

APC : un cache d'OPCodes pour PHP

par Brian Shire Julien Pauli ([Tutoriels](#), [article et conférences PHP et developpement web](#)) ([Blog](#))

Date de publication : 11/09/2008

Dernière mise à jour : 4/11/2008

Le cache d'OPCodes permet des économies de traitements divers dans le coeur de PHP. Il augmente la vitesse générale de traitement d'une requête par PHP, et il est souvent la solution d'optimisation la plus simple à mettre en place. Cet article va expliquer comment installer, configurer et gérer un cache OPCodes pour PHP : Alternative PHP Cache (APC).

Cet article est une traduction de  [Optimizing with APC](#), par  [Brian Shire](#)


I - Introduction à APC.....	3
I-A - Installation et configuration.....	3
II - Comment fonctionne un cache d'OPCodes ?.....	4
II-A - En quelques mots.....	4
II-B - de l'OPCode ?.....	4
II-C - Fonctionnement technique avancé.....	4
II-C-1 - Une table géante.....	6
II-C-2 - Stat ou pas stat ?.....	6
II-C-3 - TTL.....	6
II-C-4 - Remplissage du cache au démarrage.....	6
II-C-5 - Filtrage.....	7
II-C-6 - RFC1867, progression de l'upload de fichier.....	7
III - API de APC et cache utilisateur.....	8
IV - Conclusions.....	9


I - Introduction à APC

Les caches d'OPCodes permettent des économies de travail coté serveur, ce qui se manifeste par une accélération de la réponse des pages PHP pour l'utilisateur final.


Il représente en plus une des solutions les plus simples à mettre en place dans ce but.

APC est "l'Alternative PHP Cache", un cache libre, gratuit et robuste pour mettre en cache et optimiser le code intermédiaire PHP aussi appelé "OPCode".

APC est activement maintenu dans  **PECL** et offre non seulement un cache OPCode mais aussi un cache utilisateur. Il est configurable et facilement installable.

 *D'autres solutions de cache existent, vous les trouverez en conclusion. Cet article se base sur une installation de PHP en environnement Apache module (mod_php).*

I-A - Installation et configuration

L'installation se fait grâce à la commande `pecl install apc`. Vous pouvez aussi l'installer  **depuis ses sources** avec les commandes `phpize`, `./configure`, `make` et `make install`.

Les utilisateurs de Windows se tourneront vers la DLL téléchargeable depuis  **Pecl4Win**.

Dans les sources se trouve aussi un fichier appelé `apc.php`. Ce fichier est à installer sur le serveur et à interroger comme une vulgaire page PHP. Il s'agit du fichier qui permet **le monitoring du cache**.

Après l'installation du cache, un appel à `phpinfo()` devrait confirmer que celui-ci est bien installé.

APC rajoute des options dans votre fichier `php.ini`, nous les passerons en détail plus tard, assurez vous simplement que celle appelée **apc.enabled** est bien sur la valeur **On**.

Vous pouvez dès lors vérifier le gain de performances sur une application avec un outil comme **ab** (Apache Bench). La directive intéressante à monitorer sera le nombre de requêtes par seconde qu'Apache peut servir. En théorie, ce nombre devrait augmenter de manière plus ou moins significative en fonction de l'application concernée et de l'OS utilisé. Nous allons voir comment fonctionne le cache d'OPCode afin de mieux comprendre ce que l'on est en droit d'attendre de lui.

apc

APC Support	enabled
Version	3.1.0-dev
MMAP Support	Disabled
Locking type	File Locks
Revision	\$Revision: 3.151 \$
Build Date	Nov 8 2007 15:17:30

II - Comment fonctionne un cache d'OPCodes ?

II-A - En quelques mots


Une partie importante du temps de traitement d'une page, servie via HTTP avec un langage de script dynamique comme PHP, provient du fait que ce langage doit transformer un code lisible par un humain, en un code exécutable par une machine virtuelle et le microprocesseur de la machine physique.

Des processus lourds interviennent alors, comme l'analyse syntaxique du code source et sa transformation en un code binaire compréhensible par la machine.

Ceci étant coûteux, beaucoup de langages mettent ce code binaire en cache lors du traitement de la première requête, afin de sauter toutes ces étapes les fois suivantes.

APC fonctionne par interception de la phase de compilation en la remplaçant par une étape de récupération de l'OPCode déjà généré précédemment et mis en mémoire. On peut voir ceci comme un compilateur de code intelligent.

II-B - de l'OPCode ?

Les OPCodes sont des structures de données C qui peuvent être interprétées par la machine virtuelle PHP : Zend Engine. L'extension PHP  **Vulcan Logic Disassembler** (VLD) permet de prendre connaissance de ce code intermédiaire. Elle est utilisée par les développeurs de PHP afin d'optimiser leurs algorithmes et a été créée par l'un d'entre eux : Derick Rethans.

code source PHP

```
<?php
$output = 'Hello World';
if($_GET['exclaim']) {
    $output .= '!';
} else {
    $output .= '.';
}
echo $output;
?>
```

OPCode PHP

```
1 ASSIGN      !0, 'HELLO+WORLD'
2 FETCH_R GLOBAL $1, '_GET'
3 FETCH_DIM_R $2, $1, 'exclaim'
4 JMPZ        $2, ->6
5 ASSIGN_CONCAT !0, '%21'
6 JMP         ->7
7 ASSIGN_CONCAT !0, '.'
8 ECHO        $0
9 RETURN     1
10 ZEND_HANDLE_EXCEPTION
```

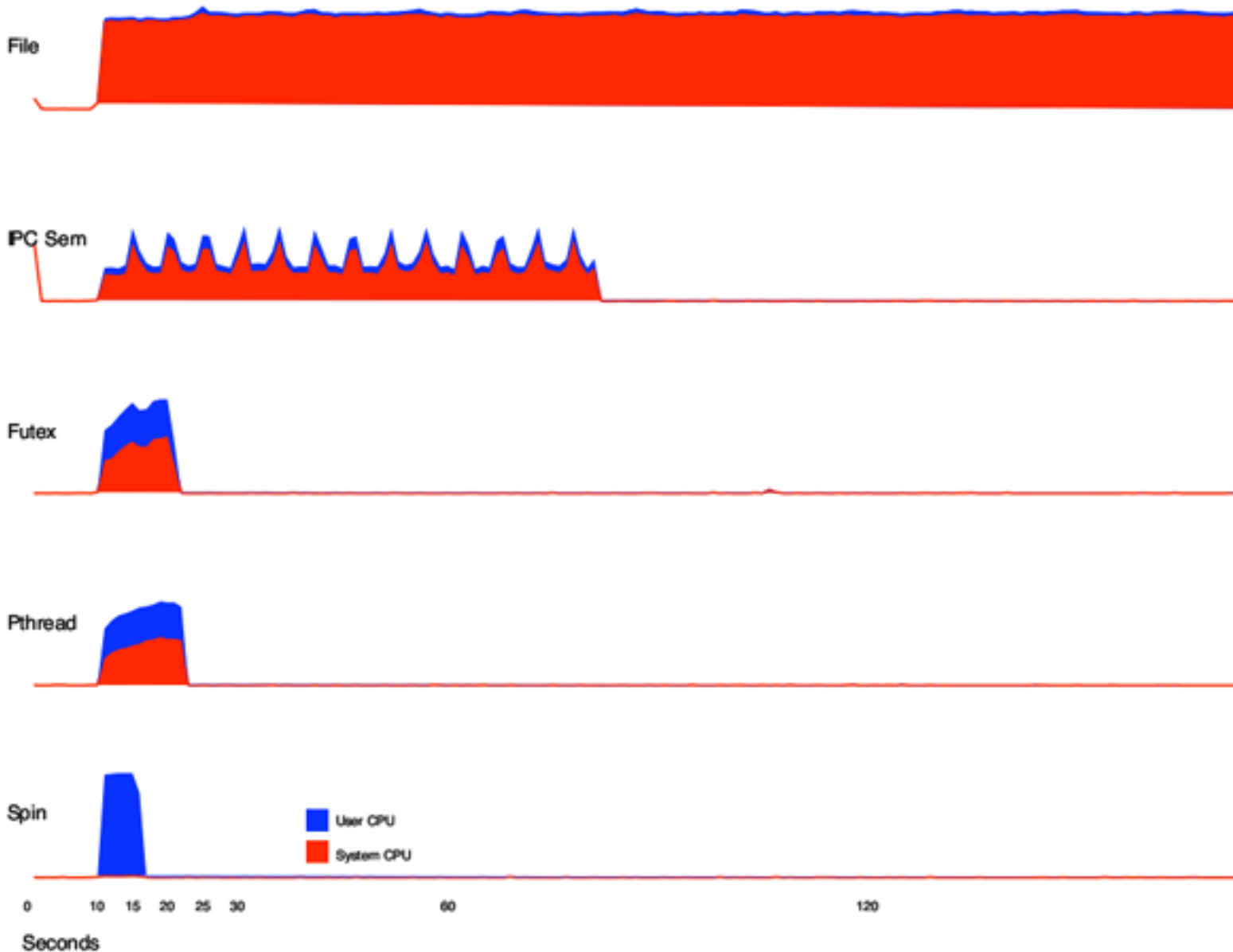
Comme on peut le voir, les OPCodes utilisent toujours plus de place que le code PHP, environ 25 à 33%. Cette remarque est importante lorsqu'il s'agira de configurer le cache, car sa taille sera alors à calculer pour le code intermédiaire et non le code PHP.

II-C - Fonctionnement technique avancé

APC stocke les opcodes dans un segment de mémoire partagée pour accéder rapidement aux données. Cette mémoire étant partagée entre les processus du serveur, un mécanisme de verrou doit être mis en place afin de s'assurer de l'exclusivité des accès modifiant les données.

Ceci crée un goulot d'étranglement de performances important. Les anciennes versions d'APC utilisaient un verrouillage au niveau des fichiers, ce qui était alors très fiable, mais avec un coût en performances d'une importance extrême.

Les versions récentes d'APC utilisent les verrous pthread qui sont standards sur la plupart des matériels. D'autres solutions de verrouillage sont aussi disponibles comme l'Inter-Process Control (IPC), les verrous Linux Futex et plus récemment, les verrous tournants (spin locks) en provenance du projet PostgreSQL. Voici un graphique qui représente un benchmark des différentes solutions de verrouillage en environnement stressé :



Comme le système pthread est très utilisé dans d'autres projets, il se place comme une solution fiable et choisie dans les versions actuelles d'APC comme défaut.

Les verrous tournants ont été portés du projet PostgreSQL et peuvent donner de bien meilleurs résultats que le système pthread. Cependant leur support dans APC demeure expérimental étant donné des problèmes de verrouillage rencontrés avec des signaux d'exécution de PHP. Une solution est en cours d'élaboration, elle devrait proposer un support stable des verrous tournants dans le futur.

II-C-1 - Une table géante

APC peut être vu comme une table de partage géante en mémoire, entre tous les processus du serveur web. Comme toutes les tables mémoire, ses performances proviennent directement de la gestion des collisions. La taille de la table devrait être proportionnelle au nombre d'éléments à y stocker.

APC propose pour cela 3 directives de configuration :

- 1 `apc.shm.size` : Taille du cache en Mo.
- 2 `apc.num_files_hint` : La valeur du nombre maximum de fichiers à cacher est conseillée.
- 3 `apc.user_entries_hint` : La valeur du nombre maximum d'entrées utilisateur (cache utilisateur) à stocker est conseillée.

Il n'est pas conseillé de tenter de leurrer APC en doublant (par exemple) ces valeurs. APC double déjà en interne les valeurs spécifiées, et le gain en performances est fortement réduit avec des valeurs non adaptées.

II-C-2 - Stat ou pas stat ?

Contrairement à la mémoire vive, les disques durs sont extrêmement lents, et devraient être évités le plus possible dans les solutions de cache. Malgré le fait qu'APC charge les OPCodes depuis la mémoire directement, il accède au disque pour être certain que le fichier source n'a pas été modifié, ce qui déclencherait une procédure de mise à jour du code en cache. Sans cette vérification, un changement du fichier source n'aurait aucun effet sur le serveur. Si cependant, la mise à jour des fichiers sources est rare, l'étape de vérification sur disque (stat) peut être évitée. Ceci rend le serveur presque indépendant du disque dur physique. L'option `apc.stat` gère ce paramètre, mise à false, elle évite tout appel disque. Ceci évite l'appel disque, ainsi que l'appel système nécessaire. Le gain en performances devient important pour les applications possédant un nombre de fichiers sources significatif (Frameworks).

Au contraire, certaines applications nécessitent des mises à jour fréquentes des sources. Cela se fait, en général, au travers d'outils tels que `rsync`, `cvs` ou `svn`. Certains de ces outils altèrent la date de modification des fichiers, vers une date passée. Si c'est le cas, `apc.stat_time` doit être mis à true pour forcer APC à se baser sur les dates de création des fichiers, et non plus sur les dates de modification.

II-C-3 - TTL

APC propose deux configurations du TTL (Time To Live) : une configuration utilisateur, et une configuration par défaut dite de secours. Le TTL limite le temps pendant lequel une valeur demeure en cache, ce mécanisme permet de s'assurer qu'aucune donnée périmée ne sera chargée du cache alors qu'une donnée plus fraîche existe.

`apc.gc_ttl` contrôle le TTL du garbage collector. Comme de multiples processus peuvent accéder en même temps à une donnée, celle-ci ne doit être détruite que lorsque tous les processus l'ont relâchée. Si un processus meurt sans relâcher sa référence à la donnée, alors APC ne pourra jamais déterminer si celle-ci est périmée ou valide. `gc_ttl` est utilisé pour contrôler combien de temps APC doit attendre avant d'effacer une donnée, même si celle-ci demeure accédée par un ou plusieurs processus. Typiquement, on spécifie une valeur haute de manière à être sûr de ne pas effacer une donnée du cache toujours valide.

II-C-4 - Remplissage du cache au démarrage

Le cache APC est vidé à chaque (re)démarrage du serveur, ainsi les performances seront impactées par son remplissage lors des premières requêtes. APC ne donne des résultats pertinents que lorsqu'il est bien rempli.

Remplir le cache au démarrage est une technique qui contourne ce problème en insérant les données (c'est-à-dire les variables, et les fichiers PHP) dans le cache avant qu'il ne traite la première requête HTTP via le serveur.

La commande `iptables` sous Linux est utile dans ce but. Il faut bien se rendre compte que le remplissage doit être fait via un processus du serveur Web, et non un client PHP quelconque. Ainsi, par exemple, un script PHP utilisant `compile_file()` et `apc_store()` peut être placé dans un espace restreint sur le serveur. Avant le démarrage du serveur, `iptables` peut être utilisé pour détourner toutes les IP sauf 127.0.0.1 (par exemple) du serveur web.

Le serveur est ensuite démarré et une commande telle que *curl* peut alors lui être envoyée vers la page spéciale qui se chargera de compiler et de mettre en cache toutes les données nécessaires. Enfin, *iptables* lève le blocage du serveur, qui est alors prêt à encaisser pleinement sa première vague de requête, avec un cache déjà plein.

II-C-5 - Filtrage

Dans certains cas, il peut y avoir des fichiers qui ne doivent pas être mis en cache. Ceci peut venir du fait de leur grosse taille, ou de leurs fréquentes modifications. Aussi, on peut vouloir ne plus mettre en cache certains fichiers qui s'avèrent buggés avec le cache (ceci peut arriver des inclusions conditionnelles par exemple).

- 1 `apc.max_file_size` : Restriction sur la taille maximum d'un fichier à cacher. Ceci peut créer des problèmes de performance lors de la demande en cache de ces fichiers-là, mais si la mémoire est limitée, l'option demeure intéressante
- 2 `apc.filters` : Expression régulière utilisée pour exclure tout fichier dont le nom a une correspondance.
- 3 `apc.cache_by_default` : Inverse la tendance de `apc.filters` : APC cachera uniquement les fichiers qui ont une correspondance sur le filtre, au lieu de les ignorer.

II-C-6 - RFC1867, progression de l'upload de fichier

PHP 5.2.0 a apporté le support de la RFC1867, la progression de l'upload de fichier. Cette option permet à APC ou à une autre extension, d'être tenu au courant d'informations concernant l'upload d'un fichier. Javascript peut alors demander au serveur des détails sur l'upload en cours, et celui-ci lui renverra ces informations.

APC supporte ce mécanisme en insérant et en gardant à jour des variables utilisateur avec des détails concernant l'upload.

- 1 `apc.rfc1867` : option booléenne d'activation/désactivation du support de la RFC1867
- 2 `apc.rfc1867_freq` : Fréquence de mise à jour en octets ou pourcentage du fichier. Les variables APC doivent être mises à jour fréquemment pour fournir des infos pertinentes, mais pas trop souvent au risque d'impacter les performances.
- 3 `apc.rfc1867_name` : Le nom du champ caché HTML utilisé pour générer le nom de la variable APC. Cette valeur doit être unique et peut être utilisée dans les requêtes suivantes pour obtenir des détails concernant l'avancement de l'upload.
- 4 `apc.rfc1867_prefix` : Préfixe utilisé pour le nom de la variable. Ceci permet d'assurer l'unicité des noms de variables et ainsi pouvoir les reconnaître.

Un formulaire compatible avec les uploads rfc1867 ressemble à ceci :

```
<form action="/upload.php" method="post" enctype="multipart/form-data" />
<input type="hidden" name="APC_UPLOAD_PROGRESS" value="A86DCF0C">
<input type="file" name="myfile" />
<input type="submit" value="upload" />
</form>
```

La variable APC contenant l'information serait alors A86DCF0C préfixée par la valeur de `apc.rfc1867_prefix` (dont la valeur par défaut est 'upload_').

III - API de APC et cache utilisateur

En plus de permettre la mise en cache de l'OPCode issu des sources des fichiers PHP, APC offre aussi un cache dit utilisateur. L'extension APC rajoute en effet des fonctions au langage PHP, qui vont permettre de stocker dans le cache utilisateur des variables PHP.

En général ces variables contiennent des informations lourdes à calculer, comme par exemple des résultats de base de données, ou d'autres traitement coûteux, qui pourront alors être chargés depuis le cache directement. Sauf pour les objets, il n'est pas nécessaire de sérialiser ces données avant de les stocker dans APC. Les ressources, elles, ne sont pas stockables en cache en raison de leur intimité et unicité vis-à-vis d'une requête HTTP.

En prenant des précautions comme avec les sessions sur plusieurs serveurs (session clustering), le cache utilisateur est très utile pour stocker tout ce qui touche tous les utilisateurs en même temps.

Nous allons passer quelques fonctions APC en revue, pour le reste, [la documentation officielle](#) est présente.

infos

- 1 array apc_cache_info ([string \$cache_type [, bool \$limited]])
- 2 array apc_sma_info ([bool \$limited])

Insertion


- 1 bool apc_store (string \$key , mixed \$var [, int \$ttl])
- 2 bool apc_add (string \$key , mixed \$var [, int \$ttl])
- 3 bool apc_compile_file (string \$filename)

Récupération

- 1 mixed apc_fetch (string \$key)

Effacement


- 1 bool apc_delete (string \$key)
- 2 bool apc_clear_cache ([string \$cache_type])


 Le **Zend Framework** propose une classe permettant de gérer le cache utilisateur dans `Zend_Cache`

Grâce à des fonctions comme **`apc_compile_file()`**, il est possible de gérer facilement la mise à jour d'un site possédant plusieurs serveurs. Plutôt que de migrer le code sous forme de révision ou de fichier, on peut demander conditionnellement aux différents serveurs de compiler la nouvelle révision. Si la fonctionnalité ajoutée doit pouvoir être annulée facilement et rapidement, c'est encore mieux.

Il faut en effet faire attention aux sites ayant plusieurs serveurs, que les caches de chacun d'entre eux soient bien mis à jour en même temps, sinon des utilisateurs risquent de se retrouver aléatoirement avec des portions anciennes du site.

IV - Conclusions

Les caches d'OPCode sont vraiment géniaux pour PHP, et doivent être installés sur tous les serveurs nécessitant des performances et une haute  **scalabilité**.

Cet article d'introduction peut être complété par le  **manuel de PHP et de l'extention APC**.

En plus de moi-même (Brian), APC est maintenu par une quantité de contributeurs, notamment George Schlossnagle, Daniel Cowgill, Rasmus Lerdorf, Gopal Vijayaraghavan, Edin Kadribasic, Ilia Alshanetsky, Marcus Börger, Sara Golemon.

Je (Brian) suis très heureux du travail qu'ils accomplissent, les patches et corrections de bugs, que ce soit pour APC, PHP ou d'autres projets.

Voici une liste (non exhaustive) des autres possibilités de cache d'OPCodes en PHP :

- 1 IonCube:  http://www.ioncube.com/sa_encoder.php
- 2 Zend Platform:  <http://www.zend.com/en/products/platform/>
- 3 XCache:  <http://xcache.lighttpd.net/>
- 4 EAccelerator:  <http://www.eaccelerator.net/>

Cet article est inspiré de la conférence " **APC @ FaceBook**".