Phalcon Framework

Andres Gutierrez-Mendoza
andres@phalconphp.com

Eduar Carvajal-Diaz
eduar@phalconphp.com

This paper is under construction, feel free to comment or criticize it.

Abstract

We describe how the Phalcon Framework is designed and how its optimizations helps developers to get fast applications without increasing development complexity. Phalcon Framework is considerably faster than algorithms based only on PHP, is simple to implement, and results in code that is almost as efficient as that obtained using traditional frameworks. In this paper, we explore a different approach to building frameworks using low-level primitives obtaining the same results as PHP frameworks.

Categories and Subject Descriptors D.3.4 [Programming Languages]:
Web Programming—Performance; optimization
General Terms Design, Experimentation, Performance
Keywords Zend Virtual Machine, compilation, interpretation

1. Preface

For a long time, programmers collected commonly used code in libraries so that code could be reused. This saves development time and reduces errors since reused code only has to be debugged once. Looking for a more professional development in the PHP world, many frameworks have been emerged. A framework provides a well-defined structure and philosophy to create applications easy to maintain.

Phalcon Framework is written in C-language. It takes advantage of the Zend Engine API to interact with PHP internals providing a direct interaction than other PHP frameworks cannot.

For much of the paper, we will talk about extremes conditions of performance, hardware today is really fast, but it is possible to achieve better performance.

In this paper, we present benchmarks showing that our framework performance is considerably greater than traditional frameworks.

2. How PHP works

PHP is a general-purpose server-side scripting language originally designed for Web development to produce dynamic Web pages [1] . It typing discipline is: Dynamic, weak. Its main implementation is based on the Zend Engine. A virtual machine written in C created by company Zend[2].

PHP language is interpreted, Interpretation means that code is never translated to machine code in order to increase performance, instead a specialized interpreter, executes one-by-one each of the required instructions.

[1] http://en.wikipedia.org/wiki/PHP
[2] Zend Engine version 2.0: Feature Overview and Design"
[3] Abstract Syntax Tree Implementation Idioms (Joel Jones,

[2] Zend Engine version 2.0: Feature Overview and Design"

Interpretation provides a easy way to create portable applications across different platforms. PHP interpreter has become more faster each version released. Despite the many optimizations performed through published versions, speed cannot reach other JIT-based or compiled-based counterparts.

The following paragraph describes every phase taken in a PHP execution. Firstly, PHP depends on a Bison/YACC parser that checks PHP code using a previously defined LALR context-free grammar in order to build an internal abstract syntax-tree (AST)[3]. A syntax tree is a syntactic representation of every element of a source code. Every time a PHP source is executed a related syntax-tree is created, this can be as large or small depending on the size of the code to execute. This process is not restricted only to PHP; other interpreted languages have a similar behavior.

Next step, Zend Engine takes the assembled syntax tree and executes from its start point, moving from different traces, altering the flow of the execution till the program end. This code in memory is often called intermediate code (IC)[4]. Every element in the syntax tree is not normally executed because there are decisions and control flow statements that changes the flow depending of the program input. So, when many functions, classes and other program structure elements are parsed by PHP but not executed, only make up a waste of resources and memory.

To attack this problem, from PHP 5, "autoloading" capabilities are available to include and execute only code that is needed by applications. Although its intention is good, PHP frameworks and libraries due to software engineering practicalities, are including more code that is really needed, wasting resources unnecessarily.

These days, a couple of tools self-called code optimizers have been emerged to help PHP to increase performance. These tools aim to enhance the performance of the compiled code by reducing its size, merging redundant instructions and making other changes that can reduce the execution time. Caching intermediate code in shared memory chunks allow to avoid the overhead of parsing and compiling the code every time the scripts run. The use of those tools can improve the overall execution time up to 30%. However, Zend Engine should always take the cached code and interpret them.

When request is finished, the last step is free up all the resources allocated. Depending on the SAPI interface more or less elements are released. Persistent resources are maintained between a request and another.

We will explain some of the most important aspects of PHP and its impact. This is not the occasion to talk in depth about it, but talk about them to offer a better understanding of the paper.

2.1 Memory Management and Garbage Collectors

One advantage of PHP is that developers don't need to manually manage the allocating/freeing of memory at any time. This allows developers to focus on specific problems without dealing with low-level situations like that.

PHP implements also a Garbage Collector (GC) to release unused

[3] Abstract Syntax Tree Implementation Idioms (Joel Jones, University of Alabama)

[4] "How do computer languages work?". Retrieved 2009-11-04.

memory and resources. However, most often, GC is never executed due to small execution times. Unlike other frameworks, Phalcon manages its own memory, even implementing a memory manager on top of Zend Memory Manager. Later it will be discussed.

2.2 Reference counting

Is a technique of storing the number of references, pointers, or handles to a resource such as an object, block of memory, disk space or other resource. PHP uses reference counting to know exactly how many variables are referencing the same value. This approach helps PHP to save memory a lot, moreover, increases the amount of processing required in a minimal.

```
1. $a = 10;
2. $b = $a;
3. $b = 5;
```

The above example is a simple way to understand the concept:

- $a has a value of 10, $a is only a reference to 10
- $b is equal to $a, but, $b is also a reference to 10, 10 now have 2 references, both $a and $b
- $b is not referencing 10 anymore so 10 now have only 1 reference to it, $b is now referencing to value 5

Most of the C data types are very primitive so they don't need any garbage collection stuff.

Understanding reference counting has been crucial for us because a bad implementation of this could produce memory leaks.

2.3 Representing multi-type values

Due to its dynamic type discipline any variable in PHP can have any type supported by the language without getting a warning or error. This can be achieved through an internal structured called zval. This structure has the following representation in C:

```
typedef union _zvalue_value {
 long lval; /*long value */
 double dval;/* double value */
 struct {
      char *val;
      int len;
 } str;
 HashTable *ht;
 zend_object_value obj;
} zvalue_value;

struct _zval_struct {
  zvalue_value value;
  zend_uint  refcount__gc;
  zend_uchar type;
  zend_uchar is_ref__gc;
};
```

A zval have all the possible "slots" to store any value. At the zvalue_value only one of them can have a value because zvals only can have a value at time.

2.4 Hash Tables

A hash table or hash map is a data structure that uses a hash function to map identifying values, known as keys [5]. PHP makes extensive use of Hash Tables to store and retrieve data needed to run the code. Its algorithm is highly optimized for both search and insert. A few examples of the use of Hash Tables in PHP are:

- Register all functions and their arguments
- Register all classes and their methods and properties
- Register variables and their values

We cannot directly access that Hash Table from PHP, instead we should to use interfaces provided by the language such as functions. The improper modification of those structures could

[5] http://en.wikipedia.org/wiki/Hash_table

damage the consistent state producing a wide of fails. Hash Tables should have a consistent state during the entire execution to obtain functioning. Zend API provides C interfaces to work with Hash Tables:

```
ZEND_API int
_zend_hash_index_update_or_next_
insert(HashTable *ht, ulong h,
void *pData, uint nDataSize,
void **pDest, int flag
ZEND_FILE_LINE_DC)

ZEND_API void
zend_hash_copy(HashTable
*target, HashTable *source,
copy_ctor_func_t
pCopyConstructor, void *tmp,
uint size)

ZEND_API void
zend_hash_destroy(HashTable *ht)
```

The ability to access these structures directly helps the framework to be faster. Sometimes omit the need to access them is also an improvement in performance.

2.5 Local Scopes and Global Scope

A scope is the context within a computer program in which a variable name or other identifier is valid and can be used, or within which a declaration has effect. In fact, PHP implement scopes to make values accessible inside functions or global scope. Imagine you have a function with 20 variables defined. PHP needs to create a Hash Table with 20 elements to store/retrieve variables values. Only retrieving a value might need dozens of machine instructions. Although Insert, Update or Search functions are quite fast over Hash Tables, not so much as directly access a C value, which in fact is only a memory address.

2.6. Call stack

A call stack is a stack data structure that stores information about the active subroutines of a computer program. PHP implement its own structures to manage call stacks also providing debug capabilities.

Phalcon can control the call stack by avoiding unnecessary

2.7 C-Extensions

PHP provides an external API to built extensions written in C/C++. These extensions could be compiled together with PHP or be loaded at runtime as dynamic shared libraries. Phalcon is itself a C-extension.

3. Performance of Traditional Frameworks

Phalcon is the first framework completely written in C for PHP. Despite minor micro-optimizations traditional frameworks do not offer any real performance improvement. As frameworks are made more robust and large begin to tend to be heavier in terms of performance.

3.1 Disk Reading

Traditional frameworks should read from disk the code needed to be executed. Many variables are involved at this point, the disc type, how busy the server, the file system, etc. According to benchmarks we have made, some frameworks include more than 100 files per request, which means a larger penalty to application performance.

3.2 Intermediate Cache Reading

If an intermediate code cache is used, the code is readed from a shared memory chunk. This involves copying large amounts of bytes per request into the PHP process. Anyway, the process is very fast and does not feel the difference.

3.3. Entire code is interpreted

At this point any PHP framework is hopeless, all code should be interpreted line by line. A framework can be from hundreds to thousands of lines, each instruction certainly does not have the complexity of the others, but the processes of interpretation are not the best in terms of performance.

It is not common that developers change frameworks and tools they use, only in extreme situations. This is due that frameworks does not provide detailed documentation on how to do. If we want to upgrade to the next version, it is not good to have different files with changes made in them that will be lost.

If installed frameworks never change, Why they must to interpret their hundreds or thousands of lines each time you execute a request?

4. How a C-extension for PHP works

C extensions are loaded one time when the web server starts. Some implementation of web servers could reload the extensions every certain time or when the numbers of requests by thread are reached. Anyway, will be recharged much less often than the traditional code in PHP.

There is a special event in C-extensions called MINIT (module init) it is executed only one time. It registers classes and functions to be ready at any moment by the PHP interpreter. This means you can access any time the functionality provided by the framework without extra overhead.

Extensions are compiled for a specific platform. Compilers can make a lot of optimizations on the compilation process such as machine code optimization, reduce redundancy, loop unrolling etc. The compilation process takes place only one time; often developers do not have the need to do because someone else did for the platform they require.

5. Framework Benchmarks

Before, performance is not the most important thing when developing web applications, but, for example, Google decided to take site speed into account in the search rankings. This is yet another way in which improving web performance will have a positive impact

Benchmarking below shows how efficient Phalcon is when compared with other traditional PHP frameworks. This benchmark is updated as stable versions are released from any of the frameworks mentioned or Phalcon itself.

We made a "Hello World" benchmark looking to identify the minimal load overhead of each framework. Many people don't like this kind of benchmark because real-world applications need more complex features. However, these tests you will understand how much is the minimum time spent by each framework to perform a simple task. A task like this is the minimal task that runs on every request made to an application.

A controller and a view have been created for each framework. Action only gives pass to the view for say "Hello!”

Using the "ab" benchmark tool we sent 1000 requests using 5 concurrent connections to each framework.

Benchmark results may vary from one machine to another and from one operating system to another. However it is expected that the variation is minimal.

5.1 Measures Taken

These were the measurements we take to measure the overall performance of the frameworks:

- Requests per second
- Time across all concurrent requests
- Number of included PHP files on a single request (measured using function get_included_files()
- Memory Usage per request (measured using function memory_get_usage()

5.2 Test Environment

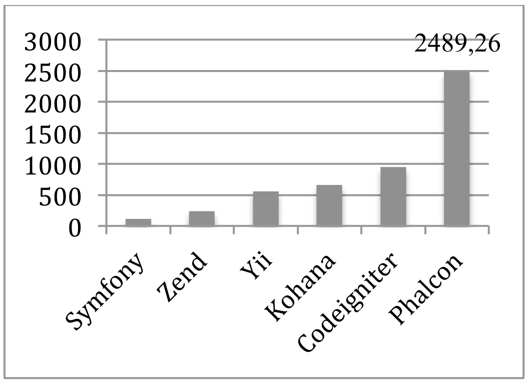
APC intermediate code cache was enabled for all frameworks. apc.stat was set to Off. Any Apache mod-rewrite feature was disabled to avoid possible extra overheads.

The testing hardware environment is as follows:

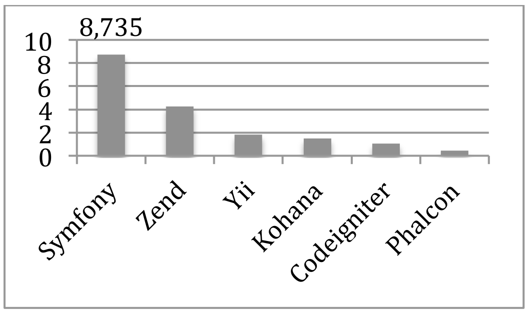
- Operating System: Mac OS X Snow Leopard 10.6.8
- Web Server: Apache httpd 2.2.21
- PHP: 5.3.8
- CPU: 3.06 Ghz Intel Core 2 Duo
- Main Memory: 4GB 1067 MHz DDR3
- Hard Drive: 500GB SCSI/SAS HDD

5.3 Result Graphs

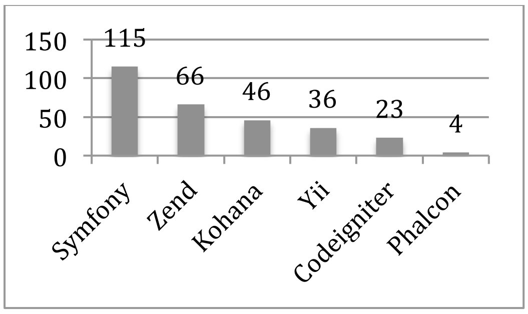
The first graph shows how many requests per second each framework was able to accept.



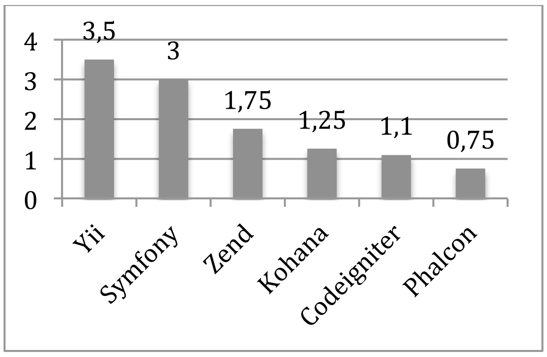
Second shows the average time across all concurrent requests.



Third shows the number of interpreted files in a single request:



Fourth shows the memory consumed in a single request:



6. Low Level Optimizations

In this section we will talk about low-level optimizations made to achieve greater performance:

6.1 Simple Optimizations

Then we will discuss some simple optimizations we have done:

6.1.1 Avoid string counting
Counting strings is very common in the Zend Engine. This process are quite fast, however, we avoid most of this operations by resolving the strings lengths at compile time.

6.1.2 Instantiate classes
Every time a class is instantiated, PHP lookups a zend_class_entry on the global classes hash table. We avoid this behavior by using a global c-pointer directly in the source code. This removes the lookup overhead of find out classes.

6.1.3 Functions cache
When a function is called PHP checks if it exists. Phalcon implements a function cache to have a pointer to a procedure in memory for each PHP function called. So if a function is executed more than once, the code associated with it will be found much faster.

6.1.4 Function to lowercase
Every time a function or method is called out, PHP converts is name to lowercase copying the memory of the name to another chuck even if the string is already in lowercase. Phalcon avoids this by calling functions in a special way.

6.1.5 Visibility checking
Every time a method is called, PHP checks if the current scope has visibility to the required method. This is an overhead when many objects are used in a single request. Phalcon avoids this by calling methods in a special way.

6.2 Advanced optimizations
Following are complex optimizations made to the source code:

6.1.1 SSA-based optimizations [6]

We tried to write the source code using a static single assignment form SSA-form, which says that each variable is assigned exactly once. Not all code can be written that way. We expect that compilers can make better optimizations at compile time.

This kind of optimizations is indented to compilers, so we want to force this improvement by doing this.

GCC compiler already implements this kind of optimization.

6.2.2 Sparse conditional constant propagation [7]

Phalcon improves performance by avoiding the use of expressions that cannot be evaluated in runtime to predict the trace of execution will help the compiler to reduce the final executable object.

6.2.3 Constant Propagation
Since zval are complex structures that cannot be optimized well by compilers, we already added new functions on the top of the Zend API to avoid the internal type comparison improving performance. We implement constant propagation to eliminate the overhead of lookup constants at runtime.

7. Conclusions and Outlook

Where traditional frameworks so far have been quite similar to their extension counterparts, "pretty much doing the same things, just interpreted", our approach is markedly different. Functionality is available to the developer without requiring those

[6] Briggs, Preston; Cooper, Keith D.; Harvey, Timothy J.; and Simpson, L. Taylor (1998). Practical Improvements to the Construction and Destruction of Static Single Assignment Form

[7] Wegman, Mark N. and Zadeck, F. Kenneth. "Constant Propagation with Conditional Branches." *ACM Transactions on Programming Languages and Systems*, 13(2), April 1991, pages 181-210.

techniques like lazy loading or dependency injection.

The Phalcon compiled nature offers extraordinary performance that outperforms all other frameworks in this comparison. Phalcon is a real high performance framework that allows us to scale more easily than ever.

8. References

D. Novillo. Tree SSA, a new optimization infrastructure for GCC. Proceedings of the 2003 GCC Developers' Summit, pages 181–193, 2003

D. Bruening and E. Duesterwald. Exploring optimal compilation unit shapes for an embedded just-in-time compiler. 3rd ACM Workshop on Feedback-Directed and Dynamic Optimization, pages 13–20, 2000.

Mitchell, John C.; Plotkin, Gordon D.; *Abstract Types Have Existential Type*, ACM Transactions on Programming Languages and Systems, Vol. 10, No. 3, July 1988, pp. 470–502

Pierce, Benjamin C. (2002). *Types and Programming Languages*. MIT Press. ISBN 0-262-16209-1. ^ Infoworld 25 April 1983

PHP Accelerator 1.2 (page 3, Code Optimisation)" (PDF). Nick Lindridge. Retrieved 2008-03-28.

McCarthy, J. (April 1960). "Recursive functions of symbolic expressions and their computation by machine, Part I". *Communications of the ACM* **3** (4): 184–195. doi:10.1145/367177.367199. Retrieved 24 May 2010

Cytron, Ron; Ferrante, Jeanne; Rosen, Barry K.; Wegman, Mark N.; and Zadeck, F. Kenneth (1991). "Efficiently computing static single assignment form and the control dependence graph". *ACM Transactions on Programming Languages and Systems* **13** (4): 451–490. doi:10.1145/115372.115320

Zend Memory Manager - Xinchen Hui laruence@php.net **https://wiki.php.net/internals/zend_mm** 2011-11-09

Reference Counting Article - http://blog.golemon.com/2007/01/youre-being-lied-to.html 2007-01-26

Extension Writing: Introduction to PHP and Zend http://devzone.zend.com/303/extension-writing-part-i-introduction-to-php-and-zend/ Sara Golemon 2005-03-01

TSRLMLS Explanation http://blog.golemon.com/2006/06/what-heck-is-tsrmlscc-anyway.html Sara Golemon 2006-06-01