgniter-blog/about	300	10285	4563	8.39	400	12980	9009	9.27
/CodeIgniter-blog/auth/login	600	12698	5476	1.06	800	12372	8845	1.14
/CodeIgniter-blog/dashboard	300	12718	5248	0.50	400	12753	9333	0.53
/CodeIgniter-blog/add-category	900	16993	9115	3.58	1200	15424	29682	3.56
/CodeIgniter-blog/add-new-post	900	17519	18379	40.86	1200	16253	40719	44.95
/CodeIgniter-blog/view_post/1	900	13370	4818	29.48	1200	12303	7027	31.75
/CodeIgniter-blog/blog/search	300	11050	4222	0.73	400	14418	48685	0.73
/CodeIgniter-blog/category/as	300	11761	4194	13.43	400	13830	8668	14.31
/CodeIgniter-blog/auth/logout	300	13285	5197	0.54	400	14284	9429	0.58
TOTAL	5100	30102	88169	118.85	6800	33415	113838	128.98

	500 Users				600 Users			
sampler label	sample	average	Stddev	Throughput	count	average	Stddev	Throughput
/CodeIgniter-blog/	500	274245	336429	26.66	600	264457	369512	25.27
/CodeIgniter-blog/about	500	8424	6954	9.02	600	8848	8913	8.53

/CodeIgniter-blog/auth/login	1000	10194	8137	1.34	1200	9536	9352	1.49
/CodeIgniter-blog/dashboard	500	10316	8185	0.64	600	8813	8610	0.71
/CodeIgniter-blog/add-category	1500	13527	11983	3.75	1800	11827	12581	3.83
/CodeIgniter-blog/add-new-post	1500	13886	12169	42.48	1800	12593	13099	40.54
/CodeIgniter-blog/view_post/1	1500	10974	8331	30.82	1800	9586	8721	29.18
/CodeIgniter-blog/blog/search	500	9025	6824	0.85	600	8842	8200	0.91
/CodeIgniter-blog/category/as	500	9827	7354	13.94	600	9697	8898	13.22
/CodeIgniter-blog/auth/logout	500	10815	8271	0.69	600	10572	10132	0.75
TOTAL	8500	26953	102817	126.51	10200	25431	108213	120.93

	700 Users				800 Users			
sampler_label	Sample	average	Stddev	Throughput	sample	average	Stddev	Throughput
/CodeIgniter-blog/	700	311037	430933	25.26	800	248544	388879	25.93
/CodeIgniter-blog/about	700	11512	43758	8.32	800	13117	63069	8.80
/CodeIgniter-blog/auth/login	1400	11279	40939	1.41	1600	10361	51839	1.63
/CodeIgniter-blog/dashboard	700	10471	43147	0.68	800	10111	49912	0.78
/CodeIgniter-blog/add-category	2100	13264	48542	3.62	2400	13825	55504	3.95
/CodeIgniter-blog/add-new-post	2100	13315	42234	40.86	2400	14144	55517	41.43
/CodeIgniter-blog/view_post/1	2100	11616	43113	30.07	2400	11427	51576	31.99
/CodeIgniter-blog/blog/search	700	12558	58100	0.86	800	10549	50671	0.98
/CodeIgniter-blog/category/as	700	11202	10623	13.62	800	13743	71428	14.62
/CodeIgniter-blog/auth/logout	700	14520	58427	0.71	800	10487	37146	0.79
TOTAL	11900	29909	133223	121.17	13600	26203	121548	126.56

		900 Users			1000 Users			
sampler label	sample	average	Stddev	Throughput	Sample	average	Stddev	Throughput
/CodeIgniter-blog/	900	234792	383524	26.88	1000	213487	390668	23.43
/CodeIgniter-blog/about	900	9170	35740	9.09	1000	8750	49196	8.13
/CodeIgniter-blog/auth/login	1800	8916	36061	1.89	2000	8511	36294	2.04
/CodeIgniter-blog/dashboard	900	8398	34236	0.92	1000	8177	35419	0.99
/CodeIgniter-blog/add-category	2700	10376	35458	4.38	3000	9974	35665	4.50
/CodeIgniter-blog/add-new-post	2700	11389	39535	42.89	3000	10660	41083	38.98
/CodeIgniter-blog/view_post/1	2700	8840	32155	31.54	3000	8955	35072	29.19

/CodeIgniter-blog/blog/search	900	8298	31716	1.08	1000	8837	44789	1.16
/CodeIgniter-blog/category/as	900	9330	34681	14.29	1000	9644	48713	13.27
/CodeIgniter-blog/auth/logout	900	10292	34610	0.93	1000	10070	40926	1.00
TOTAL	15300	22937	112411	128.04	17000	21456	112915	118.63

		1100 Users		
sampler_label	Sample	average	Stddev	Throughput
/CodeIgniter-blog/	1002	228904	399205	27.25
/CodeIgniter-blog/about	1002	8989	45737	9.209
/CodeIgniter-blog/auth/login	2004	9649	48217	1.81
/CodeIgniter-blog/dashboard	1002	9725	48228	0.87
/CodeIgniter-blog/add-category	3006	11930	51277	4.15
/CodeIgniter-blog/add-new-post	3006	12247	51353	43.19
/CodeIgniter-blog/view_post/1	3006	10599	51744	33.54
/CodeIgniter-blog/blog/search	1002	12022	66960	1.08
/CodeIgniter-blog/category/as	1002	9401	33104	15.36
/CodeIgniter-blog/auth/logout	1002	11823	58222	0.89
TOTAL	17034	23793	120283	134.60

Table 9.2: CodeIgniter Dataset for local server

9.2.1.2 Data Sets on live server

The following tables show the datasets on live testing.

9.2.1.2.1 CAKEPHP Data Sets on live server

		100 Users			200 Users			
sampler label	sample	average	Stddev	Throughput	Sample	average	Stddev	Throughput
/	100	2469	730	21.53	200	3718	1631	42.02
/posts/about	100	725	242	6.21	200	818	445	9.15
/users/login	300	756	284	7.92	600	894	498	7.59
/users/dashboard	100	576	128	2.49	200	656	295	2.42
/categories/add_category	300	782	297	9.92	600	965	533	9.64
/posts/add_new_post	300	1213	426	37.52	600	2025	1255	58.33
/posts/view_post/2	300	1258	465	44.75	600	2089	1478	64.13
/posts/search	100	758	189	6.06	200	1380	936	8.42
/categories/view_category/2	300	600	153	7.21	600	720	380	6.62
/users/logout	100	1191	236	3.41	200	1303	429	3.13
TOTAL	2000	977	553	128.13	4000	1398	1202	190.26
		300 Users			400 Users			
sampler_label	Sample	average	Stddev	Throughput	sample	average	Stddev	Throughput
/	300	4331	1922	63.58	400	6262	5344	79.23
/posts/about	300	973	537	12.10	400	1417	1303	14.12
/users/login	900	917	433	8.149	1200	1102	901	7.58
/users/dashboard	300	677	245	2.43	400	871	690	2.26
/categories/add_category	900	992	497	9.69	1200	1187	925	9.09
/posts/add_new_post	900	2341	1472	86.25	1200	3045	2428	105.27
/posts/view_post/2	900	2227	1375	96.05	1200	3191	2563	113.69
/posts/search	300	1560	907	11.79	400	2855	3472	13.63
/categories/view_category/2	900	716	285	6.99	1200	858	598	6.38
/users/logout	300	1394	414	3.30	400	1626	899	3.02
TOTAL	6000	1526	1324	279.78	8000	2059	2484	329.69
		500 Users			600 Users			
sampler label	Sample	average	Stddev	Throughput	sample	average	Stddev	Throughput
/	500	8295	6680	89.20	600	12494	10762	91.46
/posts/about	500	3111	3479	15.46	600	4758	6461	15.99
/users/login	1500	2071	2578	7.30	1791	3535	5679	6.74
/users/dashboard	500	1524	2002	2.43	600	2847	3907	2.18
/categories/add_category	1500	2065	2435	9.14	1798	4302	8948	6.74
/posts/add_new_post	1500	4655	4479	93.75	1793	8649	10667	89.26
/posts/view_post/2	1500	4491	4367	87.76	1791	8856	9116	112.26
/posts/search	500	4018	4705	10.41	597	6811	6868	12.21
/categories/view_category/2	1500	2338	3406	6.29	1790	3733	11305	5.33
/users/logout	500	3170	3621	2.74	593	5202	13049.25	2.42
TOTAL	10000	3349	4100	285.25	11953	5967	9600.41	297.20

Table 9.3: CakePHP Dataset on live server

9.2.1.2.2 CodeIgniter Data Sets on live server

		100 Users			200 Users			
sampler label	count	average	Stddev	Throughput	count	average	Stddev	Throughput
/	100	2960	1335	32.39	200	6131	2688	143.36
/about	100	864	208	12.319	200	1274	989	46.08
/auth/login	200	498	173	3.916	400	520	179	4.18
/auth	100	486	78	1.84	200	508	211	1.95
/add-new-category	300	699	298	13.29	600	754	488	14.07
/add-new-entry	300	1074	463	51.75	600	1970	1763	188.76
/post/102	300	1094	486	64.05	600	2289	2133	232.19
/blog/search	100	514	138	6.75	200	1047	930	18.23
/category/cate	300	657	401	46.99	600	2123	1916	169.29
/auth/logout	100	476	56	1.98	200	520	211	1.92
TOTAL	1900	888	710	208.40	3800	1680	1976	714.77
		300 Users			400 Users			
sampler_label	count	average	Stddev	Throughput	count	average	Stddev	Throughput
/	300	4814	2157	107.25	400	9369	10986	123.16
/about	300	1201	1402	34.70	400	2531	4141	39.62
/auth/login	600	532	398	4.14	800	701	1029	3.61
/auth	300	520	258	1.93	400	623	589	1.68
/add-new-category	900	732	397	14.13	1200	1046	1237	12.23
/add-new-entry	900	1614	1600	149.40	1200	4391	7191	166.98
/post/102	900	1812	1544	185.95	1200	4483	7612	212.05
/blog/search	300	865	883	14.95	400	1648	2495	16.43
/category/cate	900	1430	1254	139.74	1200	3264	5470	162.25
/auth/logout	300	492	94	2.09	400	635	657	1.79
TOTAL	5700	1354	1536	615.23	7600	2935	5888	695.47
		500 Users			600 Users			
sampler label	count	average	Stddev	Throughput	count	average	Stddev	Throughput
/	500	14207	16212	147.63	600	70951	210265	26.89
/about	500	4120	5492	48.02	598	38444	233617	9.87
/auth/login	1000	768	835	3.52	1189	4976	109658	0.66
/auth	500	675	525	1.64	594	1517	3052	0.96
/add-new-category	1500	1216	1160	11.89	1780	4338	75274	2.18
/add-new-entry	1500	6933	7624	202.83	1775	30064	130770	43.39
/post/102	1500	7871	9318	251.02	1767	35841	128782	57.97
/blog/search	500	3613	11528	19.09	587	8897	15480	12.67
/category/cate	1500	6279	5966	200.51	1756	28255	143178	49.86
/auth/logout	500	713	626	1.69	583	1476	3247	0.96
TOTAL	9500	4830	8004	833.73	11229	22486	127650	195.65

Table 9.4: CodeIgniter Dataset for live server

9.3 T-testing

9.3.1 T-testing on local server

9.3.1.1 T-test on response time of CAKEPHP and CodeIgniter

t-Test: Two-Sample Assuming Unequal Variances		
	Average Response time CAKEPHP (sec)	*Average Response time CodeIgniter (sec)*
Mean	36.43	22.71
Standard Deviation	11.89	10.02
Observations	10	10
t Stat	2.78	
T critical one tail level of significance = 0.05	1.73	
t Critical one-tail with Level of significance 0.01	2.56	
T critical one tail level of significance = 0.001	3.64	

Table 9.5: T-Test of CakePHP and CodeIgniter on local server

9.3.1.2 T-test on throughput of CAKEPHP and CodeIgniter

	Throughput CakePHP (kb/sec)	*Throughput CodeIgniter(kb/sec)*
Mean	37.46	127.87
Standard Deviation	2.84	12.55
Observations	10	10
t Stat	-22.20	
t Critical one-tail with Level of significance = 0.05	1.81	
t Critical one-tail with Level of significance = 0.01	2.76	
t Critical one-tail with Level of significance = 0.001	4.14	

Table9.6: T-Test on Throughput of CakePHP and CodeIgniter on local server

9.3.2 T-testing on live server

9.3.2.1 T-test on response time of CAKEPHP and CodeIgniter

t-Test: Two-Sample Assuming Unequal Variances		
	average CakePHP (sec)	*average CodeIgniter (sec)*
Mean	1.8	2.28
Standard Deviation	0.92	1.60
Observations	5	5
t Stat	-0.57	
t Critical one-tail with Level of significance = 0.05	1.94	
t Critical one-tail with Level of significance = 0.01	3.14	
01t Critical one-tail with Level of significance = 0.001	5.20	

Table 9.7: T-Test of CakePHP and CodeIgniter on live server

9.3.2.2 T-test on throughput of CAKEPHP and CodeIgniter

t-Test: Two-Sample Assuming Unequal Variances		
	Throughput CakePHP	*Throughput CodeIgniter*
Mean	242.56	613.48
Standard Deviation	81.57	239.55
Observations	5	5
t Stat	-3.27	
t Critical one-tail with Level of significance = 0.05	2.01	
t Critical one-tail with Level of significance = 0.01	3.36	
t Critical one-tail with Level of significance = 0.001	5.89	

Table 9.8: T-Test of CakePHP and CodeIgniter on live server