Vitesse de chargement d'un site web et SEO (2ème partie)

Retour au sommaire de la lettre

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

Tout internaute sait qu'un site web qui s'affiche lentement est désagrable à consulter et apporte une mauvaise expérience utilisateur. Mais Google indique également depuis plusieurs années que ce critère est important pour lui. Il semblait donc capital de bien comprendre en quoi la rapidité d'affichage d'un site est un point à prendre en compte, quels sont les critères pris en considération par Google pour le mesurer et, bien sûr, les différentes options disponibles pour y remédier. Après avoir défini un certain nombre de termes et analysé les différents moyens de mesurer ce critère, nous listons ce mois-ci les différents moyens à notre disposition pour optimiser un site. Etes-vous prêt à appuyer sur l'accélérateur ?

Nous sommes en 2014. Vous avez donc certainement déjà entendu parler de l'importance de l'amélioration la vitesse de chargement de vos pages web. Certains webmasters ont déjà agi à ce niveau, tandis que pour d'autres, les bonnes pratiques sont plus difficiles à être appliquées, avec souvent comme justification une mise en place réputée complexe. Pourtant, il n'en est rien, ou plus exactement, on peut se concentrer sur une sous-partie des bonnes pratiques uniquement, celles qui sont à la fois faciles à appliquer et qui amènent des résultats visibles. Une fois que l'on aura gouté à la vitesse et vu les résultats arriver, on voudra aller plus loin, et déployer les choses les moins évidentes.

Le mois dernier, nous avons défini un certain nombre de concepts et vu comment mesurer ce temps de chargement. Ce mois-ci et dans cette seconde partie, nous entrons dans le vif du sujet en vous donnant des pistes concrètes d'optimisation de votre site.

Optimiser les images

Sur de trop nombreux sites, les images ne sont pas ou peu optimisées. On rencontre souvent :
- des images qui sont affichées bien plus petites que leur taille sur le serveur (pour afficher un tout petit visuel sur mobile, inutile de faire télécharger une image de 10 Mo) ;
- des images qui ne sont pas dans leur format optimal (des aplats de couleur en jpg par exemple, alors qu'un gif serait plus adapté) ;
- des images qui intègrent une palette de couleurs trop riches, et sont donc plus lourdes, sans raison (un PNG24 qui aurait le même aspect en PNG8) ;

Dans de nombreux cas, la faute en incombe aux CMS utilisés par les sites, qui permettent d'*uploader* des images facilement… sans prendre le soin de les convertir dans un format optimal.

Optimiser les images est une chose à faire absolument car le régime est (trop) souvent très efficace. Quand vous n'aurez plus que 80ko de ressource à télécharger au lieu de 3 Mo, vous verrez la différence immédiatement sur le temps de chargement (rien n'empêche d'avoir la ressource en haute résolution, disponible pour un internaute qui cliquerait sur l'image, si vous êtes perfectionniste).

Bien sûr, pour optimiser des images, il faut connaître quelques notions techniques à ce niveau, mais sur le Web il n'y a globalement que 3 formats utilisés : le GIF, le PNG et le JPEG. Cela limite donc l'acquisition de connaissances.

Certes, Google essaie de diffuser le WebP, un format d'image qui serait 30 à 80% plus léger que le JPEG et le PNG, mais il n'est pas encore franchement adopté, donc on peut

l'oublier, et les formats vectoriels sont utilisés dans des cas si particuliers que si vous n'avez pas eu l'idée de les utiliser c'est que vous n'en avez certainement pas besoin.

Pour aider à compresser les images, il existe de nombreux outils (smush.it [http://www.smushit.com/ysmush.it/], Kraken.io [https://kraken.io/], PunyPng [http://www.punypng.com/], Photoshop [http://www.adobe.com/fr/products/photoshop.html], ScriptCX [http://css-ig.net/scriptcx], jpegmini [http://www.jpegmini.com/]). Certains se lancent même en ligne de commande Linux (jpegtran [http://www.ijg.org/], imagemagick [http://www.imagemagick.org/]), ou proposent des API. Plus d'excuse alors pour que votre CMS n'optimise pas un minimum les images.

Une fois toutes les images optimisées et à la bonne taille, il faut penser à d'autres aspects :
- Supprimer les métadonnées pour alléger l'image (même si on peut se demander si ce n'est pas utile pour le référencement web) ;
- Indiquer les dimensions de l'image dans la balise <img> (le navigateur réserve alors l'espace, ce qui lui évite, dans certains cas, de devoir redessiner la page après chargement, ce qui prend toujours un peu de temps) ;
- Servir les images à partir d'un nom de domaine sans cookie (si un autre nom de domaine que celui du site est utilisé). Cela évite d'intégrer le cookie en surpoids de chaque chargement d'image.
- Utiliser un script de lazyloading [http://lamidudeveloppeur.fr/javascript/lazyload-lart-du-chargement-paresseux-jquery/] pour éviter de charger les images qui ne sont pas encore à l'écran, et qui seront alors chargées seulement lors du scroll. On libère de la bande passante en ne lançant pas de téléchargements inutiles.

Pour aller plus loin, on peut avoir envie de se renseigner sur les images adaptives [http://blog.nursit.net/Adaptive-Images-et-Responsive-Web.html], une technique qui consiste à servir des images différentes en taille selon la résolution de son appareil de consultation. De cette façon, le lecteur reçoit toujours une image optimisée, quelles que soient les conditions.

Limiter le nombre d'appels et les résolutions DNS

Dans la plupart des cas, une page web est constituée d'un code source qui encapsule d'autres ressources, disponibles à des adresses différentes (les images, les fichiers CSS, JavaScript, les vidéos, etc.).

Le premier souci est que chaque requête demande des allers/retours avec le serveur, donc du temps. L'autre problème est que les navigateurs ont une limite haute du nombre de ressources qu'ils vont télécharger à la fois. Pendant longtemps, on ne pouvait récupérer que quatre ressources à la fois. C'est-à-dire que si une page contenait 10 images, le navigateur téléchargeait en même temps les 4 premières, puis les 4 suivantes, puis les deux dernières, quelle que soit la bande passante disponible. Peut-être auriez-vous pu sans mal télécharger les 10 fichiers d'un coup… mais arbitrairement cela n'était pas possible.

Cette limite est fixée par nom de domaine, et plus précisément par sous-domaine. On a donc la possibilité de placer les images sur des sous-domaines différents pour paralléliser leur téléchargement.
Si on reprend l'exemple précédent, on peut répartir :
images 1 à 4 sur sd1.exemple.org ;
images 5 à 8 sur sd2.exemple.org ;
images 9 et 10 sur sd3.exemple.org ;

Les 10 images se chargent donc en même temps : si la connexion dont on dispose le permet, tout apparaît plus vite.

Le revers de cette astuce est que pour accéder à un nouveau sous-domaine, le navigateur

doit lancer une résolution DNS (il doit obtenir l'adresse IP correspond au serveur où est hébergé la ressource). Cette résolution prend du temps, et malheureusement on s'est rendu compte qu'en général, au dessus de 2 (sous-)domaines pour répartir les images, on y perdait en vitesse de chargement.

On va donc plutôt se contenter de placer les images sur un autre domaine, sans cookie, voire deux, mais rarement plus (sur un site web en https, on installera SPDY [http://www.chromium.org/spdy/spdy-whitepaper] et on ne fera pas de répartitions sur plusieurs domaines).

Le temps de chargement supplémentaire induit par les résolutions DNS introduit un autre problème : quand on fait appel à de nombreux éléments extérieurs au site web dans une page, on augmente le temps de réponse. Déjà, parce que les fournisseurs tiers n'ont parfois pas mis en place d'optimisation de la vitesse de leur côté (vous subissez directement leur manque de vélocité), mais aussi, tout simplement parce qu'il faut attendre la résolution DNS.

Ces deux facteurs combinés expliquent pourquoi on peut être réticent à introduire les widgets des réseaux sociaux tels que Facebook, Google+ et Twitter. Il faut alors réussir à rendre cette intégration non bloquante (exécution asynchrone), ou faire autrement. Et pour cela, on peut placer des liens ne faisant pas d'appel extérieur :

Pour Facebook : http://www.facebook.com/sharer.php?u=URL
Pour Google+ : https://plus.google.com/share?url=URL
Pour Twitter : http://twitter.com/share?url=URL

Mais on perd en même temps en attrait… à vous de voir (on peut également bricoler une solution où l'on utilise le widget mais on place en cache le résultat, qu'on réutilisera ensuite sans réinterroger le réseau social, pendant plusieurs heures).

C'est aussi pour éviter les appels superflus que l'on va toujours essayer de faire en sorte de n'avoir qu'un seul fichier CSS et un seul fichier JS. Dès qu'on en a plus de deux, on a des appels superflus.

Dans le même esprit, pour les petits images que l'on retrouve un peu partout sur le site (barre de navigation, petits pictogrammes, etc.), on ne devrait jamais avoir à faire un appel par élément. On utilise plutôt la technique des sprites CSS [http://www.alsacreations.com/tuto/lire/1068-sprites-css-background-position.html], qui permet de placer toutes les images dans une seule et de faire apparaître seulement des portions bien choisies de l'image, là où il le faut, *via* le CSS. Le chargement se fait ainsi de façon plus fluide.

Encore plus surprenant peut-être, on peut être amené, pour afficher une toute petite image qui ne serait pas trop (voire pas du tout) répétée sur un site, d'intégrer directement le code de l'image dans le fichier html.

```
<img alt="alt"
src="
ALAAAAAAYABgAAAJjlIepi2HHGpggmmANjUjws32RI1AVhzGPOWATp6wT+MKdRwl
BaJPmXrMNZBWekGg6HXHJIMzHMn40OdayNMM6p4OcdruAZpM24lA7ij0wD4jKIyzr
4qO2O06y3+z8/sUP+FAAADs=">
```

L'attribut src est fabriqué de la façon suivante :
- "data :image/"
- suivi du type d'image ("gif" par exemple)
- suivi de " ;base64,"
- suivi du code source de l'image en base 64.

on peut faire appel à plusieurs techniques pour récupérer le code en base 64 :
- En PHP :

```
$image = file_get_contents('image.gif');
```

```
    echo base64_encode($image);
```
- En ligne de commande sous linux :
```
    cat image.gif | base64 –w 0
```
- En utilisant un outil de conversion en ligne
    par exemple : http://webcodertools.com/imagetobase64converter

Dans certains cas, il est plus rapide d'avoir le code source de l'image directement dans le fichier html que de devoir lancer une requête extérieure. Mais ce n'est pas toujours vrai, il faut vérifier à chaque fois. En tout cas, on est, pour cette pratique, bien loin de l'action systématique, importante. C'est bien de savoir que cela existe, mais de là à la généraliser, c'est une autre histoire.

Bien placer les appels Javascript et CSS

S'il y a bien quelque chose qui ralentit une page web, c'est d'avoir des scripts Javascript à exécuter. Même si les moteurs d'exécution sont devenus plus performants au fil des ans, ce langage reste encore un peu lent à interpréter dans un navigateur.

Pourtant, quand on utilise du Javascript sur une page web, c'est souvent pour provoquer une action qui aura lieu après le chargement. Alors pourquoi charger le fichier JS avant le contenu ? Mettons-les systématiquement en bas de page, et quand c'est possible exécutons les de manière asynchrone. Le reste de la page s'affichera sans attendre que le JS se charge.

Pour le CSS, c'est l'inverse : dès qu'on commence à afficher la page dans le navigateur, on devrait avoir déjà récupéré l'ensemble des propriétés CSS. Donc on va placer l'appel à la feuille de style en début de fichier, dans le <head>.

Dans tous les cas, quand vous modifiez l'emplacement des appels CSS et JS, testez, car on n'est pas à l'abri d'inverser l'ordre d'appel de deux scripts qui ne fonctionneront alors plus du tout.

Mais en plaçant correctement les appels CSS et JS, on peut parfois gagner plusieurs secondes de temps de chargement.

On profitera de cet intérêt pour nos fichiers JS et CSS pour les optimiser eux aussi, c'est-à-dire réduire leur poids en les « minifiant ».

Il s'agit de les « ranger » de façon à ce qu'ils soient le plus léger possible. En effet, il n'est pas rare que dans un fichier CSS des propriétés se répètent, et sont donc susceptibles d'être regroupées, et donc de prendre moins de place.

Pour les fichiers JS, la « minification » est moins évidente et on s'appuiera volontiers sur un script capable de le faire pour nous (en vérifiant toujours que le résultat soit fonctionnel).

Pour le CSS, on peut utiliser Minify [https://code.google.com/p/minify/], pour le JS ce sera Closure Compiler [https://developers.google.com/closure/compiler/?hl=fr].

N'est-ce pas un peu compliqué tout cela ?

Nous avons passé beaucoup de points d'optimisation importants en revue. Pourtant il en reste de nombreux encore sous silence. Cependant, si cela vous semble déjà très compliqué, sachez que des sociétés ont pensé à vous.

Google, bien sûr, met à disposition deux plugins pour faire appliquer automatiquement à votre serveur web certaines bonnes pratiques :

mod_pagespeed pour Apache ;

ngx_pagespeed pour nginx.

Les deux sont téléchargeable chez Google [https://developers.google.com/speed/pagespeed/module/using?hl=fr].

Ces plugins donnent des résultats en contrepartie d'un effort proche de zéro, c'est certain, mais, encore une fois, cela ne remplace pas une optimisation faite à la main. Quand on veut activer toutes les optimisations sur un des modules cités, on a parfois de curieux problèmes de charge serveur. Oui, automatiser demande de la ressource, et parfois cela réduira d'autant plus la vitesse de chargement. Un comble !

Si vous êtes adepte de Wordpress, vous pourrez avoir envie de tester le plugin de cache WP Rocket [http://wp-rocket.me] qui bénéficie de bons retours clients, et si vous cherchez à la fois une optimisation et un CDN, c'est du côté de CloudFlare [http://www.cloudflare.com/] que votre attention devrait se porter (la version payante étant à privilégier pour avoir un service plus qualitatif), même si leur discours sur le TTFB est assez étonnant [http://blog.cloudflare.com/ttfb-time-to-first-byte-considered-meaningles].

Optimiser le back end

Pour le moment, nous avons évoqué presque uniquement des bonnes pratiques qui permettent d'améliorer la vitesse de chargement dans le navigateur. Mais avant d'arriver au navigateur, les pages doivent être générées par le serveur web.

Or, parfois, le serveur est sous-dimensionné, mal configuré, pas aussi optimisé qu'il pourrait l'être, et cela se ressent au niveau de la vitesse.

Si du côté front end on peut s'exaspérer en voyant un site lent afficher petit à petit des images, quand c'est le back end qui pèche, c'est encore pire : on ne voit rien tant que la page n'arrive pas. Ne pas apercevoir un seul morceau de page pendant plusieurs secondes, c'est la recette assurée pour perdre des visiteurs. Le pire, c'est que si vous monitorez votre trafic uniquement via des solutions en Javascript, vous n'en serez certainement jamais conscient.

Décrire les meilleures façons d'optimiser le back end d'un site ou d'un serveur web nécessiterait des livres entiers, spécialisés dans chaque technologie impliquées, mais ce n'est pas l'objet de cet article. Nous souhaitons plutôt pointer, encore une fois, les bonnes pratiques, les bons réflexes à acquérir, et les aborder suffisamment pour donner des pistes et l'envie de les appliquer.

Améliorer son Time To First Byte (TTFB)

Le TTFB dépend de trois facteurs :
- le temps de latence entre le client et le serveur ;
- la charge serveur ;
- la vitesse de génération du back end.

On peut essayer d'améliorer le temps de latence en utilisant un CDN, de façon à « rapprocher » le client du serveur. On peut éventuellement l'améliorer en ayant un réseau plus robuste autour du serveur (un serveur dans un datacenter professionnel, c'est autre chose que d'avoir son serveur connecté au bout d'une ligne ADSL dans le bureau d'à côté).

La charge serveur, elle, est facile à mesurer, et en cas de besoin on peut tout simplement changer de serveur pour aller vers plus puissant, ou bien encore avoir une architecture web en grappe, qui permettra de faire du *load balancing*, ou de répartir les calculs / les visiteurs / les bases de données, sur plusieurs machines.

Au niveau d'un seul serveur, cela plaide pour une utilisation réfléchie. Par exemple, on n'utilise pas un script php pour sauvegarder une base de donnée. On fait un dump en ligne de commande pour éviter une saturation de MySQL et une indisponibilité temporaire du serveur due à la charge de la machine.

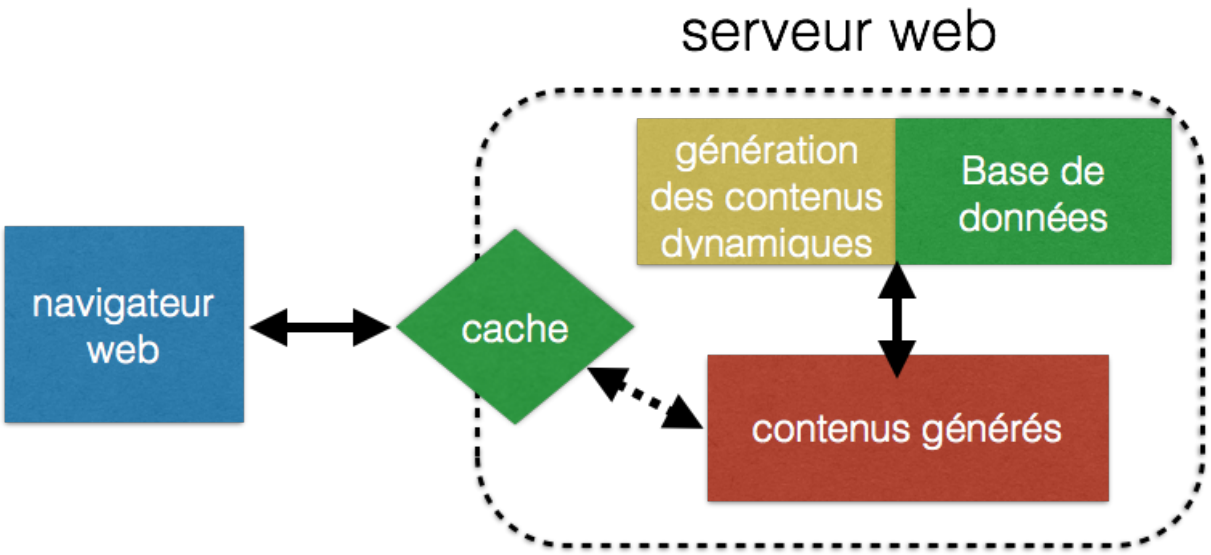
On peut en revanche vraiment agir sur la vitesse de génération du back end, qui se joue quelque part entre la qualité du code et le dimensionnement de la machine. Au final, on doit parvenir à gagner quelques millisecondes, voire secondes si on part de loin, en optimisant le TTFB.

Intuitivement, on peut penser que, pour réduire la charge serveur, il faut arrêter de trop l'utiliser, pour ne plus avoir de problème de vitesse de génération de back end, il faut arrêter d'en faire par trop usage. C'est l'un des secrets de l'optimisation , ou plutôt de deux couches d'optimisation. L'optimisation dite réelle : vous avez vraiment travaillé pour améliorer votre back end, et l'optimisation « virtuelle », qui consiste à « tromper » le visiteur et les robots d'indexation des moteurs de recherche. Vos pages mettent 10 secondes à se générer ? Ce n'est pas grave, vous les générez à l'interne, et vous proposez à la consultation une version en cache, déjà pré-générée. De cette façon on gagne des points sans pour autant revoir son back end de fond en comble.

Utiliser un cache

Vous l'avez bien saisi, le cache, c'est la solution à beaucoup de problèmes de vitesse. Généralement le serveur web est capable de travailler extrêmement vite avec des **contenus statiques**. Si vous avez seulement des pages html « en dur », des images, des fichiers CSS, JS, etc. mais aucune génération, tout va très vite et on peut absorber énormément de trafic sans trop forcer sur la puissance de la machine.

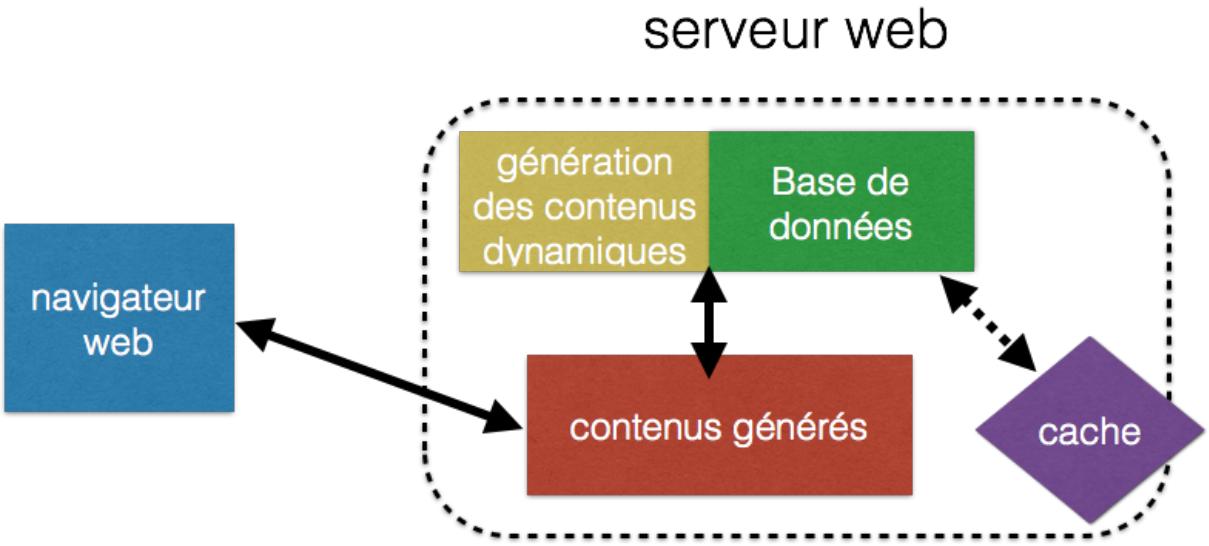
Quand on a un **site dynamique**, c'est une autre histoire. Si générer une page demande 128 Mo de mémoire vive et 25% de votre processeur, comment va se comporter votre serveur web quand 250 personnes vont demander la même page en même temps ? Donc, on va essayer de toujours avoir un cache, c'est-à-dire une copie du site web en version statique. Ce sera un cliché pris à un instant T, qui ne sera peut-être pas tout à fait à jour (il faut prévoir des systèmes de suppression sélective du cache pour éviter cela), mais qui sera rapide à délivrer.



Pour un e-magazine, par exemple, on peut générer le cache au lancement du serveur, puis à chaque fois qu'une page est modifiée (mise à jour d'un article, ajout d'un commentaire en bas de page, etc.), on la régénère, en interne.

Pour une boutique en ligne, même principe, si ce n'est qu'on peut être éventuellement inquiet quant aux quantités d'articles disponibles qui s'affichent ou au prix changeants : une grande vigilance est requise lors du déploiement du système de rafraîchissement de cache.

On peut aussi faire des mélanges de systèmes de cache. Pourquoi ne pas avoir les fiches-produits en dur dans le cache, et les quantités chargées par un script ajax, plus léger, pour le visiteur ? Et si on faisait cela aussi pour le panier ?



Et on peut même aller plus loin : pourquoi ne pas fournir à Google sa propre version du cache ? Une version dont on aurait enlevé les appels Javascripts inutiles, et toutes les fonctionnalités « à la mode » que seul un véritable internaute utilisera ?

En faisant de la détection sur user-agent on peut facilement détecter GoogleBot, et pour peut qu'on fasse aussi du reverse DNS, on met toutes les chances de notre côté pour ne jamais se tromper [https://support.google.com/webmasters/answer/80553?hl=fr].
La résolution DNS étant couteuse, on gardera en cache les associations IP/Identité, bien sûr.

Attention, en servant une version spécifique du site aux robots d'indexation de Google, vous faites ce que l'on appelle du cloacking (le moteur voit une autre information que celle qui est servie à l'internaute). Si l'utilisation de ce procédé semble légitime pour nous, on peut se demander s'il sera perçu comme tel par la firme de Mountain View. Une astuce intéressante donc, mais à ne pas mettre entre toutes les mains.

Utiliser le cache du navigateur

Mais le cache ne se limite pas qu'au serveur. Pour alléger ce dernier, le plus efficace est encore que le client ne télécharge pas certaines ressources.

Bien sûr, il a besoin de les télécharger au moins la première fois qu'il les rencontre, mais ensuite, sur les autres pages du site où elles sont à nouveau présentes, on va s'assurer qu'il se contente de les lire à partir du cache de son navigateur web.

Pour cela, on va configurer le serveur pour qu'il donne au navigateur une durée pendant laquelle mettre en cache les ressources [*mod_expires* pour Apache 2 - http://httpd.apache.org/docs/2.2/mod/mod_expires.html].

Utiliser le cache d'OPcode

Si vous utilisez PHP comme langage pour générer vos pages web, vous pouvez aller encore plus loin dans l'utilisation du cache en utilisant un cache d'OPcode.

Il s'agit de stocker l'étape d'analyse du code PHP, ce qui permet, lors des appels suivants d'un même script de ne pas avoir à ré-analyser le code. Cela génère souvent un gain de temps important de plusieurs dizaines de pourcents, ce qui est formidablement efficace quand on sait qu'installer un cache d'OPcode se fait généralement en quelques minutes, sans heurt.

Parmi les systèmes de caches d'OPcode, voici deux solutions solides : APC [http://php.net/manual/fr/book.apc.php] et Zend OPCache [http://pecl.php.net/package/ZendOpcache].

Compression des pages

La compression des pages est le gain de temps facile par excellence. Elle permet de réduire le poids des fichiers html qui transitent entre le serveur et le client. C'est particulièrement facile à mettre en place car tout bon serveur web qui se respecte dispose de paramètres ou d'une extension dédiée pour activer cette compression [*mod_deflate* d'Apache 2 - http://httpd.apache.org/docs/current/mod/mod_deflate.html]. Ce qui est moins évident par contre, c'est d'estimer le facteur de compression à appliquer : une petite compression sauvera peu de bande passante, mais permettra de peu solliciter le processeur, tandis qu'une compression élevée demandera plus de puissance au niveau du serveur, mais allégera les transferts.

Si vous ne voulez pas vous poser de question, compressez fortement, en interne, puis placez en cache le fichier pré-compressé. C'est lui qui sera délivré lors de la consultation par un internaute.

Optimisation de la base de données

Si vous avez un site web « classique », votre base de données est sans doute MySQL. Il existe de nombreuses optimisations à faire, au niveau du fichier de configuration général, principalement pour adapter les ressources de MySQL à votre trafic et à votre volume de données. Pour vous aider dans cette tâche, on pourra essayer de tirer quelques informations du MySQL Performance Tuning Primer Script [http://www.day32.com/MySQL/], mais s'il y a bien une chose qu'il faut garder en tête, c'est qu'il est très difficile d'automatiser une optimisation parfaite. Donc, il faudra peut-être tâtonner un peu avant de trouver les meilleurs paramètres.

Quoiqu'il en soit, même si la configuration générale est bonne, mais pas exceptionnelle, ce n'est pas à ce niveau que tout se jouera. C'est au niveau des requêtes MySQL que vous faites dans vos scripts et à la structure des bases que vous pourrez faire la différence.

Par exemple, oublier un index dans une table peut créer des dizaines de secondes d'attente. De même, une requête qui récupère des millions d'enregistrement avant de les traiter pour n'en garder qu'une dizaine, peut parfois se transformer de façon à directement réduire l'échantillon à traiter.

Pour devenir conscient des requêtes MySQL particulièrement problématiques, on peut activer la journalisation :
log_slow_queries = /var/log/mysql/mysql-slow.log (emplacement du fichier de log)
long_query_time = 2 (on veut récupérer la liste des toutes les requêtes qui prennent plus de deux secondes à s'exécuter)

log-queries-not-using-indexes (on journalise les requêtes qui n'utilisent pas d'index, ou qui récupèrent l'ensemble des enregistrements, ce qui est souvent un problème)

Utiliser un serveur plus léger ?

On a tous entendu parler de serveurs web qui seraient plus efficaces qu'Apache. Il faut dire que si celui-ci est vraiment très polyvalent, c'est une usine à gaz pour la plupart des usages. Oui, on sait bien le configurer car depuis le temps qu'il est dans le paysage, on a pu tout essayer. Passer à une autre solution serait prendre le risque de perdre la maîtrise du système.

Et puis, un jour, on a un peu trop de temps libre. Et on se décide à installer nginx [http://nginx.org/]. Et on se rend compte que la configuration est finalement loin d'être compliquée, qu'on peut faire du rewriting de façon très comparable à ce qu'on fait avec Apache, et qu'en plus on peut gérer facilement un système de cache, servir du statique à la vitesse de l'éclair, générer du dynamique en interfaçant avec php-fpm, et toujours en gardant une charge raisonnable sur le serveur et une mémoire tranquille.

On délivre plus vite les pages, on augmente la capacité d'accueil du serveur. Pourquoi n'a-t-on pas essayé avant ?

La transition [https://www.digitalocean.com/community/articles/how-to-migrate-from-an-apache-web-server-to-nginx-on-an-ubuntu-vps] mérite tout de même de l'attention : il y a moins de documentation sur nginx, et il vaudra mieux faire des tests avant de lancer en production !

Nginx n'est bien sûr pas le seul serveur alternatif à Apache, mais celui-ci est testé et approuvé. Si vous vous sentez l'âme aventurière, vous pouvez essayer Hiawatha [https://www.hiawatha-webserver.org/download], Cherokee [http://cherokee-project.com/] (qui a le mérite d'avoir une interface user friendly) ou encore Lighttpd [http://www.lighttpd.net/].

Utiliser un reverse-proxy ?

Nous avons évoqué la bonne pratique d'avoir toujours un cache statique à délivrer plutôt qu'une version dynamique, plus lente. Si on pousse l'idée à l'extrême, on peut imaginer que ce cache se transforme en brique logicielle, pour tirer le meilleur parti de deux univers.

On peut ainsi faire appel à Varnish [https://www.varnish-cache.org/], une application web spécialisée dans la délivrance ultra-rapide de contenu et de résistance à la charge.
En d'autres termes, c'est un générateur de cache web. On le place devant le serveur web « classique ». On parle de reverse-proxy car là où le proxy est dédié à permettre à un utilisateur à accéder au web, le reverse-proxy permet à un utilisateur d'accéder à des serveurs internes (le serveur web, qui n'est plus accessible directement). Il va générer le cache automatiquement en faisant passer, seulement quand c'est nécessaire, les requêtes à Apache, puis en recueillera les fruits. La fois suivante, il pourra servir directement le cache.

Mettre en place Varnish n'est pas très facile, mais les gains en temps de chargement sont vraiment importants, ce qui permet, de fait, d'accueillir énormément de visiteurs avec un matériel raisonnable.

Mettre en place un reverse-proxy n'est pas la première chose à faire si vous n'avez pas d'expérience dans ce domaine et si vous n'avez pas une problématique de fort trafic : la complexité qu'on ajoute en plaçant un serveur en frontal devant un autre doit être bien maîtrisée.

De même, maintenant que nginx est un projet mûr, avec des performances attirantes et un système de cache intégré, on peut avoir envie de n'utiliser qu'un seul serveur web, efficace lui aussi.

Mais Varnish a un autre avantage : il gère le load balancing nativement. Parfois, c'est essentiel: le cache permet certes de soulager un serveur de manière considérable, mais quand il y a un fort trafic, si fort que même délivrer une versions statique n'est plus possible, il faut songer à répartir l'afflux de visiteurs entre plusieurs machines.

Conclusion

Améliorer la vitesse de chargement d'un site est une tâche qui concerne de nombreux leviers et de nombreuses actions. L'enjeu est souvent de décider quelles sont celles à mener en premier, parce qu'elles sont faciles à mettre en place, ou parce qu'elles vont avoir un effet optimal.

Pour le référencement web, on privilégiera l'optimisation du back end, sans oublier le front end des versions mobiles. Mais à vrai dire, on sera davantage porté sur la résolution de problèmes rencontrés (absence de crawl de GoogleBot, pages extrêmement lentes non indexées) que sur une véritable optimisation préventive. On gardera également toujours en tête que le TTFB semble être la mesure adoptée par GoogleBot pour estimer la vitesse de chargement d'un site.

Pour la conversion, c'est tout autre chose. Là, on a carte blanche pour tout faire, absolument tout. Il faut simplement suivre les effets des actions mises en place. Tant que les conversions augmentent (quel que soit le critère utilisé derrière le terme conversion), il faut continuer à optimiser !

***Sylvain Peyronnet**, Professeur des Universités à l'Université de Caen Basse-Normandie (http://sylvain.berbiqui.org/) et **Guillaume Peyronnet**, gérant de Nalrem Médias (http://www.gpeyronnet.fr/nalrem-medias.html). Ensemble, ils font des formations (http://www.peyronnet.eu/blog/masterclass-moteurs-seo/) et essaient de battre les loutres à la pêche à la truite.*