PHP Framework Performance for Web Development

Håkan Nylén

Blekinge Institute of Technology, Karlskrona, Sweden,
hakan@dun.se

Abstract. [Context] The new PHP Framework ideas have become popular among developers. It is also important to know how big an impact they have on the performance of the website. Comparing the two top frameworks with each other can shed some light on what the performance looks like today on the web with PHP as its base. **[Problem]** Visitors nowadays have less patience to wait for a website to load. Meanwhile, PHP Framework has become known among developers, but the part of the performance that reduces the load time even more, so visitors can surf without any problems as be missing. Therefore it is a good idea to try to discovery how the performance of the PHP Framework can change and improve the visitor experience. **[Contribution]** In this paper is a description of one of the first performance experiment on PHP Frameworks. It can help people make the right decisions regarding PHP Framework in future. The lack of data in this area is also one of the decision to make this paper as well.

1 Introduction

The internet became popular in the beginning of the 90s. Different websites, running on HTTP-servers and based on the official HTML language, were the face outwardly seen by visitors. And there were discussion about the ability to store and perform calculations on data on websites, but it was not possible to do such with static files like HTML. However, you could use several languages to do so. The Founder of PHP, Rasmus Lerdorf, used PERL to create PHP because of the massive amount of code you needed to code in PERL.[10] At first there were only small PERL codes in order to make coding a website easier than before. He created PHP for his own personal need in his quest for the best possible webpage. He released it just for fun and it became famous as a good idea to a simplify web development. His idea has become and is still today a popular project, a script language that increasingly is becoming object-oriented, if not already.

The web is different nowadays. Many are talking about speed and performance. There are different ways to make the website faster, because when the page loading is to slow you will lose visitors. A new service, by the name of cloud,

has become famous and everyone want to use it, for example CDN[1] shows have cloud can be used. Except CDN there are also other different performance approaches, like the web-server, the server in general or the page. This could mean configuration or tweaks making the page load faster.

PHP Framework, i.e. libraries and helpers, mainly focuses on making the development of websites faster. This makes it very popular among developers wanting to make a fast demo page for a new project. When the page later is up for production on a website, everyone can reach it. At this stage it is important to think about performance, because users do not wait long for a page to load, like most did in the 90s because of the slow internet connection. PHP Framework has help the development of websites but the important point which is faster websites for the visitors is still missing.

The research focus is an evaluations of web performances in general and on testing the most used and scientifically researched tool to evaluate two of the biggest PHP Frameworks, CodeIgniter and CakePHP.

The goal of this thesis is to specifically define a way to start thinking about the performance of PHP Frameworks and Web performance in general.

The expected outcome of this thesis is to be able to point the developer in the right direction when evaluating PHP Frameworks, and the tests can been seen as guidelines for future work and an initial contribution to get any real data on the performance for PHP Frameworks.

2 The Background

Evaluation in this paper is all about the performance of the load[2], but it can be hardware, network and more. Evaluation in general is how you can test the performance in what way. This is very important in the choice of products nowadays.

The story began when the PHP (PHP: Hypertext Preprocessor) was born in 1995 and this was only a beginning. Now PHP is bigger and mainly a "programming language" in itself, more called a script language[10] for websites, inspired by Java, C, and PERL. [10] Developers using PHP have issues making better and faster websites, so therefore some developers have started to develop PHP Frameworks in order to help create better websites faster. There are two PHP Frameworks that have the most users, one is CodeIgniter [4], that was first released on 28 February 2006 [5]. It is focused on creating clean, fast framework for developers. The other Framework is CakePHP[12], which was released in 2005. CakePHP is mainly inspired by Ruby on Rails. [3]

[1] CDN is Content Delivery Network, a network of servers. Where static data for websites and more can be stored, a CDN is spread all over the world, making the request faster.

[2] load means how long a request takes in milliseconds (ms).

2.1 MVC

Model View Controller is a pattern that consisting of 3 different of types of classes, each class summarises what they will do. [1] Model manages datahandling, like data from databases. View manages how the data should be shown for the user. Controller is the middle component, collecting data from Model and making the data readable before sending it to the View. This paper will be about testing the performance of CodeIgniter[4] and CakePHP[12].

An example of a webpage with MVC could be a news page. A user uses a web browser to access the website, which connects to the controller as seen in point 1, in Figure 1. Controller needs data from the database so it will call a function in the model-class, as seen in point 2. The model connects to the database and is handles the queries and is returning the data to the controller, as seen in point 3. Controller then contacts and retrieves View, that mostly contains HTML and only uses PHP to loop out the data, and returns it to the user, as seen in point 4.

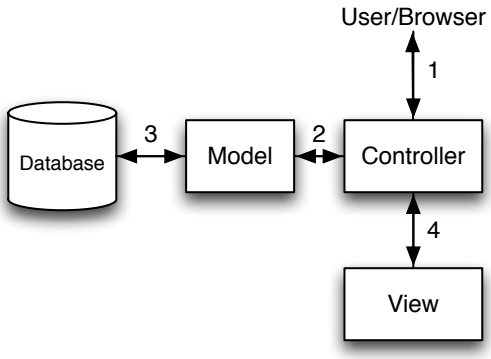

Figure 1. A simple example of an MVC structure

2.2 PHP Framework

PHP is a server-side programming language famous for making it simple to develop web applications in web development, especially in agile methods. The Frameworks are based on MVC with add-ons, i.e. formhelpers and database-controllers, in various languages like PHP for example. Frameworks are built to make it even faster to develop a website. The biggest PHP Frameworks with MVC-thinking are CodeIgniter and CakePHP.

Codeigniter and CakePHP have many libraries to help the developer. Codeigniter has no official inspiration, but its goal is to make it faster for the developer

to create good websites rather than creating them from scratch in raw PHP. CakePHP is inspired by Ruby on Rails, and you can see that in the structure of how you call the functions in a library in CakePHP. The strongest ability of Codeigniter is the openness where the developers can publish libraries and plugins to Codeigniter. CakePHP has this ability too, but doesn't have as many followers making requests for changes in the code as Codeigniter.[3]

The Difference is the structure but basically both are the same. They are based on MVC-thinking but above that both have different structures and different thinking about how the libraries for databases and controllers should behave and be called. The biggest difference is the handling of the model. In CakePHP you normally only type the variables, which the model should handle from the database. In Codeigniter you build the whole model with variables and functions to get the data from the database. You can do that with CakePHP but by using the already implemented functions in order to get, update, add, and remove data from the table in the database.

2.3 Client-side

Client-side means the browser and the scripts and other rendering happening on the visitor‘s side on the website. This does have a big impact in the performance on a server. [8] If using many images, the browser first loads the page and then sends the request to get the images and scripts on the page, adding more load to the server, because it generates more requests and thereby increasing the server load. [11]

2.4 Server-side

Server-side is the server. Here can the network, CPU, and RAM have an impact on the performance for the visitor on the site. [11] When Apache, the server software for http, is getting many requests at the same time, more CPU and RAM will be used and therefore unable to handle all the requests. In that case the visitor could experiences the website as slow. The network is often what stops more load to the server, because when Apache is optimized it does not use so much of the CPU and RAM, making the CPU and RAM the less of a problem.

2.5 Performance

Performances means different things, to keep it simple this paper will be about fast requests. Performance can also have a big impact on servers, for example a big server park in a data center. But, the data center cost a lot of money, which is a problem when experimenting in general. That is why there will only be tests

[3] This can be shown on their Git (version control system) repositories on Github, [7] and [6], a service where followers of the repository can be seen and also how many pull requests each project has.

on cloud with one server in order to see how the framework changes the ms of a request. This server is running on Amazon EC2, an isolated server environment on the cloud. A good thing with this set-up is that it is more or less how a real website is set-up nowadays, making it more trustworthy in the data. The server specification can be seen in Table 6.

3 Research

3.1 Research Questions

RQ1: What kind of web performance evaluations exist and how were they performed?
RQ2: What factors impact web performance?[4]
RQ3: To what extent are open source php frameworks evaluated?
RQ4: What are differences in the performance between the most commonly used open source php frameworks and how can they be evaluated?

My searched included more or less different types of strings like "web development evaluation" and "web development performance". In the findings there where 300 papers and of those 8 were relevant because the others were written about the evaluation in general, not about performance in web development.

3.2 Research Methodology

Research Question	Methodology
RQ1	Literature
RQ2	Literature
RQ3	Literature
RQ4	Data from RQ1, RQ2 and RQ3 to design experiment

3.3 Literature Survey

The literature was found in different stages. And the following sources were used:

- Google Scholar
- IEEE
- Google.

These strings were used, as seen in Table 1.

The search for data was about performance evaluation and the selection of literature research in this field was not plentiful but easy to find. The important is a good way on how they did the research to find out the information in the data, and this was often found in the abstract but sometimes in the background. There were often over 300 results in the search, but only up to 1-3 relevant pages, in each string, and the same papers were often found in many of the strings.

[4] There is a lack of data on php performance. However there is some data about web performance in general.

Strings
Web development Evaluation
Web development Performance
PHP Framework evaluation
PHP Framework performance
PHP Evaluation
PHP Performance
Website Performance
Website evaluation

Table 1. The strings for searching literature

4 Literature review

4.1 The Approach

The search was by the strings presented in Table 1, and in the search some hundreds of papers were found. The question was which approach would be the best by using the strings to find text about such as impact on performance and about performance of PHP. because of the lack of research in this area it became difficult to find any research about PHP performance at all. However it was easier to find research about web performance in general. I took everything of value in this area, that had something that could be used for the questions to be answered, such as impact on evaluation tests, php frameworks, different evaluation types and hows. The rest of the finding approach depended on the author's ability to find something that had made an impact for a website or something similar that could be an important thing to bring up in this research.

4.2 The Papers

In this section there are several papers listed along with descriptions of their contains and what these papers have in common.

PHP Team Development is a book that has its focus on how the MVC idea and techniques make a difference in the development. This book has MVC as a big part of PHP development and is about what it does for web development. This book was published in 2009. [1]

Analysis Of Model-based MVC Framework For PHP Development CodeIgniter is an analysis of how the MVC based framework Codeigniter is doing in the development of performance and of user ability for developers. This is a paper from May 2009 which has much in common with *PHP Team Development*, the book which has a chapter about MVC. [4]

EC2 FAQ: What Is An EC2 Compute Unit is a website where the user will find frequently asked questions (FAQ) and answers about EC2 and what a compute unit is. This website was last visited in april 2012. [2]

Understanding Web Performance presents how performance can be used and its importance in the web business. It also brings up how performance can be tested and what can impact the tests. It is a paper presented in the Business Communication Review, October 2001. [8]

A Performance Comparison Of Dynamic Web Technologies is paper in which there is a description of the different impacts on performance, comparative analyses of different performance types and how they can be done. Presented in ACM Sigmetrics, December 2003. [11]

Web-based IDE To Create Model And Controller Components For MVC-Based Web Applications On CakePHP is a paper about how the CakePHP is built and how to use it for coding and a little about how MVC works. The paper was published in December 2010. It has much in common with the other paper about MVC framework Codeigniter and the book *PHP Team Development.* [12]

5 Literature Results

5.1 Old Papers

The results below are based on old papers. These papers are based on the general web performance and the web performance has not changed much in 10 years. The papers about web performance, less then 10 years old, were difficult to find. They were mostly about Java, another programming language, that is not built up in the same way as PHP, making them more or less useless for this paper. Impacts and other topics are described in the reviewed papers to help answer the research questions (RQ). This same papers are as important today as when they were published, so the relevance has not changed.

5.2 RQ1: What kind of web performance evaluations exist and how were they performed?

Evaluation of the web is mostly done by load testing and counted in ms, but also by optimizing the page itself to minimize requests to the server, for example by reducing the amount of pictures, or by moving them to another server. This can be seen in Table 2.

Request/ms is mostly used to evaluate performance of web applications. There is mention of two different things in two different papers. Firstly, *Understanding Performance* contains the following comment:

Evaluation Topic	How	Year
requests/ms	experiment	2001[8][11]
server/CPU	experiment	2003 [11]
on-page optimization	experiment	2003[11]

Table 2. Different types of evaluation of a web performance

> Sure, speed matters, but it is not a one-dimensional problem. And, despite what you have heard, just adding more bandwidth does not always make things go faster. [8]

Large and many requests need a good Central Processing Unit (CPU) and network, but according to the article *Understanding Performance* speed also matters but is not always the only problem. Secondly, in *A Performance Comparison of Dynamic Web Technologies* tests are done with requests in ms:

> Consideration of overload behaviour may be just as important as the peak request rate when website administrators are choosing dynamic Web content generation technologies. [11]

The researchers understood the importance of choosing the right technologies when using dynamic web content, because of different peaks of requests, creating a need for handling the overload right. The amount and size of requests can change. They also show how good it can be to evaluate the performance by using requests per milliseconds or seconds.

Server/CPU is important when the server needs to handle many requests at the same time. In *A Performance Comparison of Dynamic Web Technologies* the following explanation can be found:

> Once the servers become overloaded, the CPU utilization of Apache 2.0.45 is lower than that of Apache 1.3.27. Under overload, Apache 2.0.45 is unable to accept TCP connections (and hence requests) as quickly as Apache 1.3.27. [11]

This is important for the CPU to function, but does not mean that Apache itself causes the CPU overload. It can be the network the CPU need to handle, that can causes the CPU overload.

On-page optimization is about how to create less requests to the server by removing or grouping several images in one, because browsers load the HTML first and then sends more requests to get the images. This also does have something to do with the size of the images, bigger images take more time to download. In *A Performance Comparison of Dynamic Web Technologies* there are tests done with static content [11] that can be images or normal HTML files. It shows it goes faster than dynamic content, but it is on request.

The Choice will be to use request per ms in the tests, often used in other research experiments and therefore more trustworthy. The tests are to see if it

are any difference in the performance from the visitor's perspective, making it less important to evaluate server hardware and on-page optimization, even if the last one is interesting. We need to focus on one thing.

5.3 RQ2: What factors impact web performance?

There are things that can have an impact on the performance while testing. The impacts are listed in Table 3.

Factor	How It Impacts	Comment
server	network, CPU, RAM	This can be fixed with a large server or a data center, something that can not be tested here.
network	speed	If the test will perform a request/ms load simulation, the network can be maximized without the server being reaching the performance limit, which has been set already.
wrong configuration	slowness, bad performance	Wrong configuration on the server, which can result in lower performance, will not be a problem because default configurations will be used, meaning no focus on optimization configurations in this paper.

Table 3. Different types of evaluation of a web performance

The server can have a big impact on performance because the CPU and RAM might not have the ability to handle the big amount of requests tested. Small CPU and RAM can make it overload and deny new requests, making it impossible to connect when that amount of requests are full.

The Network is not very important, but the network should be stable for the amount of requests sent to the server. Normal speed would be at least 10mbit down and 10mbit up to send and receive at good speed. The only problem with the network in my home is the stability, making it hard to trust the speed, making the ms higher than it should be. For example, the internet connection where the tests were run can have 100 mbit up and 100 mbit down, but only getting 34 mbit of both because of sharing the internet connection with 60 other people. this can have a big impact if alot of people are using the network, but the risk for that is small.

Wrong configuration is the thing that can present a problem because many people might not know that it can be a default or a wrong setup configuration that make the server to overload or not accept as many requests as it should. The tests will use defaults just to make it as clean as possible, because that is how many people actually are using the defaults in the configurations.

The Choice will be to try to make it more realistic by using server on the cloud with good internet, good hardware, and default configurations.

5.4 RQ3: To what extent are open source PHP frameworks evaluated?

No data were to be found regarding the performance of PHP framework and evaluations thereof. For example other authors could think that this area is covered because the papers on web performance already exist. The Papers are also outdated but there have not been any new inventions in HTML performance either. PHP is a script language making it totally different from normal web performance because of the compiling on run, as we can call it. It means it runs through the PHP code when the file is called. It creates a difference in response, and then you need to put a PHP Framework over that. A difference in how you use PHP with libraries in a MVC-pattern could change the response time. This is why we will do what we will do in this paper.

The Choice will be to create this paper as a first step in how to do a performance evaluation on PHP Framework2 because of the lack of data in this area.

6 Experiment Design

6.1 Investigated Question

Which of the following PHP frameworks provide better performance measured in terms of **page speed in ms**.

6.2 Experiment artifacts and variables

Controlled Variables, the following variables are controlled:

Variable	Description
request size	Constant
server	See table 6
framework	Compatible API, look at table 7 for info about the classes.
response size	Constant. See the size in table 8.

Table 4. Controlled variables in Experiment

The tests are as you can see listed below, and 3 tests will be run on database and 3 tests without database for each framework. It makes 6 tests total for one framework, with and without database. the tests are run on concurrency level 5, i.e. sending 5 requests each time, to be sent as fast as the previous requests are done. This is to help count the response time more exactly. You can see how the tests will be done in Table 5.

	Codeigniter		CakePHP		
	Concurrency	Requests	Concurrency	Requests	
Test 1	5	20,000	5	20,000	with database
Test 2	5	20,000	5	20,000	with database
Test 3	5	20,000	5	20,000	with database
Test 1	5	20,000	5	20,000	without database
Test 2	5	20,000	5	20,000	without database
Test 3	5	20,000	5	20,000	without database

Table 5. How the tests will de done.

The total amount of the request of 20,000 is to have something to base the average time on, not too much and not too little. It does not need to be much more because the average will be the same ± 2 ms. That makes unnecessary to have more. But, it took as many as 20,000 requests in order to show the right response time in the test results.

A good thing to know is how the server will be setup, and below is the table of the specification of the server that will be used for the experiment on the Amazon cloud. This is how the Framework classes will be used in the experiment.

System	Version/size
OS	Ubuntu 10.04
Apache	2.2.14
MySQL	5.4
PHP	5.3.2
CPU	1 EC2 Compute Unit[4]
RAM	1.7 GB
HDD	160 GB
bits	64 bits

Table 6. The server specification where the test will be done on.

Type	Codeigniter class	CakePHP class
Controller	CI_Controller	AppController
Model	CI_Model	AppModel
Database	database	database
Form helper	form	Form

Table 7. The different classes to use in the frameworks while coding a blog.

[4] One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. [2]

Manipulated variables Server load measured in terms of **amount of request**.
Manipulated variables

Response Type HTML without database, HTML with database

Response Type is the only thing that will be changed. The database support will be removed or added based on the type of test, making it possible to see the difference in the performance of the core of the framework and the database controller.

Experiment Artifact Environment the server is mostly already set, see Table 6.

Blog is a simple website with post list on the start page and with its own page for each single post. The most important thing about the blog is the index page in the experiment, because it is there the requests will be sent. The focus will be on making a decent list of posts on the blog with title, text body, and date of creation. Both frameworks are using the same database making it simple and thereby lessening the chance for the database to make a bad impact on the performance. You can see the code in Appendix A1 and A2.

Apache benchmark does the same as all tools for performance for a web application does. It is the same idea as sending an amount of requests to the server and counting how long it takes to get the response back. Apache benchmark is an tool for benchmarking and has a reputation to do fair tests.[9]

7 Experiment

7.1 How it was made

Apache benchmark was used to make 20,000 requests on a currency level of 5 to a server. The tests were made with a database and randomly picked which framework to test. The tests without database were run at different times, creating the difference in the ms in the results.

I used the command line below to send the load request to a randomly picked ip-address[5] to the test server on the Amazon cloud on Ireland, both day and night.

```
ab -wn 20000 -c 5 http://11.11.11.11/framework
```

The variables controlled were response size, amount of request, and currency of requests. See the data for the variables in Table 8.

[5] The ip is typed as 11.11.11.11 to make it clear that the server is not running.

[6] Db means database. i.e. the framework with the database. The database was used with models for posts and without db it was just plain html in view with controller used.

[7] Response size is total bytes of data sent from the server in one test on every interval.

Variable	Codeigniter db[6]	Codeigniter	CakePHP db[6]	CakePHP
Response size[7]	6440000 bytes	6440000 bytes	6360000 bytes	6360000 bytes
Requests	20,000	20,000	20,000	20,000
Currency	5	5	5	5

Table 8. The tests with database average request/connection times in ms.

7.2 The results

The experiment was made randomly at different times. You can see how the tests were done in Table 5. The results of the tests you can see in Appendix A3.

The results you can see in Figure 2. The results were almost the same on both, but a small gap between the two frameworks of just a few ms, to be exact it was a gap of 5,34 ms on a average.

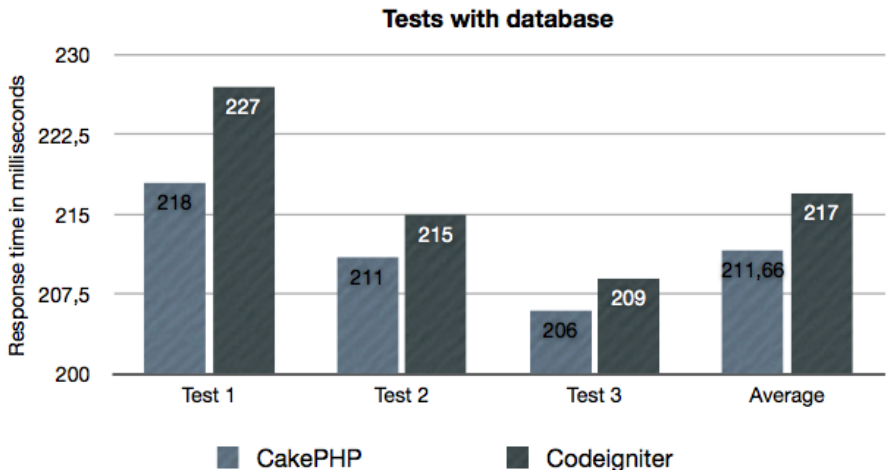

Figure 2. The average test results with database per request/connection times in ms.

This was with database, which show if we will focus to the ms total average, are the tests run with CakePHP faster. This need to be tested without a database too, You can see the result in figure 3.

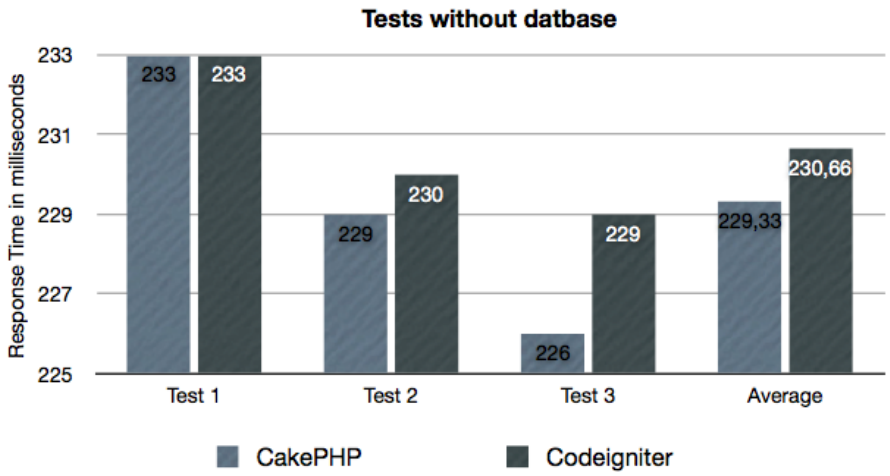

Figure 3. The tests without database average request/connection times in ms.

The tests without database shows that they are almost equally fast, only 1,3 ms gap between them making CakePHP the fastest here, showing CakePHP has faster structure on its core.

7.3 Data analysis and discuss

This was good to see how different the performance is between the frameworks. This is also a very good way to evaluate the performance on any web service or application, the load of request can show how fast the request can be handled by the framework and show the speed of it.

The biggest difference of the performance was when using a database, The difference was **5,34 ms**. It sounds little but in big sites just few ms does matter. When using no database, it was static files of the page generated by the database-version of the blog, with all the posts. The time had the difference of **1,3 ms**. Almost not noticeable at all. And If we take the size in the reflection, is the gap even smaller. Maybe is Codeigniter faster, if they was in same size? That can be true. The default template in CakePHP and Codeigniter was used, making the page in different sizes and could make a difference in the performance of the tests and request.

The difference in milliseconds is so small without database because the frameworks seems to have good structure in their core of controllers and support for views, which we used in the tests without database for both framework.

The difference in milliseconds for the tests with a database is bigger, 5.34 ms. Smaller than a thought but it is a difference. Making it noticeable when having a database and the PHP framework on a big website with many visitors. Creating a big queue of requests, making the small difference bigger, 5.34 ms can feel like the double, 10,68 ms or more. That is why a small difference can be important.

A thing to think about is the size difference, which was around **80,000 bytes**, for both the test with and without the database. This can make gap between the framework understandable with db, but without the db was the tests very close to each others. If the size was the same, Codeigniter could be faster.

The difference in the size is because the use of the default in CakePHP and Codeigniter, CakePHP has more style and a usable template as default for the application, Codeigniter do not have it making it confused of the bigger size of the response. But the size is not a problem because of the choice to use the default of each framework, making the difference in the size.

This does not seem to be the problem, used the default tools in both framework to make a very simple blog webpage. The size should in other word be different, it would be weird if it was not. But the thought that the size can make a difference is of course in the mind. It is request time in ms, and it does have a very big impact by the size.

Would love to see any **future work** on different sizes for the PHP framework, which tests how the size does matter of the performance, with same size of both.

7.4 Validity Threats

Tests are not isolated making the tests to be disturb by something from outside, can be the traffic of the line on Amazon cloud or something else that can make the response time longer or not trusted. This was tried to be reduced by been testing in the night. But you can see in the tests without, that was made more on the morning that the response time is already higher. This should not change the gap between the frameworks because all the tests was tested in the same time, making it having all around same response time in general.

On other hand, a good thought would be that a good performance tests in a web perspective would be to have the webserver and the client on different locations, simulate real production website. Making it a more real data collected.

Concurrency level could be misunderstood which the low level, at 5. A good thought is that the tests is meant to just test how the framework and PHP is handling in normally. normally mean in this case 5 visitors at the same time. This could be bigger or smaller. But these tests was not meant to see how the frameworks are handling the requests in high visitor numbers.

Client's impact could be a problem. The tests were running Apache benchmark, but Apache benchmark is ignoring stylesheet and pictures (The blog in the test did not have pictures). It is just getting the generated html page, after php have generated it with the use of the code of PHP Framework. making the

impact smaller than a normal browser, that is getting the stylesheet and pictures too.

8 Conclusion

The frameworks performance has big difference, but mostly just if in used of database, the classes for using database can in other words be more effective. CakePHP was fastest in both tests, with and without a database. But the size was different on 80,000 bytes. This could be something that could make CakePHP a winner. But my conclusion is that used default template, very simple html page. So it does not have a big impact.

The fastest framework, according to the data is CakePHP, with just 1,3 ms in average without database and with 5,34 ms in average with database. This is something that is a big impact in the PHP performance and it can be fixed in newer versions.

The experiment shows that the evaluation can be done in milliseconds for websites and web applications. With these data is the difference of evaluation between PHP framework not big at all, but it is a difference, even if it is a small one.

The relevance and contribution of this paper is big because of the lack of papers and experiment made on PHP Frameworks. This is one of a new area of technology to test in performance.

Let the war begin between the php frameworks in performance.

References

1. S. Abeysinghe. *PHP Team Development*. Packt Publishing, 2009.
2. Amazon Web Services LLC. Ec2 faq: What is a ec2 compute unit. `http://aws.amazon.com/ec2/faqs/`, April 2012.
3. Cake Software Foundation. Intro to cakephp, what is cakephp. `http://book.cakephp.org/1.1/view/307/Introduction-to-CakePHP`, June 2012.
4. G. Da-gang. Analysis of model-based mvc framework for php development codeigniter. May 2009.
5. EllisLab LLC. Codeigniter changelog and releasedates. `http://codeigniter.com/user_guide/changelog.html`, June 2012.
6. GitHub Inc. Git repository for cakephp. `https://github.com/cakephp/cakephp`, July 2012.
7. GitHub Inc. Git repository for codeigniter. `https://github.com/EllisLab/CodeIgniter`, July 2012.
8. P. Sevcik and J. Bartlett. Understanding web performance. *Business Communications Review*, 31(10):28, October 2001.
9. The Apache Software Foundation. ab: Apache benchmark specification and information. `http://httpd.apache.org/docs/2.2/programs/ab.html`, July 2012.
10. The PHP Group. Php faq what is php and what does it stand for. `http://php.net/manual/en/faq.general.php`, June 2012.
11. L. Titchkosky, M. Arlitt, and C. Williamson. A performance comparison of dynamic web technologies. *ACM SIGMETRICS*, 31(3):2–11, December 2003.
12. S. Widjaja. Web-based ide to create model and controller components for mvc-based web applications on cakephp. December 2010.

Appendix

A1. CodeIgniter Code

controller/blog.php

```
<?php
class Blog extends CI_Controller {
        public function __construct()
    {
        parent::__construct();
        $this->load->model('Posts_model', 'Posts', TRUE);
    }

    function index() {
            $this->data->query = $this->Posts->getPosts(10, 0);

            $this->load->view('blog', $this->data);
    }

    function view($id) {
            $this->data->query = $this->Posts->getPost($id);
            $this->load->view('view', $this->data);
    }
}
?>
```

view/blog.php

```
<html>
<head>
<title>Le Blog</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<link href="/codeigniter/css/cake.generic.css" rel="stylesheet">
<link href="/codeigniter/favicon.ico" type="image/x-icon" rel="icon" />
<link href="/codeigniter/favicon.ico" type="image/x-icon" rel="shortcut icon" />
</head>
<body>
<div style="hight:100px;font-size:18pt;font-family:helvetica,arial;width:100%">
<h1>My simple blog.</h1>
</div>

<div class="container">
<?
if($query->num_rows() > 0) {
        foreach ($query->result() as $row)
        {
```

```
                ?>
                <div style="width:300px;font-size:8pt">
                        <a href="blog/view/<?=$row->id?>">
                                <h2><?=$row->title?></h2>
                        </a>
                <p>Skrevs <?=$row->created?></p>
                <p><?=$row->body?></p>
                </div>
                <?
        }
}
else {
        ?><p>No posts yet.</p><?
}
?>
</div>
</body>
</html>
```

view/view.php

```
<html>
<head>
<title>Le Blog</title>
<link href="/codeigniter/bootstrap/css/bootstrap.css" rel="stylesheet">
</head>
<body>
<div style="hight:100px;font-size:18pt;font-family:helvetica,arial;width:100%">
<h1>My simple blog.</h1>
</div>

<div class="container">
<?php
if($query->num_rows() > 0) {
$row = $query->row();
?>
<h2><?=$row->title?></h2>
<p>Created <?=$row->created?></p>
<p><?=$row->body?></p>
<?
}
?>
</div>
</body>
</html>
```

model/posts_model.php

```php
<?php
class Posts_model extends CI_Model {
        public $id = "";
        public $title = "";
        public $body = "";
        public $created = "";
        public $modified = "";

        function install()
        {
                        /* First, create our posts table: */
                $this->db->query("CREATE TABLE posts (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(50),
    body TEXT,
    created DATETIME DEFAULT NULL,
    modified DATETIME DEFAULT NULL
);");
        }

        function addPost($data) {
            if($data) {
                    $this->db->insert('posts', $data);
            }
        }

        function getPosts($amount, $lastId = 0) {
                if($lastId == 0) {
                        $data = $this->db->query("SELECT * FROM posts ORDER
                         BY ID DESC LIMIT 10");
                }
                else {
                        $data = $this->db->query("SELECT * FROM posts WHERE id
                         <$lastId ORDER BY ID DESC LIMIT 10");
                }
                return $data;
        }

        function getPost($id = 0)
        {
                $data = "";
                if($id != 0) {
                        $data = $this->db->query("SELECT * FROM posts
                         WHERE id ='{$id}' LIMIT 1");
                }
```

```
                return $data;
        }

}
?>
```

A2. CakePHP Code

controller/postsController.php

```
<?php
class PostsController extends AppController {
    public $helpers = array('Html', 'Form');

    public function index() {
        $this->set('posts', $this->Post->find('all'));
    }

    public function view($id = null) {
            $this->Post->id = $id;
        $this->set('post', $this->Post->read());
    }
}
?>
```

view/posts/index.ctp

```
<!-- File: /app/View/Posts/index.ctp -->

<h1>Blog posts</h1>
<table>
    <tr>
        <th>Id</th>
        <th>Title</th>
        <th>Created</th>
    </tr>

    <!-- Here is where we loop through our $posts array, printing out post info -->

<?php foreach ($posts as $post): ?>
    <tr>
        <td><?php echo $post['Post']['id']; ?></td>
        <td>
            <?php echo $this->Html->link($post['Post']['title'],
array('controller' => 'posts', 'action' => 'view', $post['Post']['id'])); ?>
        </td>
        <td><?php echo $post['Post']['created']; ?></td>
```

```
    </tr>
    <?php endforeach; ?>

</table>
```

view/posts/view.ctp

```
<!-- File: /app/View/Posts/view.ctp -->

<h1><?php echo h($post['Post']['title'])?></h1>

<p><small>Created: <?php echo $post['Post']['created']?></small></p>

<p><?php echo h($post['Post']['body'])?></p>
```

model/Post.php

```
<?php
class Post extends AppModel {
        public $validate = array(
        'title' => array(
            'rule' => 'notEmpty'
        ),
        'body' => array(
            'rule' => 'notEmpty'
        )
    );
}
?>
```

A3. The Tests Results

Tests WITH database

This is ApacheBench, Version 2.3 <*$Revision: 655654 $*>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

CODEIGNITER

first test

Server Software:	Apache/2.2.14
Server Hostname:	54.247.130.232
Server Port:	80
Document Path:	/codeigniter
Document Length:	322 bytes
Concurrency Level:	5
Time taken for tests:	911.508 seconds
Complete requests:	20000
Failed requests:	0
Non-2xx responses:	20000
Total transferred:	12100000 bytes
HTML transferred:	6440000 bytes
Requests per second:	21941.67
Transfer rate:	13274.71 kb/s received

Connection Times (ms)

	min	avg	max
Connect:	68	226	38113
Processing:	36	1	1
Total:	104	227	38114

second test

Server Software:	Apache/2.2.14
Server Hostname:	54.247.130.232
Server Port:	80
Document Path:	/codeigniter
Document Length:	322 bytes
Concurrency Level:	5
Time taken for tests:	861.753 seconds
Complete requests:	20000
Failed requests:	0

Non-2xx responses:	20000
Total transferred:	12100000 bytes
HTML transferred:	6440000 bytes
Requests per second:	23208.52
Transfer rate:	14041.15 kb/s received

Connnection Times (ms)

	min	avg	max
Connect:	56	214	38048
Processing:	48	1	0
Total:	104	215	38048

third test

Server Software:	Apache/2.2.14
Server Hostname:	54.247.130.232
Server Port:	80
Document Path:	/codeigniter
Document Length:	322 bytes
Concurrency Level:	5
Time taken for tests:	837.755 seconds
Complete requests:	20000
Failed requests:	0
Non-2xx responses:	20000
Total transferred:	12100000 bytes
HTML transferred:	6440000 bytes
Requests per second:	23873.32
Transfer rate:	14443.36 kb/s received

Connnection Times (ms)

	min	avg	max
Connect:	59	207	20491
Processing:	45	2	0
Total:	104	209	20491

CAKEPHP

first test

Server Software:	Apache/2.2.14
Server Hostname:	54.247.130.232
Server Port:	80
Document Path:	/cakephp

Document Length:	318 bytes
Concurrency Level:	5
Time taken for tests:	873.316 seconds
Complete requests:	20000
Failed requests:	0
Non-2xx responses:	20000
Total transferred:	11420000 bytes
HTML transferred:	6360000 bytes
Requests per second:	22901.23
Transfer rate:	13076.60 kb/s received

Connection Times (ms)

	min	avg	max
Connect:	57	216	38197
Processing:	47	2	0
Total:	104	218	38197

second test

Server Software:	Apache/2.2.14
Server Hostname:	54.247.130.232
Server Port:	80
Document Path:	/cakephp
Document Length:	318 bytes
Concurrency Level:	5
Time taken for tests:	847.232 seconds
Complete requests:	20000
Failed requests:	0
Non-2xx responses:	20000
Total transferred:	11420000 bytes
HTML transferred:	6360000 bytes
Requests per second:	23606.27
Transfer rate:	13479.18 kb/s received

Connection Times (ms)

	min	avg	max
Connect:	58	210	20516
Processing:	46	1	0
Total:	104	211	20516

third test

Server Software:	Apache/2.2.14

Server Hostname:	54.247.130.232
Server Port:	80
Document Path:	/cakephp
Document Length:	318 bytes
Concurrency Level:	5
Time taken for tests:	826.001 seconds
Complete requests:	20000
Failed requests:	0
Non-2xx responses:	20000
Total transferred:	11420000 bytes
HTML transferred:	6360000 bytes
Requests per second:	24213.03
Transfer rate:	13825.64 kb/s received

Connnection Times (ms)

	min	avg	max
Connect:	53	204	11978
Processing:	51	2	0
Total:	104	206	11978

Tests without database

CODEIGNITER

first test

Server Software:	Apache/2.2.14
Server Hostname:	54.247.130.232
Server Port:	80
Document Path:	/codeigniter
Document Length:	322 bytes
Concurrency Level:	5
Time taken for tests:	933.235 seconds
Complete requests:	20000
Failed requests:	0
Non-2xx responses:	20000
Total transferred:	12100000 bytes
HTML transferred:	6440000 bytes
Requests per second:	21430.84
Transfer rate:	12965.66 kb/s received

Connection Times (ms)

	min	avg	max
Connect:	57	232	38060
Processing:	47	1	1
Total:	104	233	38061

second test

Server Software:	Apache/2.2.14
Server Hostname:	54.247.130.232
Server Port:	80
Document Path:	/codeigniter
Document Length:	322 bytes
Concurrency Level:	5
Time taken for tests:	922.767 seconds
Complete requests:	20000
Failed requests:	0
Non-2xx responses:	20000
Total transferred:	12100000 bytes
HTML transferred:	6440000 bytes

Requests per second:	21673.96
Transfer rate:	13112.74 kb/s received

Connection Times (ms)

	min	avg	max
Connect:	59	229	71415
Processing:	45	1	0
Total:	104	230	71415

third test

Server Software:	Apache/2.2.14
Server Hostname:	54.247.130.232
Server Port:	80
Document Path:	/codeigniter
Document Length:	322 bytes
Concurrency Level:	5
Time taken for tests:	918.796 seconds
Complete requests:	20000
Failed requests:	0
Non-2xx responses:	20000
Total transferred:	12100000 bytes
HTML transferred:	6440000 bytes
Requests per second:	21767.63
Transfer rate:	13169.41 kb/s received

Connection Times (ms)

	min	avg	max
Connect:	60	228	37012
Processing:	44	1	0
Total:	104	229	37012

CAKEPHP

first test

Server Software:	Apache/2.2.14
Server Hostname:	54.247.130.232
Server Port:	80
Document Path:	/cakephp
Document Length:	318 bytes
Concurrency Level:	5
Time taken for tests:	935.934 seconds

Complete requests:	20000
Failed requests:	0
Non-2xx responses:	20000
Total transferred:	11420000 bytes
HTML transferred:	6360000 bytes
Requests per second:	21369.03
Transfer rate:	12201.72 kb/s received

Connection Times (ms)

	min	avg	max
Connect:	57	232	20396
Processing:	47	1	1
Total:	104	233	20397

second test

Server Software:	Apache/2.2.14
Server Hostname:	54.247.130.232
Server Port:	80
Document Path:	/cakephp
Document Length:	318 bytes
Concurrency Level:	5
Time taken for tests:	917.710 seconds
Complete requests:	20000
Failed requests:	0
Non-2xx responses:	20000
Total transferred:	11420000 bytes
HTML transferred:	6360000 bytes
Requests per second:	21793.38
Transfer rate:	12444.02 kb/s received

Connection Times (ms)

	min	avg	max
Connect:	54	228	74417
Processing:	50	1	0
Total:	104	229	74417

third test

Server Software:	Apache/2.2.14
Server Hostname:	54.247.130.232
Server Port:	80
Document Path:	/cakephp

Document Length:	318 bytes
Concurrency Level:	5
Time taken for tests:	907.820 seconds
Complete requests:	20000
Failed requests:	0
Non-2xx responses:	20000
Total transferred:	11420000 bytes
HTML transferred:	6360000 bytes
Requests per second:	22030.79
Transfer rate:	12579.58 kb/s received

Connection Times (ms)

	min	avg	max
Connect:	58	225	37924
Processing:	46	1	1
Total:	104	226	37925