# Bien gérer le cache serveur pour un site plus performant

*Par Aymeric Bouillat*

| | | |
|---|---|---|
| **Domaine :** | Recherche | **Référencement** |
| **Niveau :** | Pour tous | **Avancé** |

*Dans le cadre de l'optimisation de l'accélération du temps d'affichage de vos pages web, pour le plus grand bénéfice de Google mais surtout de l'internaute, il existe une large gamme de possibilités à mettre en œuvre. Nous avons parlé précédemment de la configuration des en-tête http transmises par le serveur afin de gérer au mieux le cache du navigateur distant. Analysons maintenant la mise en place de modules de cache et de reverse proxy sur le serveur, et notamment l'outil Varnish, très souvent utilisé à cette fin.*

Dans la lettre Recherche et Référencement de Juillet/Aout 2015, nous avions vu que le temps de chargement d'une page web avait un impact indirect sur le référencement. Il a un impact d'une part pour Googlebot : crawl accéléré qui permet entre autres d'avoir des pages plus fraîches dans l'index, un calcul du PR interne plus rapide et une désindexation accélérée du Duplicate Content. Mais il peut également avoir son rôle à jouer dans les pages de résultats : si les internautes sont servis rapidement, on limite le taux de rebond dans les SERP (pogosticking).

Un exemple récent nous a d'ailleurs confirmé l'existence de différents paliers de crawl par rapport à la vitesse de téléchargement des pages HTML, qui varient en fonction du volume de pages d'un site et de son importance pour un temps donné (voir fig. 1).


*Fig. 1. Corrélation entre crawl et temps de chargement des pages.*

La capacité d'un serveur à délivrer rapidement du contenu est donc primordiale. Comme vu dans la lettre Recherche et Référencement du mois d'Octobre (« Bien gérer le cache navigateur pour un site performant »), il est déjà nécessaire de mettre en cache le contenu côté client sur son navigateur via les en-têtes de cache, pour limiter le nombre de requêtes lors de l'affichage d'une page Web ainsi que pour réduire les sollicitations et la bande passante côté serveur.

La majeure partie des sites Web actuels utilisent des CMS ou frameworks reliés à une base de données ou des webservices, qui permettent un affichage dynamique du contenu *via* un système de templates pour chaque URL appelée. Que ce soit Drupal, WordPress ou encore Magento, ces applications peuvent devenir très gourmandes en fonction des plugins ou modules utilisés, mais également des fonctionnalités proposées par certains thèmes. Un cache côté serveur devient donc indispensable avec ces systèmes de gestion de contenus.

## *Principe du cache serveur*

Si on revient quelques années en arrière, une partie des sites du Web était statique, à savoir composée d'éléments restitués tel quel au navigateur de l'internaute à partir du serveur. Parmi ces ressources statiques, on trouvait des feuilles de style (css), des fichiers Javascript (js), des images (jpg, gif...) ainsi que des pages HTML intégrés à l'avance, chaque page étant disponible sur le serveur dans une version finale pour l'internaute.

A l'opposé de ces ressources statiques (HTML « en dur ») sont apparues des ressources dynamiques, qui se sont maintenant généralisées. Ces ressources sont générées par une application, à partir de différents langages (Asp.net, PHP, Perl/CGI), pouvant faire appel à des bases de données ou divers services web (API).

Là où les pages dynamiques générées peuvent être délivrées relativement rapidement aux internautes pour un site à faible trafic, cela peut s'avérer beaucoup plus compliqué en cas de montée en charge… A l'appel d'une URL, une page devra être reconstituée par le serveur *via* le langage utilisé par le CMS et ses composants en relation avec la base de données, pour être délivré à l'utilisateur final, cette opération pouvant être répétée pour chacun d'entre eux.

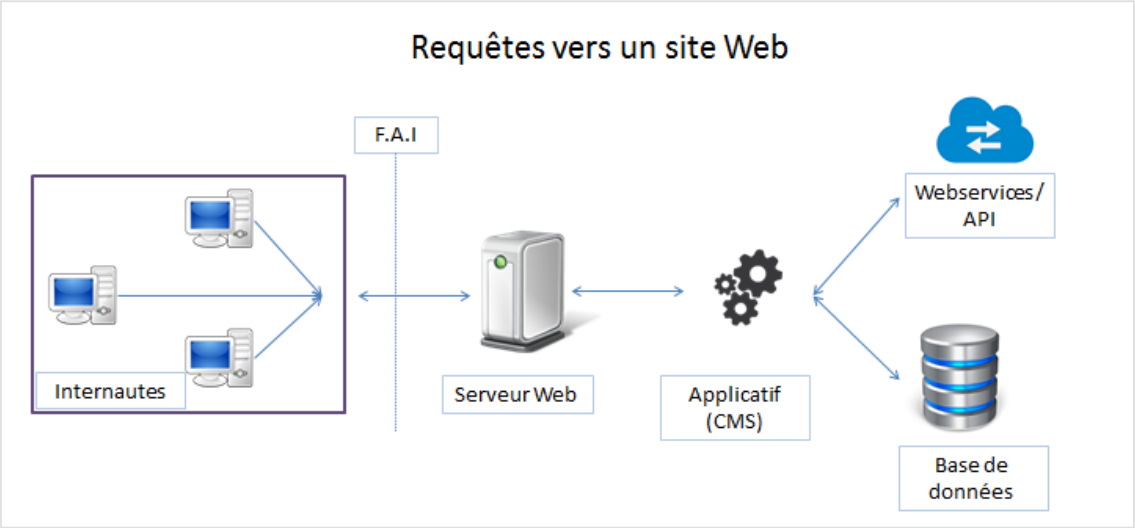



*Fig. 2. Transit des informations entre un internaute et un site web.*

Ces calculs simultanés, couplés à des centaines de requêtes vers la base de données pour chaque utilisateur, ainsi que l'appel de nombreux fichiers de script pour générer la page finale (bien qu'il existe une possibilité de mettre en cache un certain nombre de ces opérations : APC, cache d'Opcodes, cache de requêtes SQL), peuvent amener le serveur d'un site Web à saturation, où l'ensemble des processus qui permettent de mettre en place le contenu HTML final provoqueront un goulet d'étranglement.

## *Qu'est-ce qu'un reverse proxy ?*

Le reverse proxy se positionne entre les internautes et le serveur Web. Pour ne pas confondre le proxy et le reverse proxy, voici un petit rappel :

**Proxy :** le proxy se positionne côté client. Il permet la mise en cache d'éléments pour l'ensemble des salariés d'une entreprise par exemple, ainsi que le filtrage de contenu. C'est un intermédiaire entre des internautes spécifiques et l'ensemble du Web, leur cache commun.

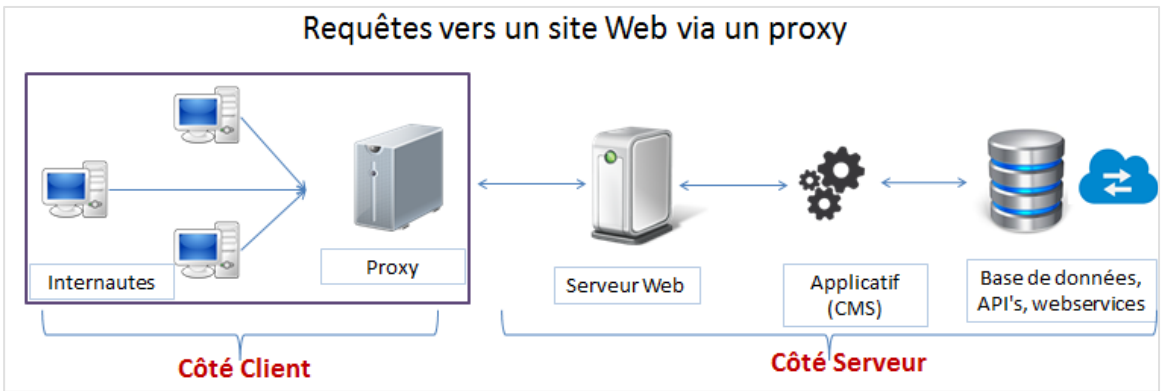



*Fig. 3. Transit des informations entre des internautes et un site web, via un proxy.*

ReverseProxy : le ReverseProxy se positionne côté serveur. Il permet la mise en cache d'éléments d'un site web spécifique pour tous les internautes. C'est un intermédiaire entre un site spécifique et l'ensemble des internautes.


*Fig. 4. Transit des informations entre des internautes et un site web accompagné d'un reverse proxy.*

Vous l'aurez compris, le reverse proxy aura alors un rôle clé dans le processus décrit précédemment avec un site web dynamique, puisqu'il conservera une copie HTML de chaque page pour un temps donné, en évitant de solliciter le serveur web et la couche applicative pour chaque internaute : il distribuera la même copie de chaque page HTML (précédemment générée dynamiquement) qu'il aura conservée dans son cache, à tous les internautes.

En cas de forte sollicitation lors de marronniers par exemple (soldes, Noël, fête des mères, etc.), le ReverseProxy sera capable de servir un grand nombre de pages HTML grâce aux éléments déjà dans son cache, et soulagera donc le serveur en évitant ainsi une dégradation des temps de réponses habituels.

Il est également possible d'utiliser des systèmes de cache sur la partie applicative, directement au niveau des CMS. Parmi les plus célèbres, on citera le module « Boost » pour Drupal ou encore « WP Rocket » pour WordPress. Ces solutions sont tout à fait adaptées et peuvent suffire dans certains cas. Cependant, elles passent par la partie applicative du site web, et font appel à des technologies intermédiaires pour gérer le cache (PHP, parfois SQL, règles de réécriture Apache) ce qui rajoute encore des ressources à attribuer dans le traitement du cache, au lieu d'alléger la charge serveur.

L'intérêt du reverse proxy est qu'il se situe juste devant le serveur. C'est son unique fonction : délivrer des fichiers mis en cache précédemment pour répondre à des milliers de requêtes en quelques secondes.

## *Améliorer le crawl de Googlebot*

Pour qu'une page puisse être mise en cache par le Reverse proxy (ou un système de cache applicatif), il faut qu'elle soit appelée *via* une requête HTTP. Le premier appel peut donc provoquer une requête plus longue : puisque le contenu n'existe pas encore dans le cache, le

reverse proxy lancera un appel au serveur web pour générer le rendu HTML d'une page dynamique, afin d'en garder une copie dans son cache pour les internautes suivants. Ce délai plus élevé pour la première requête sera donc raccourci par la suite.

GoogleBot crawle de façon régulière les pages d'un site, avec une fréquence plus élevée pour les URL les mieux maillées. Mais en crawlant des URL profondes correspondant à des filtres ou à des pages issues de la navigation à facettes, il appelle des pages qui sont plus rarement demandées par les internautes, pour lesquelles aucune version en cache n'existe, bien souvent.

Si certaines de ces URL trop « longue traîne » n'ont pas été restreintes *via* le fichier robots.txt, les temps de chargement des pages seront donc plus élevés pour Googlebot, puisque les pages HTML appelées sur le serveur web devront être générées dynamiquement *via* de nombreuses requêtes. Cette hausse du temps de téléchargement de ces URL profondes limitera donc le crawl de pages pertinentes, GoogleBot ne pouvant allouer qu'un temps machine spécifique pour chaque site web qu'il crawle.

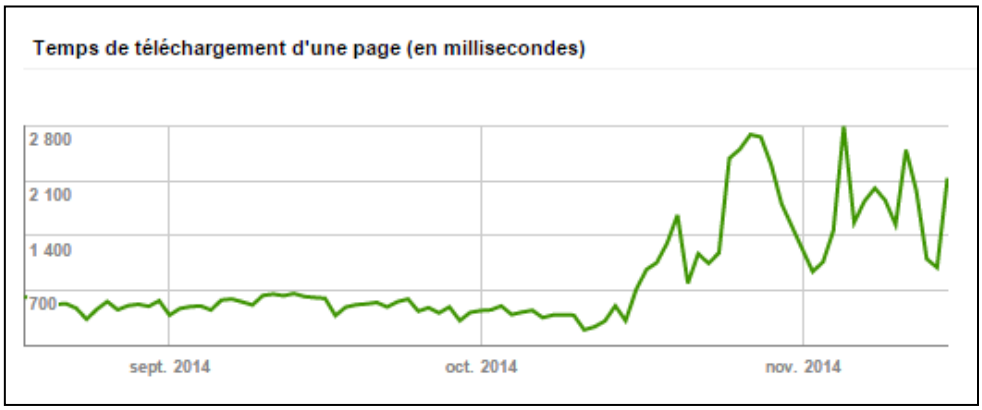



*Fig. 5. Hausse du temps de téléchargement dans Google Search Console suite à l'ouverture au crawl de tout le moteur à facettes d'un site.*

Pour améliorer le crawl de GoogleBot avec un reverse proxy, il sera possible d'augmenter la durée de vie du cache pour certains éléments (en fonction des URL, du type de page), ou d'appeler ces URL de façon régulière en période nocturne (serveur avec une faible charge) pour générer à l'avance les versions cache des contenus pouvant être gourmands.

## *Varnish devant Apache pour booster votre site*

**Varnish** est un Reverse proxy, également capable de faire du *Load Balancing* (répartition de trafic entre différents serveurs pour améliorer les performances d'un site). A l'instar des serveurs web Nginx ou Apache qui proposent en option des fonctionnalités de reverse proxy, Varnish n'est qu'un reverse proxy.

Il s'agit d'un logiciel open-source, développé sous licence BSD. A l'origine conçu pour un tabloïd norvégien, il a commencé à se généraliser rapidement, pour être placé devant 545 000

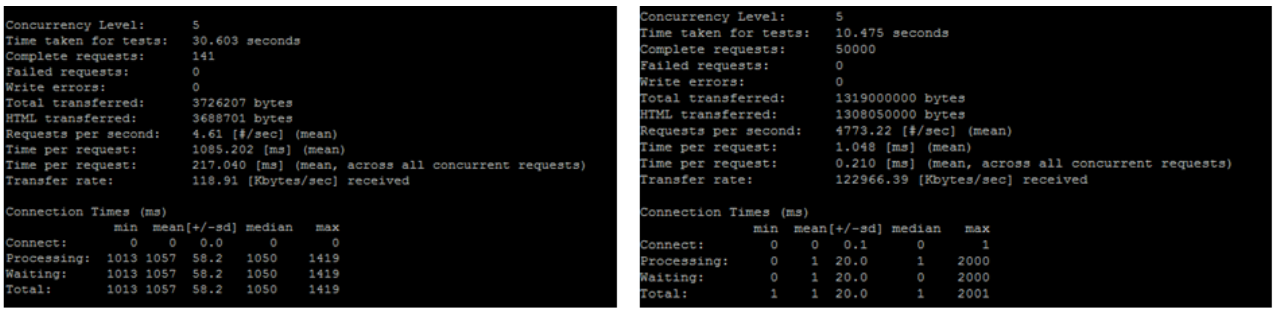
noms d'hôtes en 2010. Aujourd'hui, de très gros sites l'utilisent comme The Guardian, Tumblr, le New York Times ou encore Wikipedia.

Il utilise un langage spécifique : VCL (*Varnish Configuration Langage*), qui est compilé en langage C à chaque démarrage du service, ce qui lui procure de très bonnes performances. Nous avons effectué un test sur un petit serveur, avec un WordPress fonctionnant sans aucun plugin de cache, et le résultat est sans appel. Pour 5 threads lançant des requêtes pendant 30 secondes sur le site testé :

• Sans Varnish : 4, 61 requêtes par seconde, 141 requêtes complétées, temps moyen par requête : 217ms

• Avec Varnish : 4 773 requêtes par seconde, 50 000 requêtes complétées, temps moyen par requête : 0,21ms

Il est évident qu'avec de tels résultats, Varnish est indispensable pour faire face à toute montée en charge.



```
Concurrency Level:      5
Time taken for tests:   30.603 seconds
Complete requests:      141
Failed requests:        0
Write errors:           0
Total transferred:      3726207 bytes
HTML transferred:       3688701 bytes
Requests per second:    4.61 [#/sec] (mean)
Time per request:       1085.202 [ms] (mean)
Time per request:       217.040 [ms] (mean, across all concurrent requests)
Transfer rate:          118.91 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:  1013 1057  58.2   1050    1419
Waiting:     1013 1057  58.2   1050    1419
Total:       1013 1057  58.2   1050    1419
```

```
Concurrency Level:      5
Time taken for tests:   10.475 seconds
Complete requests:      50000
Failed requests:        0
Write errors:           0
Total transferred:      1319000000 bytes
HTML transferred:       1308050000 bytes
Requests per second:    4773.22 [#/sec] (mean)
Time per request:       1.048 [ms] (mean)
Time per request:       0.210 [ms] (mean, across all concurrent requests)
Transfer rate:          122966.39 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       1
Processing:     0    1  20.0      1    2000
Waiting:        0    1  20.0      0    2000
Total:          1    1  20.0      1    2001
```

*Fig. 6. Test de montée en charge sans Varnish (à gauche) et avec Varnish (à droite)*

Afin de bien comprendre Varnish, voici le principe de fonctionnement des briques qui constituent ce reverse proxy. Il est composé de plusieurs routines pour lesquelles des traitements particuliers peuvent être appliquées à chaque requête :

• vcl_recv = requête entrante
• vcl_hash = création d'une clé de cache
• vcl_fetch = on récupère la requête sur le backend/serveur web
• vcl_deliver = on retourne la réponse au client
• vcl_error = traitement des erreurs

## Principe des VCL et routines Varnish

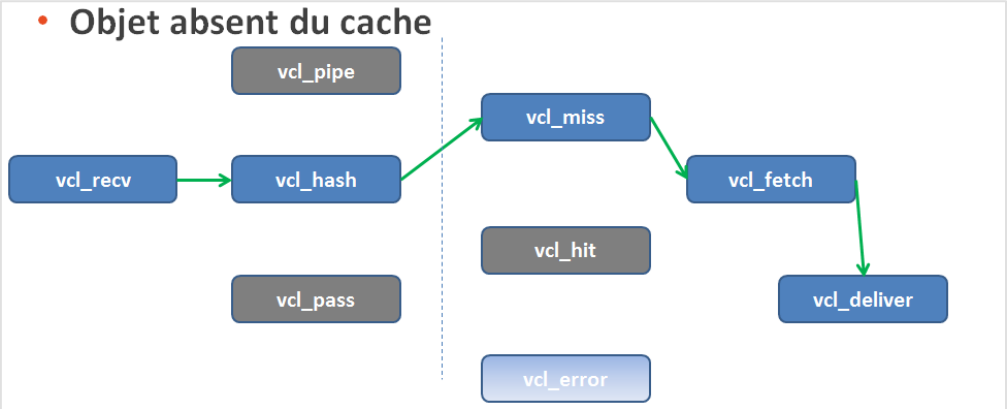



*Fig. 7. Objet absent du cache.*

Pour une requête entrante (vcl_recv), Varnish génère une clé/un identifiant unique (vcl_hash). Il vérifie ensuite la présence de la page en cache en fonction de cette empreinte. Si l'URL demandée n'est pas en cache (vcl_miss), il récupère la ressource sur le serveur Web (vcl_fetch), et délivre ensuite la page qu'il vient de stocker en cache à l'internaute (vcl_deliver).

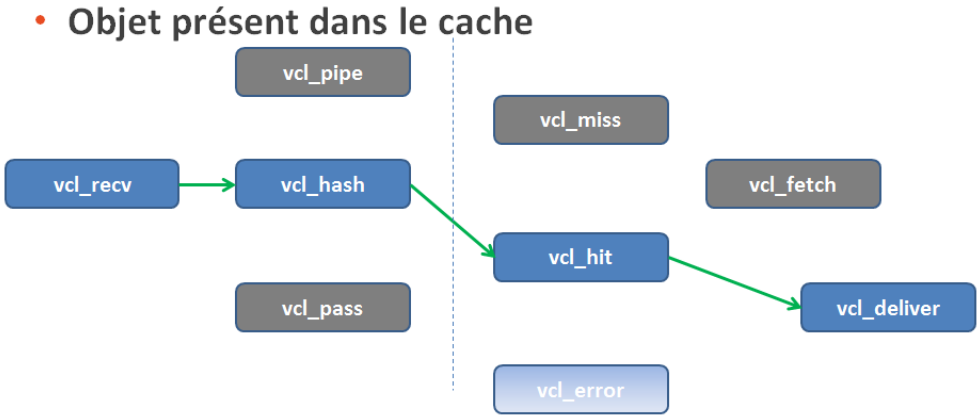



*Fig. 8. Objet présent dans le cache.*

Pour une requête entrante (vcl_recv), Varnish génère une clé/un identifiant unique (vcl_hash). Il vérifie ensuite la présence de la page en cache en fonction de cette empreinte. Si l'URL demandée est en cache (vcl_hit), il délivre directement la version en cache de la ressource à l'internaute (vcl_deliver).

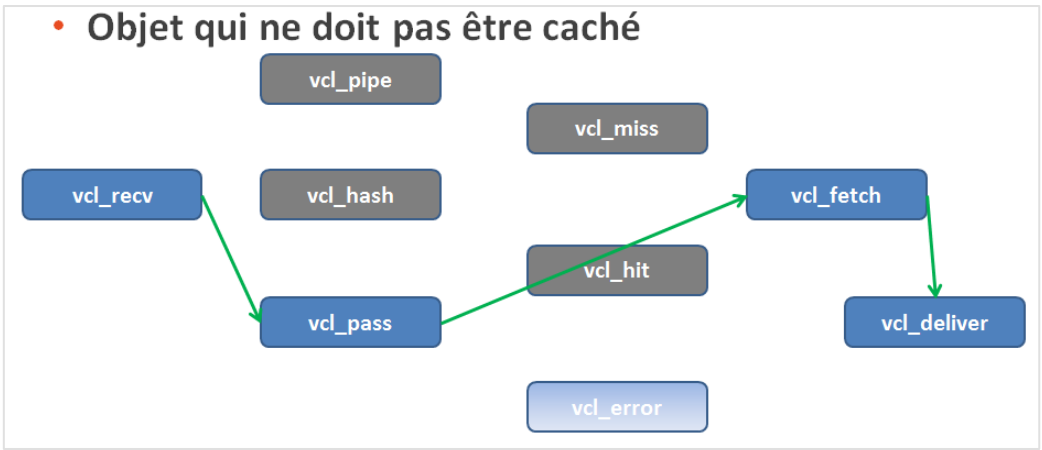


*Fig. 9. Objet qui ne doit pas être caché.*

Pour une URL qui ne doit pas être mise en cache (ex : page Mon panier/Mon compte), Varnish reçoit la requête entrante (vcl_recv), analyse le fait qu'elle ne doit pas être cachée *via* une configuration réalisée au préalable, et envoie directement la requête au serveur Web (vcl_pass) pour qu'elle soit executée (vcl_fetch) puis renvoyée à l'internaute (vcl_deliver).

Différentes manipulations peuvent avoir lieu dans chacune des routines grâce au langage de configuration VCL. Pour l'exemple ci-dessus, on va détecter la présence de cookie ou d'une authentification :

```
sub vcl_recv {
   if (req.http.Authorization || req.http.Cookie) {
      /* Cookie au auth, pas de cache*/
      return (pass);
   }
}
```

Si la requête entrante est authentifiée ou contient un cookie, on ne cache pas le contenu et on va le chercher directement sur le serveur.
On pourra également paramétrer des délais spécifiques pour la mise en cache des éléments statiques, avoir un cache spécifique pour Googlebot avec des TTL plus longs, où ne pas cacher les images pour éviter de surcharger le cache inutilement, d'autant plus quand c'est la RAM qui est utilisée comme support de cache. On notera que de nombreuses manipulations des en-têtes http sont possibles via Varnish.

Par défaut, Varnish ne met en cache aucune URL contenant des cookies dans ses en-têtes http, les cookies étant souvent personnels. Pour une page qui se composerait d'éléments personnalisés qui ne sont pas communs à tous les internautes, Varnish utilise un système d'includes (ESI = *Edge Side Includes*) qui permettent de mettre en cache uniquement certaines portions d'une page HTML. La figure 10 en un exemple.


Fig. 10. Structure de site utilisant des includes ESI

Sur la page web de la fig. 10, deux blocs seront traités de façon spécifique :

• « Mon panier » : pas de mise en cache, contenu spécifique à chaque utilisateur.

• 5 derniers articles : durée du cache + courte que les autres ressources, de nouveaux articles pouvant arriver toutes les 30 minutes (TTL : *Time To Live*).

Le serveur Web renvoie une page HTML avec des includes ESI, et c'est Varnish qui se charge de recomposer l'intégralité de la page, en appelant le contenu des différents blocs spécifiques.

Pour éviter de « faire tomber » le backend dans le cas où un nombre élevé d'internautes appelleraient une même ressource au même moment, Varnish n'envoie qu'une seule requête au serveur Web, et place toutes les autres utilisateurs en attente. Dès que la réponse du backend revient à Varnish, il la délivre à toute la file d'attente. Cela évite d'envoyer un nombre élevé de requêtes identiques au backend de façon simultanée, ce qui pourrait faire tomber le serveur assez rapidement (surcharge).

## *Gain de performance avec quelques contraintes*

L'ajout d'un reverse proxy sur l'architecture d'un site Web n'est pas anodin. Cela rajoute une couche supplémentaire à l'architecture, et donc des coûts d'exploitation et de maintenance. Les résultats obtenus seront bien évidemment à la hauteur de vos attentes comme vous avez pu le constater dans notre test, mais tout doit être analysé en amont, pour s'assurer d'une bonne compatibilité de votre applicatif web avec un serveur de cache de ce type. En effet, la mise en place d'un reverse proxy n'est pas adaptée dans certains cas, il ne faut pas oublier que les contenus mis en cache seront communs à tous les utilisateurs.

Un reverse proxy pour des sites proposant du contenu personnalisé en fonction des internautes connectés sera plus complexe à mettre en place, et ne sera pas une solution adaptée, excepté si seuls quelques éléments de la page sont spécifiques à chaque utilisateur : un système d'includes (type ESI) permettra de répondre à ce besoin. Attention malgré tout, cela complexifiera la configuration de l'architecture et demandera des ressources supplémentaires : tout doit être jaugé dès le début du projet pour éviter que la mise en place d'une telle solution ne produise l'effet inverse de celui escompté.

Sources :
https://en.wikipedia.org/wiki/Varnish_(software)
http://info.varnish-software.com/blog/november-545-000-new-hostnames-have-varnish

**Aymeric Bouillat,** *consultant SEO, Resoneo (http://twitter.com/aymerictwit ou http://www.yapasdequoi.com)*