Optimiser php

Date de dernière mise à jour : 28/09/2007 à 00:09

PHP est un langage de programmation interprété, c'est à dire que le fichier texte contenant le code PHP est analysé puis traité directement (pas de code compilé). Nous allons voir comment améliorer les performances de vos scripts PHP pour tirer le maximum de performances.

Les différentes techniques d'optimisation que nous allons voir ici vous permettront :

-De générer vos pages plus rapidement pour le visiteur
-D'économiser des ressources serveur
-D'accueillir plus de visiteurs en même temps sur votre site
-De coder plus proprement, car optimisation rime souvent (pas toujours) avec clarté et propreté.

Suite aux différentes remarques plus ou moins fondées que j'ai pu lire sur le net, je tiens à préciser que le début de cet article ne vous fera pas gagner un temps de génération énorme si votre page est mal codée et qu'il s'agit de micro-optimisations, ça ne remplace donc en AUCUN cas un algorithme optimisé et ce pour n'importe quelle page de votre site. Privilégiez donc une amélioration de vos algorithmes avant de finir par optimiser les quelques tournures qui pourraient vous faire perdre quelques millièmes de secondes supplémentaires.

Nous allons donc commencer par optimiser les quelques millisecondes que vous pourriez perdre par l'emploi de fonctions moins efficaces que d'autres. Tous les benchs qui vont suivre ont été effectués avec la version 5.2.4 de PHP, la dernière au moment où j'écris ces lignes. Ils consistent en l'exécution d'une boucle comptant un nombre **n** d'itérations (ce nombre **n** est précisé en face de chaque bench car les différences sont parfois tellement minimes qu'elles exigent un nombre d'itérations plus important afin que l'on puisse les constater). A l'intérieur de cette boucle se trouve le code pour lequel on teste les performances. A la fin du script, on décompte le nombre de secondes qu'il a nécessité pour s'exécuter et on compare donc les différents codes entre-eux. Les benchs ont été exécutés 5 fois de manière à prendre la valeur moyenne du temps de génération et limiter les erreurs. A l'attaque !

Les simples et doubles quotes :

Commençons avec un classique en PHP à savoir les simples et doubles quotes (guillemets). Elles tiennent leur différence principale dans le fait que tout ce qui se trouve à l'intérieur des guillemets simple n'est pas interprété, contrairement à ce qui se trouve à l'intérieur des guillemets doubles. En pratique le code suivant fait exactement la même chose :

```
<?php
echo
   'Voici une chaîne de test'
;
?>

<?php
echo
    "Voici une chaîne de test"

;
?>
```

Par contre le code suivant donnera deux résultats totalement différents :

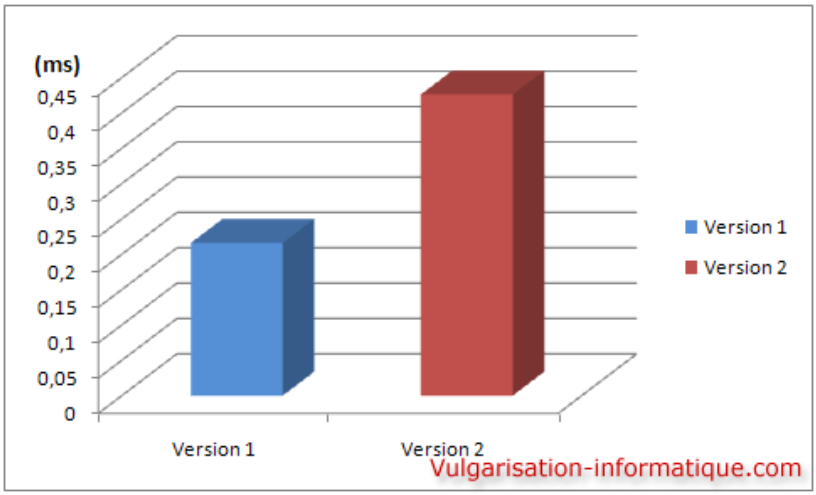
Version 1:

```
<?php          $mavariable
=
'bonjour tout le monde'

;          echo
'Valeur de la variable    : $mavariable'

;
?>
```

Version 2:

```
<?php              $mavariable
=
'bonjour tout le monde'

;                  echo
"Valeur de la variable       : $mavariable"

;
?>
```

Résultats avec 200 itérations :



Lorsque vous utilisez la deuxième version de cet exemple, vous constatez que la chaîne **$mavariable** est remplacée par sa valeur, contrairement à ce qui se passe dans la première version de cet exemple. Or, remplacer $mavariable par sa valeur nécessite du temps supplémentaire pour PHP car il doit non seulement rechercher le nom exact de la variable mais ensuite la remplacer par sa valeur. C'est pourquoi il est plus rapide d'utiliser la

version 1. Bien me direz-vous, mais comment faire si on souhaite afficher le contenu d'une variable ? et bien on va pouvoir utiliser la concaténation. Reprenons notre code lent et optimisons-le :

Version 1:

```
<?php        $mavariable
=
'bonjour tout le monde'

;            echo
"Valeur de la variable     : $mavariable"

;
?>
```

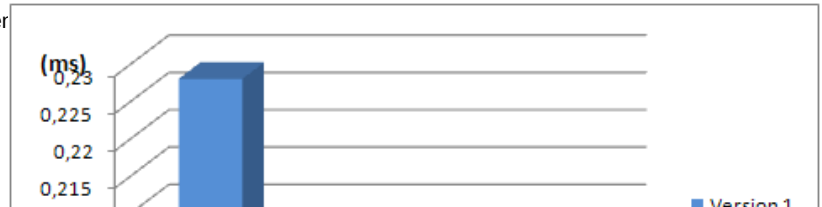
Version 2:

```
<?php           $mavariable
=
'bonjour tout le monde'

;           echo
'Valeur de la variable    : '
.
$mavariable

;
?>
```

Version 3:

```
<?php           $mavariable
=
'bonjour tout le monde'

;           print
'Valeur de la variable    : '
.
$mavariable

;
?>
```

Nous pouvons également (cf version 3) utiliserla fonction print. Elle a une différence avec echo cependant : elle renvoie toujoursla valeur 1, ce qui fait que vous pouvez l'utiliser dans des conditions ou autre.En pratique, elle est supposée être plus lente. Voici une comparaison des temps de gér



On constate qu'il n'y a ici aucune différence moyenne entre print et echo (cependant attention, j'ai constaté que print pouvait être plus rapide que echo, mais c'est généralement l'inverse. La moyenne a lissé les résultats). Vous pouvez (et uniquement pour **echo**) remplacer le point de concaténation par une virgule. Ceci peut se révéler plus rapide (sur 200 itérations je n'ai pas obtenu de différences suffisantes toutefois). Cela vient du fait qu'il est souvent plus rapide de concaténer un petit bout de chaîne et de tout envoyer dans le même buffer que de fermer et réouvrir le buffer nécessaire à chaque fois que vous utilisez echo. Voici en quoi cela consiste :

Concaténation avec le point :

```
<?php	$mavariable
=
'bonjour tout le monde'

;		echo
'Valeur de la variable	: '
.
$mavariable

;
?>
```

"Concaténation" avec la virgule :

```
<?php		$mavariable
=
'bonjour tout le monde'

;		echo
'Valeur de la variable	: '
,
$mavariable

;
?>
```

Après ce bref passage autour de la fonction print et de la structure de langage echo, attaquons-nous aux différentes manières de codage. Commençons par la déclaration (et l'initialisation) des variables. Il existe plusieurs moyens de le faire :

Version 1:

```
<?php	$mavariable
;
$mavariable2

;
$mavariable
=
```

'valeur'

;
$mavariable2
=
'valeur'

;
?>

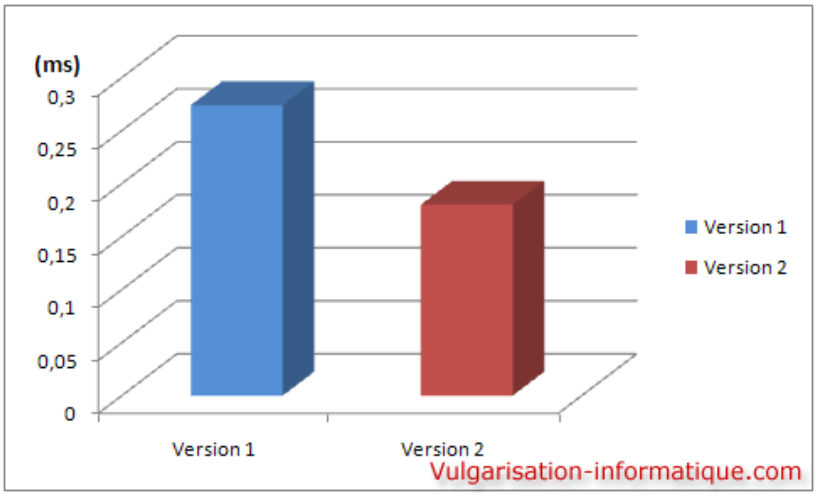
Version 2:

<?php $mavariable
;
$mavariable2

;
$mavariable
=
$mavariable2
=
'valeur'

;
?>

Voyons laquelle des deux est la plus rapide (toujours sur 200 itérations) :



Contrairement à PHP4 où j'avais obtenu des résultats plus rapides pour la version 1, c'est ici la version 2 (plus logique) qui l'emporte d'une courte tête. Passons maintenant aux conditions. Il existe trois manières de faire des conditions en PHP :

-La condition if/else
-L'instruction Switch
-L'opérateur ternaire

Les trois permettent de faire la même chose avec plus ou moins de syntaxe. Voyons trois exemples qui font absolument la même chose mais codés de manière différente :

Version 1:

```
<?php $variable
=
1

;
$i
= 


; if(
$variable
===
1

) {
$i

++; } elseif(
$variable
===
2

) {
$i
=
2

; } else {
$i
=
3

; }
?>
```

Version 2:

```
<?php $variable
=
1

;
```

$i
=

;
switch(
$variable

)
{
case
1

:
$i

++;
break;
case
2

:
$i
=
2

;
break;
default:
$i
=
3

;
break;
}
?>

Version 3:

<?php
$variable
=
1

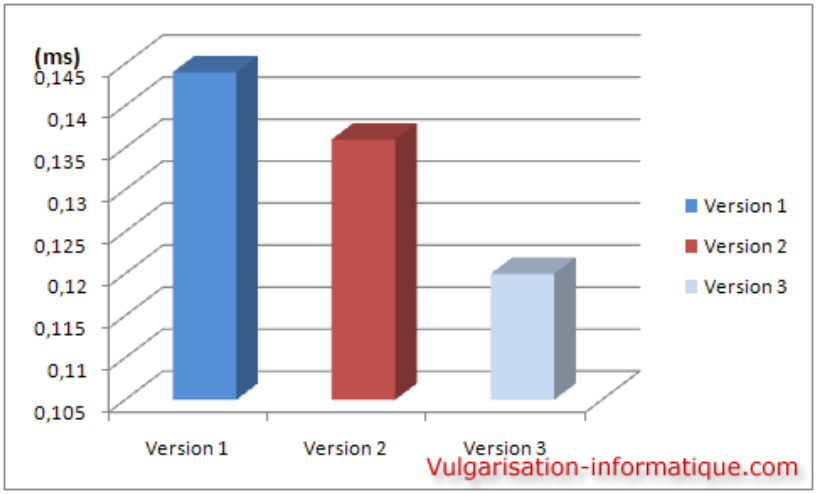
;
$i
=

;
(
$variable
===
1
) ?
$i
++ : ((
$variable
===
2
) ?

$i
=
2
:
$i
=
3

);
?>

La première version utilise l'opérateur classique en PHP, à savoir la condition if/else. La seconde version utilise un switch tandis que la dernière utilise ce qu'on appelle l'opérateur ternaire. Les trois solutions ont des avantages et inconvénients différents du point de vue maintenance notamment, mais nous ne sommes ici que pour juger de leurs performances ! Voici donc les résultats du test, effectué encore une fois sur 200 itérations :



Test encore une fois étonnant, car il est le contraire de ce que j'obtenais en PHP4. La solution la plus rapide est donc l'opérateur ternaire bien que sa lisibilité soit absolument affreuse !

Passons maintenant aux boucles, et plus précisément à l'utilisation de la fonction **count()** dans une boucle. Vous êtes nombreux à mettre un **count()** dans une boucle for de manière à ce que cette boucle s'arrête une fois que la variable à incrémenter dépasse la valeur retournée par la fonction **count()**. Or, en général, vous travaillez sur un tableau de dimension fixe, c'est à dire que **count()** va retourner à chaque fois la même valeur. Lorsque vous faites un tour de boucle, la valeur du **count()** est donc recalculée pour rien ! Voici les deux codes dont je veux parler :

Version 1:

<?php $w
=

;
$tableau
= array(
'test'
,
'hop'

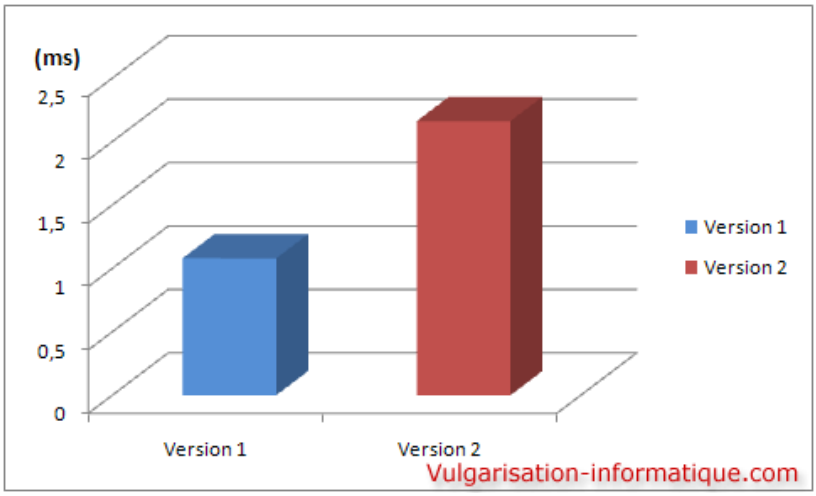
```
);
$n
    =
count
(
$tableau

);                  for(
$i
=

;
$i
$n
;
$i

++)       {
$w

++;         }
?>
```

Version 2:

```
<?php       $w
=



;
$tableau
    = array(
'test'
,
'hop'

);              for(
$i
=

;
$i
count
(
$tableau
);
$i

++)       {
$w
```

++; }
?>

Voici maintenant les résultats pratiques de 200 itérations :



Placer un **count()** dans une boucle est ici deux fois plus lent ! On privilégiera donc la première solution. Dans le même genre de chose, voyons voir l'impact que peut avoir d'utiliser une boucle for ou une boucle while :

Version 1:

<?php $i
=

; while(
$i
50

) {
$i

++; }
?>

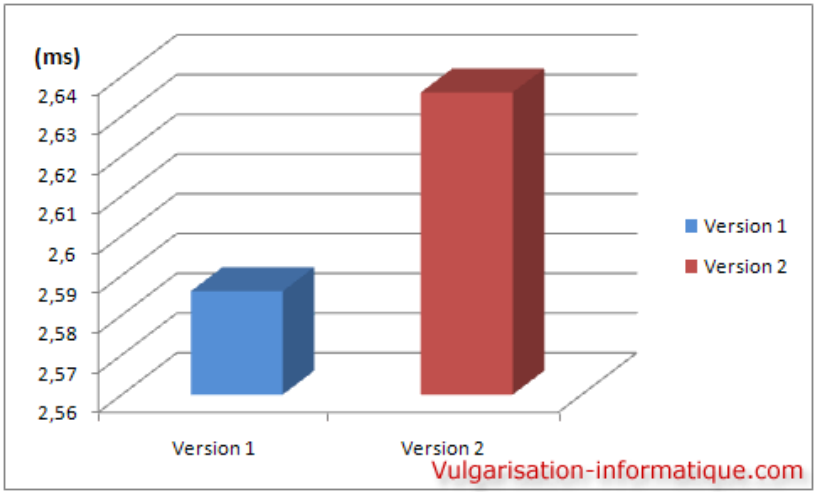
Version 2:

<?php

for(

```
$i
=

;
$i
50
;
$i

++)       {       }
?>
```

Comparons maintenant les résultats de ces deux codes réalisant la même fonction (attention, en cas de code plus complexe, le for peut s'avérer bien utile car il accepte des conditions supplémentaires), toujours sur 200 itérations :



Comme on peut le constater, le while est légèrement plus rapide.

Ouverture d'un fichier:

Il y a beaucoup de fonctions permettant en PHP d'ouvrir un fichier pour lire son contenu. Nous allons tester trois fonctions qui ouvriront un fichier contenant une quinzaine de lignes. Voici les trois fonctions utilisées :

file_get_contents()
file()
fopen()- suivi de **fread()**- pour récupérer les données et **fclose**- pour fermer le handle.

Voici les codes utilisés :

Version 1:

```
<?php     $fichier
=
file_get_contents
(
```

```
'bench.txt'

);
?>
```

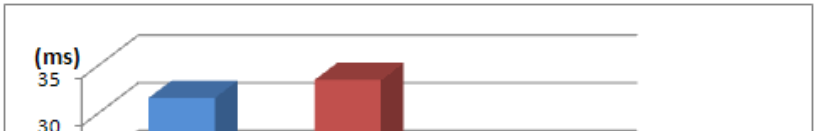
Version 2:

```
<?php            $fichier
=
file
(
'bench.txt'

);
?>
```

Version 3:

```
<?php        $fp
=
   fopen
(
'bench.txt'
,
'r'

);
$fichier
=
fread
(
$fp
,
filesize
(
'bench.txt'

));
fclose
(
$fp

);
?>
```

Nous utiliserons toujours une boucle de 200 itérations. Voici les résultats (sans appel) : la fonction **fopen()** est de loin la plus efficace. La fonction **file()** nécessite la création d'un tableau et est de ce fait plus lente, de peu devant la fonction **file_get_contents()** qui malgré sa simplification n'est pas plus rapide.



Voyons un peu maintenant deux méthodes pour concaténer une valeur à une variable chaîne. Imaginions que vous ayez une variable contenant la valeur **bonjour** et que vous souhaitiez qu'elle contienne la valeur **bonjour tout le monde**. On peut utiliser trois méthodes, l'une d'entre-elles est cependant inintéressante puisqu'elle consiste à réassigner entièrement la nouvelle valeur à la variable :

```
<?php        $variable
=
   'bonjour'

;
$variable
     =
'bonjour tout le monde'

;
?>
```

Nous n'allons donc pas l'utiliser. Voyons maintenant deux autres méthodes utilisant la concaténation ou la recopie de valeur :

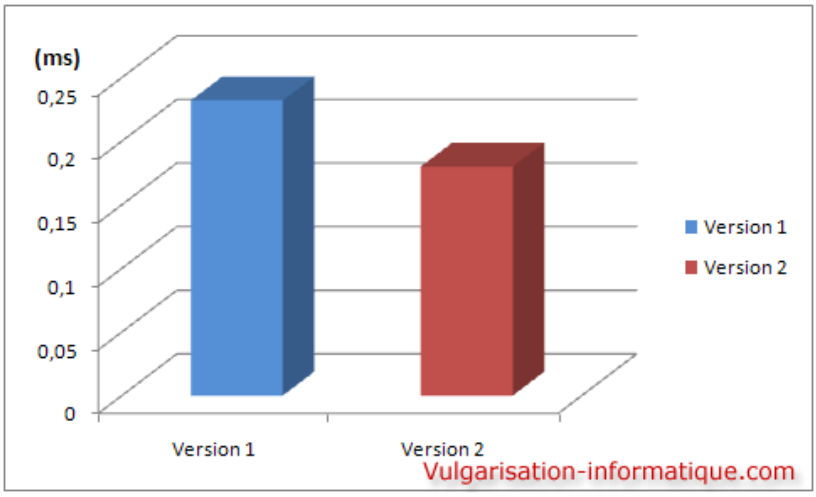
Version 1:

```
<?php       $variable
=
  'bonjour'

;
$variable
    =
$variable
.
' tout le monde'

;
?>
```

Version 2:

```
<?php       $variable
=
  'bonjour'

;
$variable
    .=
' tout le monde'

;
?>
```

Voyons maintenant le résultat des tests effectués encore et toujours avec une boucle de 200 itérations :



La concaténation l'emporte de peu, mais elle marque un net avantage en fonction de la longueur de la chaîne initiale. Plus la chaîne initiale est longue, plus la concaténation l'emporte aisément sur la recopie.

Remplacement d'une expression dans une chaîne :

Il existe de multiples façons en PHP de remplacer une portion de chaîne par une autre valeur. Voici les deux fonctions principalement utilisées pour cela :

preg_replace()
str_replace()

Sachez qu'il est plus rapide d'utiliser **str_replace()** que **preg_replace()**, car **str_replace()** ne prend qu'une chaîne fixe en paramètre, contrairement à **preg_replace()** qui utilise les expressions régulières. Bien sûr dans le cas où vous souhaitez remplacer une chaîne selon une condition bien précise, la fonction **str_replace()** n'est généralement pas assez puissante et il vous faudra recourir à la fonction **preg_replace()**.

Parcours d'un tableau :

Préférez l'utilisation de la fonction **foreach()** à la place de l'utilisation du couple **while(list() = each())**. Nous allons pour le prouver remplir un tableau de 1000 valeurs aléatoires, puis nous allons parcourir ce tableau 200 fois des deux façons que je viens de mentionner. Voici les codes utilisés :

Version 1:

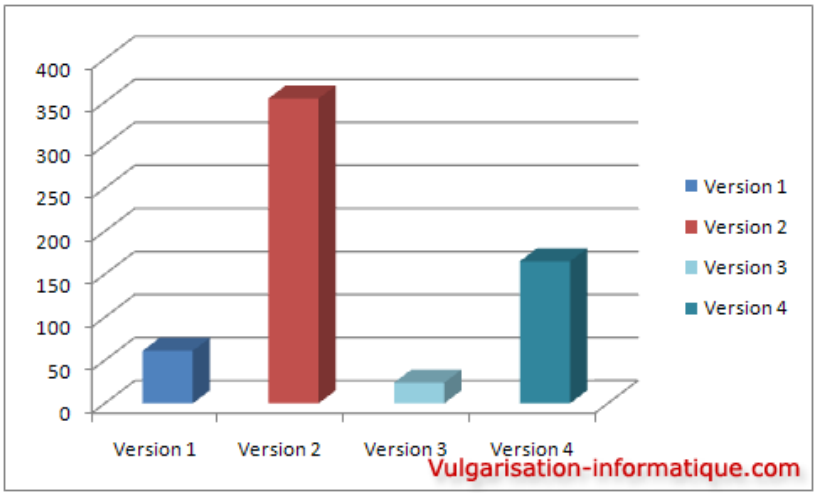
```
<?php	$w
=

;	foreach(
$tableau
AS
$cle
=>
$valeur
)	{
```

```
$w
++;         }
?>
```

Version 2:

```
<?php       $w
=

;           while(list(
$cle
,
$valeur
) =
each
(
$tableau
))          {
$w
++;          }
?>
```

Voici les résultats obtenus :



Le **while()** combiné au **list()** est plus rapide que la fonction **foreach()**.

Après toutes ces micro optimisations, nous allons maintenant parler d'autres méthodes pour optimiser vos scripts PHP de manière à améliorer cette fois-ci de manière conséquente les performances. Ces modifications sont plutôt orientées algorithmes ou bases de données. Pour consulter quelques règles d'optimisation de MySQL, vous pouvez consulter ce lien : Optimiser MySQL.

En PHP, il existe de nombreuses solutions destinées à simplifier la tâche des designers et développeurs. Elles peuvent non seulement leur faciliter la tâche, mais également accélérer le temps de génération de vos scripts si elles sont bien utilisées. On peut parler notamment des moteurs de

templates, dont vous trouverez une version conçue par mes soins ici (en PHP5) : Moteur de templates PHP. Grâce à l'utilisation de ce genre de sources, vous disposerez la plupart du temps d'un mode "cache". Cacher les informations consiste à les enregistrer dans un fichier ou une base de données de manière à réduire les traitements nécessaires pour afficher les données la prochaine fois. Par exemple, lorsque vous affichez la liste de vos actualités, vous savez qu'elle ne sera pas mise à jour toutes les minutes, il est donc inutile de redemander à la base tout le contenu des actualités et d'effectuer des traitements dessus à chaque fois qu'un visiteur clique sur la page. Vous pouvez récupérer une bonne fois pour toutes cette liste, la mettre en cache (dans un fichier texte par exemple) et la réutiliser ensuite au lieu de faire appel à la base de données. Ce genre d'astuce peut très facilement diminuer de **50 %** le temps de génération de vos pages et consommera moins de ressources CPU (au mépris parfois de la quantité de mémoire vive consommée). Ces scripts et méthodes d'optimisation prennent toute leur importance dès lors que vous avez un site web un minimum visité. Si vous avez dix visiteurs journaliers et que vos traitements ne sont pas trop lourds, il est inutile de mettre en place un système de cache, mais qui peut le plus peut le moins comme on dit !