

[Twitter](#) [Facebook](#) [YouTube](#) [LinkedIn](#) [RSS](#)

10 tips to speed up your PHP applications

Author: [Narcisa Zysman](#)

Posted on March 28, 2013 by [Narcisa Zysman](#)

[Like](#)
[Tweet](#)
[Google+](#)
[Linkedin](#)

When you think about [PHP](#) it is often associated with small applications made by passionate developers for their personal use – generally websites with low database usage and/or few visitors.



Well, how wrong we are! PHP is used for a large panel of applications that generate a lot of traffic for example public administrations or big companies. These entities require their applications maintain high scalability, availability, and, of course, no drop of performance. It's no wonder that performance and speed are very popular quality goals when it comes to PHP.

And when it comes to [code quality](#), PHP obeys the same rules as any other programming language, which can lead to a relatively small issue in the source code causing a lot of frustration for the user.

To help avoid that frustration, here are 10 tips to make your PHP applications run faster:

1. Avoid needless copies of your variables

Sometimes developers think they can make the code "cleaner" by copying predefined variables to variables with shorter names before working with them. This sounds like a good idea, but if the variable is altered in any way, the memory consumption is doubled, resulting in the slowdown of scripts. For example, a user inserts 1MB worth of characters into a text area field. This implementation would result in nearly 2MB of memory being used. Of course, less available memory leads to decreased performance for the user.

2. Avoid string concatenations in loops

When placed in a loop, string concatenation results in the creation of large numbers of temporary objects and unnecessary use of the garbage collector. Both consume memory and can dramatically slow down script execution.

3. Avoid passing function variables by reference

In most cases functions only need to use the values passed by their parameters, without altering the values of such parameters. In such cases, you can safely pass those parameters by reference (for example function (&\$parameter) rather than by value (function (\$parameter)). By doing so, you'll

avoid memory-intensive copies and increase application performance.

4. Avoid function tests in loop conditionals

Function tests in loop conditionals decrease application performance.

When you use a loop through an array, the function is called every time. Instead, do a count() beforehand, store the value in a variable, and use it for the test. This way, you can avoid needlessly firing the test function for each iteration.

5. Avoid using relative paths

For better performance, it is highly advised to try and minimize the use of relative paths for file inclusion. The general mechanism for relative path inclusion will search for default include paths then continue with current directory, and so on. In such cases, a file search will take more time.

However, if you need to use relative paths, the best practice is to declare the constant "WEB_ROOT" which defines the root and use it afterwards.

6. Avoid methods with object instantiation in loops

PHP is an object-oriented language and one of the fundamental object-oriented performance management principles is to avoid excessive object creation. This doesn't mean that you should give up the benefits of object-oriented programming by not creating any objects, but you should be wary of object creation inside tight loops when executing performance-critical code.

Object creation is an expensive operation and therefore, when possible, avoid creating temporary or intermediate objects.

7. Avoid using period for 'echo' function

When using the 'echo' function with periods, PHP concatenates the strings before it outputs them, decreasing the performance. Using commas instead of periods outputs them in the declared order, with no extra processing – reducing your chances of hurting application performance.

8. Avoid using regular expressions

Regular expressions are very useful, but also very time-consuming. For this reason, limiting their use is, to say the least, highly recommended. Regular expressions are known to be much slower than their PHP counterparts. For example, use `str_replace` instead of `preg_replace`.

9. Avoid using double quotes for long strings without variables

For strings declared using double quotes, PHP does extra processing to find potential variables which could be used inside those strings.

For example

```
echo 'This is long string'.$name
is faster than
echo "This is long string $name"
```

Combining variable concatenation with single quotes is faster than using double quotes with variables inside the declared string, because no extra string manipulation is needed. If you aren't using variables in your string it's still recommended that you use single quotes. Otherwise, if possible, still use single quoted strings with variables concatenations.


10. Avoid using include_once

The structure `include_once` includes and checks the specified file during the script execution. The functionality is similar to `include`, but in the case of `include_once`, verification is done to check that the file has been included only once. For this reason, `include_once` is more costly than `include` statement. Sometimes it's necessary, but you should default to `include()` in most situations.

These tips are useful only if they are top of mind at all times and checked continually through the [development cycle](#). Of course, it is easier and more efficient when a specialized tool remembers it and checks it for you. [CAST Highlight](#) and [CAST AIP](#) are such tools. They provide those rules and much more, permitting you to do a quick assessment of your application with the former and a deeper analysis with all the features provided by the later.


COMMENTS

2 comments




Add a comment...

Comment



Lesley Paone · Nashville, Tennessee
For example
echo 'This is long string'.\$name.
is faster than.
echo "This is long string \$name".

This breaks your number 7 rule.
Reply · 4 · Like · August 24, 2013 at 12:36pm



Eirik Sletteberg · Spjelkavik videregående skole
Number 4 is wrong: <http://schlueters.de/blog/archives/125-Do-not-use-PHP-references.html>
Reply · Like · December 11, 2013 at 7:31am

Facebook social plugin

Like Tweet Google+ LinkedIn

