

Maximum Performance : Accélérer l'affichage de vos pages web

Cyril Godefroy

Deuxième édition , Novembre 2007

Table des matières

Introduction.....	9
Pourquoi optimiser?.....	11
Baisser les coûts	11
Mieux maîtriser son architecture	11
Amélioration de la qualité perçue	12
PARTIE I.....	13
Plantons le décor.....	13
Rappels théoriques :.....	14
Afficher des pages web.....	14
Informations généralement trouvées dans les en têtes de requêtes	16
Informations trouvées dans les en têtes de réponses	16
Exemples d'échanges entre un navigateur et un serveur	17
Les types de contenus chargés.....	19

HTML 19

CSS 19

Javascript 19

Images 19

Multimedia 20

Temps de chargement de page vs temps de chargement html. 21

Les outils pour analyser son temps de chargement..... 23

Firebug 23

HttpWatch 25

Safari Web Inspector 26

Yslow 28

PARTIE II..... 30

Comment accélérer des pages web..... 30

Comment avoir une très bonne performance..... 31

La règle des 80/20 31

Télécharger les bons fichiers au bon moment.....35

Les navigateurs sont différents 36

Ce qu'il y a dans le header, c'est important 36

Le bon doctype pour le bon parser 40

Ce que l'on apprend des tests mis en ligne 41

Le chargement des css 41

Le chargement des javascripts est séquentiel 43

Télécharger plus vite en téléchargeant mieux..... 46

Faire du chargement parallèle sans devoir tout casser 48

Les limites du téléchargement parallèle 49

Améliorer le cache 50

Donner des directives de cache le long de la chaîne 51

Les directives à utiliser pour le cache 52

Comment utiliser les directives de cache? 54

Technique 0 55

Technique 1 55

Technique 2 57

Technique3 57

Le cache n'est pas tellement utilisé 57

Cacheability 58

Le temps de téléchargement c'est aussi du temps de requêtes.....59

Gérer ses cookies avec précision par domaine et sous domaine 60

Comment se débarrasser des cookies 63

Réduire la taille des pages sécurisées..... 64

Le https c'est le mal 64

Le https est indispensable 64

Gérer des ressources rares 66

https = pas d'astuces d'amélioration 66

Comment réduire l'impact des pages https 67

Des frames? Oui encore un peu, merci.. 67

Faire des redirections	68
Compresser pour aller plus vite.....	69
Comment fonctionne la compression?	69
Quels éléments compresser	69
Comment compresser avec Apache	70
Compresser avec PHP	71
Compresser (minifier) les ressources javascript	72
Utiliser un CDN.....	73
CDN ques aquo?	73
PARTIE III.....	75
Appliquer les meilleures pratiques.....	75
sur son site.....	75
Comment analyser et améliorer son propre temps de chargement.....	76
Fixer des objectifs réalistes	76

Rassembler les données des parcours les plus importants	78
Repérez les pages de type text/html qui prennent du temps	80
Faites la chasse aux redirections	80
Comparez la taille des requêtes et des réponses	81
Vérifiez l'usage du cache	82
Quelles ressources sont téléchargées en premier?	83
Identifiez les pages qui ont besoin d'être refondues	84
Une home qui fait snap!.....	85
Tout envoyer en une requête	86
Et le web 2.0 dans tout ça?.....	89
Conclusion.....	91
Annexes	93
Les sites à tester	93
Les articles de référence	93
Articles de cyrilgodefroy.com/fr/.....	95

Introduction

Durant mes quelques années d'expérience dans le domaine de la création et du développement de sites web (12 ans quand même), j'ai été plusieurs fois confronté à des problématiques de performance des sites. A chaque fois, l'exigence est devenue plus forte, au fur et à mesure que la vitesse de connexion des internautes augmentait, que les fonctionnalités étaient évoluées, que le contenu était riche. J'ai travaillé d'abord essentiellement sur des sites où les problèmes de performance se situaient au niveau du code sur le serveur. Mais au fur et à mesure de l'enrichissement des sites web, ce n'est plus cet aspect qui avait de l'importance. C'est le temps d'affichage complet de la page qui a pris le pas sur la rapidité de l'accès à la page.

Ce changement de mentalité est bénéfique à plusieurs titres :

- on s'intéresse de manière précise à ce qui est ressenti par l'internaute qui navigue sur le site,
- on partage la responsabilité de l'affichage de la page de manière transverse, du graphiste qui l'a dessinée au développeur qui l'a construite en passant par le serveur qui la donne,
- on change la vision que l'on a de la performance, d'une vision système à une vision client.

Les outils ont évolué en parallèle de la perception et des besoins des entreprises. J'ai connu un outil comme Witbe à ses débuts, à la fin des années 90, à une époque où il était très orienté système. A l'époque, on l'utilisait principalement pour avoir le temps de réponse et la disponibilité d'une page web html, sans prendre en compte le contenu

de ladite page. Aujourd'hui, certains des outils de mesure de Witbe permettent de réaliser des scénarii enchainant plusieurs pages, et d'autres outils permettent de faire des chargements de page complets avec de vrais navigateurs, tout en ayant encore la richesse statistique de l'outil.

Le savoir-faire à évolué aussi, ainsi que la manière de créer des sites. Du vilain site avec de nombreux tableaux imbriqués et de fausses images qui forçaient l'aspect et bloquaient les tailles des colonnes, cellules etc, on est passé aux sites exploitant enfin vraiment les feuilles de styles (css) et permettant d'accéder au contenu au travers du javascript pour modifier celui ci à la volée, télécharger de nouveaux fichiers, envoyer des mises à jour de données à la volée etc.

Cet ebook apporte sa pierre à l'édifice de la performance du web. Au travers d'une vue orientée internaute et client, on va aborder la question de la performance d'un site web, voir quels sont aujourd'hui les écueils les plus handicapants et les plus fréquents et les méthodes qui vous permettront d'améliorer vos performances, de baisser vos coûts, voire de gagner plus de clients. Evoluant entre des aspects fonctionnels, des aspects techniques, c'est une arme à mettre entre toutes les mains : du graphiste au chef de projet, de l'admin réseau au développeur, tous ont intérêt à faire des sites web plus beaux, plus efficaces et aussi plus rapides.

Pourquoi optimiser?

Optimiser la performance d'un site web, ce n'est pas qu'un vilain mot sorti de la bouche d'un expert fêru de temps de réponse. C'est avant tout un besoin marketing, la volonté de baisser le coût de fonctionnement, le besoin de répondre aux attentes du client. Il faut donc voir les mesures techniques qui contribuent à l'optimisation du temps de réponse et du temps d'affichage d'un site sous cet angle.

Baisser les coûts

En optimisant le temps d'affichage de votre site, vous allez économiser sur votre bande passante. L'économie ne sera peut être pas immédiate, elle ne sera peut être pas non plus définitive : il est évident qu'en améliorant la qualité de votre site, vous verrez plus d'utilisateurs, plus souvent, et qui amèneront plus de petits copains. Vous allez donc forcément augmenter le nombre de visiteurs et le nombre de pages vues. Mathématiquement, cela entraînera une augmentation de la bande passante : mais votre bande passante sera plus rentable.

Mieux maîtriser son architecture

Le fait de vouloir optimiser et rendre beaucoup plus performant vos pages web va vous forcer à avoir une meilleure maîtrise de votre architecture, et de connaître parfaitement celle ci : serveurs web, équipement de performance et de sécurité, utilité de ceux ci, systèmes de cache, flux existants.

Bien entendu, vous devez déjà connaître votre architecture, et être capable de valider l'usage de tel ou tel équipement et application dans un parcours web. Si vous n'êtes pas encore capable de le faire, soyez sûr que le fait de se pencher sur votre disponibilité et votre performance va vous forcer à le faire.

Amélioration de la qualité perçue

Si vous améliorez la performance de votre site, notamment sur ses pages les plus importantes, vous améliorerez la perception des utilisateurs : votre site est plus près plus réactif à leurs besoins, plus efficace et il est donc meilleur que celui de votre concurrent direct. Si votre site est un site de e-commerce, l'amélioration de la qualité se convertira automatiquement en espèces sonnantes et trébuchantes.

N'oubliez que votre concurrent sur internet n'est qu'à un clic de souris, et que les utilisateurs qui sont prêts à faire la queue pendant plusieurs minutes à la caisse, debout, pressés, sont les mêmes qui vont refuser d'attendre plus de 15 secondes sur le panier de votre site.

PARTIE I

Plantons le décor

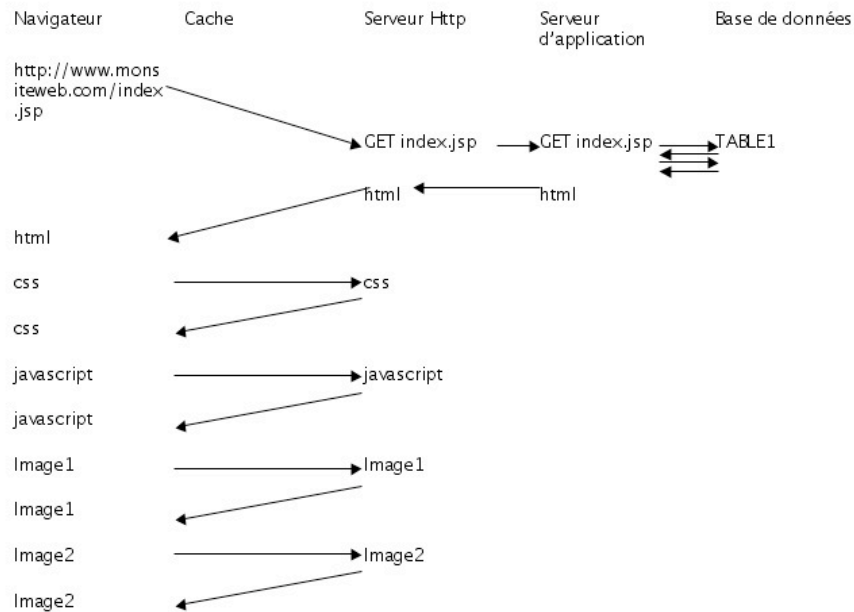
Rappels théoriques :

Afficher des pages web

Avant de commencer à voir dans le détail ce qui prend du temps dans le téléchargement d'une page web, on va reposer un peu les bases d'un certain nombre de fonctionnements du web de 2007.

Aujourd'hui, quand on visite un site web, on ouvre son navigateur préféré, qu'il s'agisse d'Internet Explorer, de Firefox, de Safari ou d'un autre encore plus exotique (Opera, Camino, lynx...). On va directement à une adresse en `www.quelquechose.fr` ou `.com`, on clique sur un lien, on remplit un formulaire, voire on accède à une application comme un webmail.

A chaque clic, on déclenche une requête http qui va chercher une ressource, en général une ressource html. Il y a donc du trafic réseau du navigateur vers le serveur (quel qu'il soit) que nous appellerons trafic montant. Cette requête donne lieu à une réponse du serveur. La réponse peut être une redirection, ou du contenu. S'il s'agit de contenu html, le navigateur attend d'avoir tout le contenu et commence à parser, c'est à dire à analyser et transformer en infos qu'il peut comprendre, le contenu. Il rencontre des balises, de nouvelles ressources qui lui sont indiquées comme nécessaires (une image, un fichier Flash, une feuille de style...). Pour chaque de ces ressources, il va initier un autre cycle requête/réponse. C'est à dire que pour chaque image, css, javascript, il va créer une requête, l'adresser au serveur et attendre que celui ci lui renvoie la ressource pour l'exploiter.



etc

A chacune des requêtes et des réponses, on va distinguer une partie en-tête (header) qui contient des méta données, et les données elles-même. L'en-tête contient des informations, le Referer (pour qui je demande cette ressource), les informations du navigateur, etc:

Informations généralement trouvées dans les en têtes de requêtes

Host
User-Agent
Accept
Accept-Language
Accept-Encoding
Accept-Charset
Keep-Alive
Connection
Referer
Cookie

Informations trouvées dans les en têtes de réponses

Date
Server
Last-Modified
Etag
Accept-Ranges
Vary
Content-Encoding
Content-Length
Content-Type

Ces entêtes ne sont pas limitatifs et nous verrons que d'autres existent qui ont un intérêt.

Quand on charge une page html, on ne sait pas d'où vient le contenu : on peut avoir une page de Yahoo, avec des images qui viennent de l'AFP, de la pub d'une régie, une autre d'une autre régie. Ces différents contenus sont fournis par différents services. Sans avoir différents services, on peut aussi avoir différents serveurs à l'intérieur d'un même domaine. Ce n'est le cas que quand on commence à avoir un gros serveur, et un gros service.

Exemples d'échanges entre un navigateur et un serveur

Request Headers

```
Host www.google.fr
User-Agent Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US;
rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4
Accept
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,t
ext/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language en-us,en;q=0.5
Accept-Encoding gzip,deflate
Accept-Charset ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive 300
Connection keep-alive
Cookie
PREF=ID=a4a.....07:LM=117605450.....6M1PV
```

Response Headers

```
Cache-Control private
Content-Type text/html; charset=UTF-8
Content-Encoding gzip
Server GWS/2.1
```

Content-Length 1688

Date Wed, 04 Jul 2007 20:15:45 GMT

Dans la vraie vie et avec les outils que l'on utilise cela donne ceci:

Headers	Response
Response Headers	
Age	0
Date	Sun, 11 Nov 2007 20:20:14 GMT
Content-Length	2571
Content-Type	image/gif
Cache-Control	max-age=3600
Server	Apache
Last-Modified	Sun, 28 Oct 2007 15:47:06 GMT
Etag	"30608-a0b-7eee3280"
SFRVia	su058719254 espacesfr-was
X-Cache	MISS from www.espacesfr.com
Request Headers	
Host	www.espacesfr.com
User-Agent	Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US; rv:1.8.1.9) Gecko/20071025 Firefox/2.0.0.9
Accept	image/png,*/*;q=0.5
Accept-Language	fr,en-us;q=0.7,en;q=0.3
Accept-Encoding	gzip,deflate
Accept-Charset	ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive	300
Connection	keep-alive
Referer	http://www.espacesfr.com/v5/webapp/wcs/stores/servlet/CategoryDisplay?catalogId=10001&storeId=10001&identifier=TLM&langId=-2
Cookie	s_cc=true; s_sq=sfrunvglobprod%2Csftrunvboutprod%3D%2526pid%253DWeb/Boutique/Les%252520Mobiles%2526pid%253D1%2526oid%253Dhttp%25253A/www.espacesfr.com/v/

La difficulté est souvent de faire le tri entre la performance des différents éléments qui font la rapidité d'une page, de simplifier les différentes requêtes et de trouver les fau-tifs. On peut s'appuyer sur les quelques outils qui sont fournis de base avec un naviga-teur, mais ceux ci s'avèrent rapidement insuffisants.

Les types de contenus chargés

HTML

Le HTML reste le principal format des pages web. Il s'agit d'un texte formaté avec des balises selon des règles assez strictes et précises qui évoluent avec les versions. Le type de version peut avoir une influence sur la performance de la page et son affichage, nous le verrons plus tard.

CSS

Les feuilles de styles (Cascading Style Sheets) sont des fichiers annexes aux pages HTML qui fournissent les règles de formatage typographiques des différents éléments de la page : paragraphes, images, titres, zones spécifiques (pieds de pages, menus etc). Les styles peuvent être insérés dans le cœur de la page ou en fichier externe (.css)

Javascript

Les fichiers javascript (js) sont des fichiers de code à exécuter par le navigateur, pas par le serveur. Ce code peut être inséré dans le cœur de la page ou en fichier externe.

Images

Les images d'un site web peuvent être de trois formats : gif (256 couleurs), jpeg (adapté aux photos), png (multiusage, destiné à internet). Un format vectoriel existe mais est très rarement utilisé : le svg.

Multimedia

Par multimedia, j'entends interactif et animé, ou vidéo. Les fichiers animés interactifs sont principalement des fichiers Flash, un format propriétaire même s'il est public. La vidéo est en général sous le format Flash Vidéo, vu l'ubiquité du lecteur Flash sur les navigateurs : la part des navigateurs équipés de la dernière version de Flash est estimée à 92%.

Temps de chargement de page vs temps de chargement html

L'outillage d'analyse de la performance est un outillage client. Ce n'est pas un outillage serveur, sur les performances des applications de serveur web ou de serveur métier.

Abordons la question de la performance des applications serveurs une dernière fois ici pour dire à quel point ce n'est pas le sujet de cet ebook, et ce n'est pas un sujet aussi complexe et mal maîtrisé que celui de la performance globale de la page.

Tout bon développeur se doit d'effectuer des tests sur les applications qu'il livre, notamment des tests de performance. Ces tests de performance existent pour vérifier qu'une application est capable, en fonction des services auxquelles elle est connectée et des serveurs sur lesquels elle est hébergée de servir le nombre de requêtes par secondes qui lui est demandé. Le niveau d'exigence de capacité vient du marketing, ou de quelqu'un avec une vision marché du service. Pour analyser des dérives qui pourraient se produire en production, ce développeur va ajouter tout un ensemble de traces, activables à la volée ou pas, qui lui permettront d'évaluer le temps de réponse d'une page de manière fine, en regardant où il perd du temps. Il pourra ensuite le faire sur une période un peu longue (quelques minutes ou quelques heures) pour rechercher la cause d'une lenteur. C'est le moins que l'on attend de lui.

Evidemment il y a le plus : la capacité à activer des logs en fonction de certains paramètres, par exemple uniquement sur certaines requêtes ou certaines pages. Ce plus

ne semble pas être à la portée de tout le monde, et la plupart des développeurs s'arrêtent à un tout ou rien : soit je trace tout, soit je ne trace rien...

Quelque soit l'analyse qui est menée à partir de ces tests et de ces traces, elle finit en général par une refonte d'architecture applicative, une augmentation de capacité de traitement, un achat de serveur, un ajout d'index : des choses connues et maîtrisées par les développeurs.

Mais les développeurs ont rarement une bonne vue de l'intégralité de la chaîne d'affichage de la page. Il comptent en requêtes par seconde sur une seule ressource, leur page. On va vite découvrir qu'ils ne sont pas en charge de la performance totale des pages.

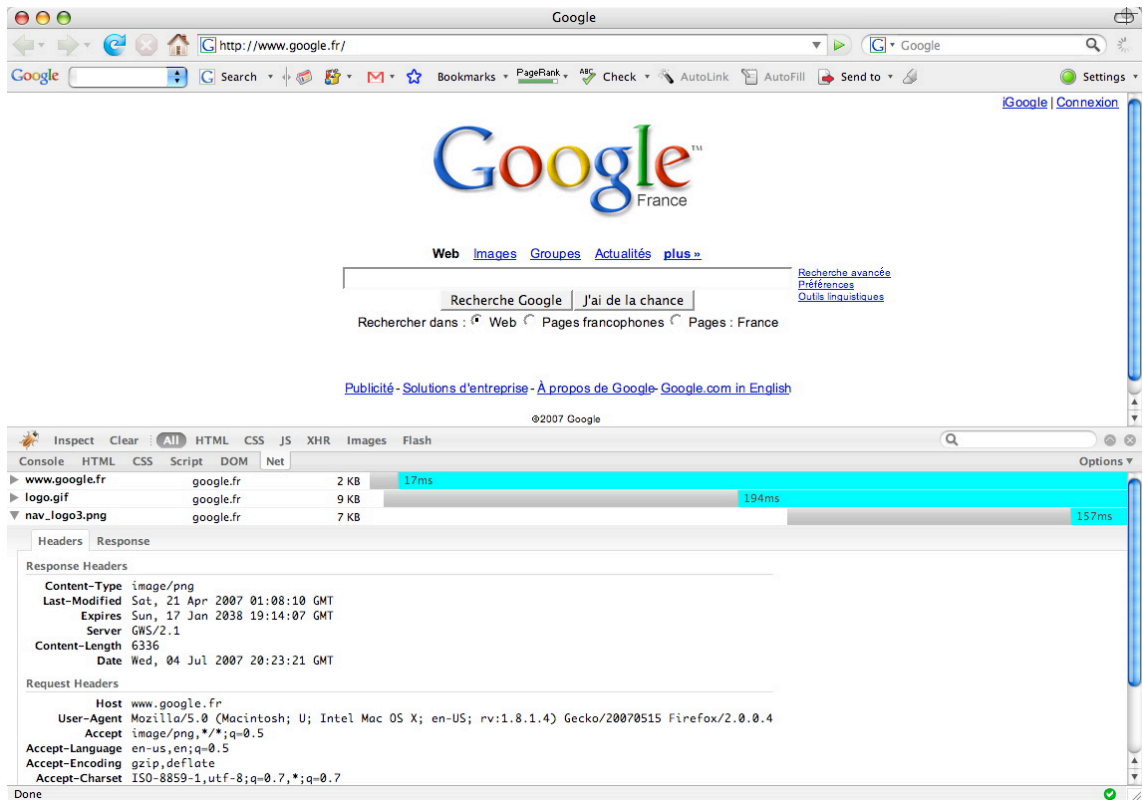
Les outils pour analyser son temps de chargement

Dans le sujet qui nous occupe, la performance vu du côté utilisateur, les outils sont relativement peu nombreux. J'en conseille quatre sur différents navigateurs.

Firebug

Firebug (<https://addons.mozilla.org/en-US/firefox/addon/1843>) est une extension pour Firefox. Pour l'installer sur Firefox, il suffit d'aller sur le site, de cliquer sur le logo. Firefox est protégé et refusera en premier lieu de le télécharger. Une fenêtre d'avertissement vous permettra d'accéder aux réglages permettant de rendre possible l'installation.

C'est un couteau suisse qui a plusieurs fonctionnalités. Celle qui nous intéresse est le dernier onglet 'Net'. Quand on accède à une page, ce module enregistre tous les entêtes de requête et de réponse. Ils les présente à la volée dans un chronogramme. Cela permet de voir comment, en combien de temps et à quel moment sont téléchargés tous les éléments de la page. Ensuite, chaque élément est accessible et on peut aller voir les entêtes de requête et de réponse.



Firebug en action sur la home de Google : les requêtes et pour chacune le détail des entêtes http

Deux défauts de Firebug :

- il prend mal en compte ce qui est dans le cache. C'est à dire qu'il représente quand même les éléments qui ne sont pas téléchargés mais viennent du cache du navigateur comme s'ils étaient téléchargés. Je n'apprécie pas personnellement la représentation du cache.

- il efface l'historique dès qu'on télécharge une page html : on n'a donc qu'une page visible à la fois, et il fait disparaître les redirections intermédiaires qui peuvent exister entre deux pages.

A part ces deux défauts, il est très agréable d'utilisation et permet d'avoir une bonne appréciation générale et en détail de ce qui se passe en affichant une page. Accessible d'un seul clic dans Firefox, il peut vous montrer ce qui s'est passé sur la page affichée sans que vous ayez besoin de le déclencher avant de la charger (c'est appréciable, je vous assure).

HttpWatch

<http://www.httpwatch.com>

HttpWatch est un plug in pour Internet Explorer, le navigateur qui domine le marché des navigateurs. Rien que pour cette raison, il est indispensable. Le comportement de Internet Explorer n'est en effet pas le même que celui de Firefox.

HttpWatch vient en deux versions : une gratuite le HttpWatch basic, et une payante. La version gratuite est suffisante dans la plupart des cas, surtout si vous associez l'usage de HttpWatch avec celui de Firebug ou Tamper Data.

La version gratuite se limite en effet à récupérer le démarrage, le temps complet du cycle requête-réponse, la taille de la requête et de la réponse, le code réponse et l'url de la requête. Elle ne vous montre pas le contenu des entêtes, il n'y a pas de chronogramme, etc.

Elle vous montre aussi un récapitulatif assez intéressant, avec la performance, les codes, le nombre de chargements, de connections etc. Quelques informations que vous ne trouverez pas, ou pas aussi bien présentées avec les deux outils précédents.

Le gros inconvénient de HttpWatch est son prix dans la version complète, plutôt excessif par rapport aux fonctionnalités.

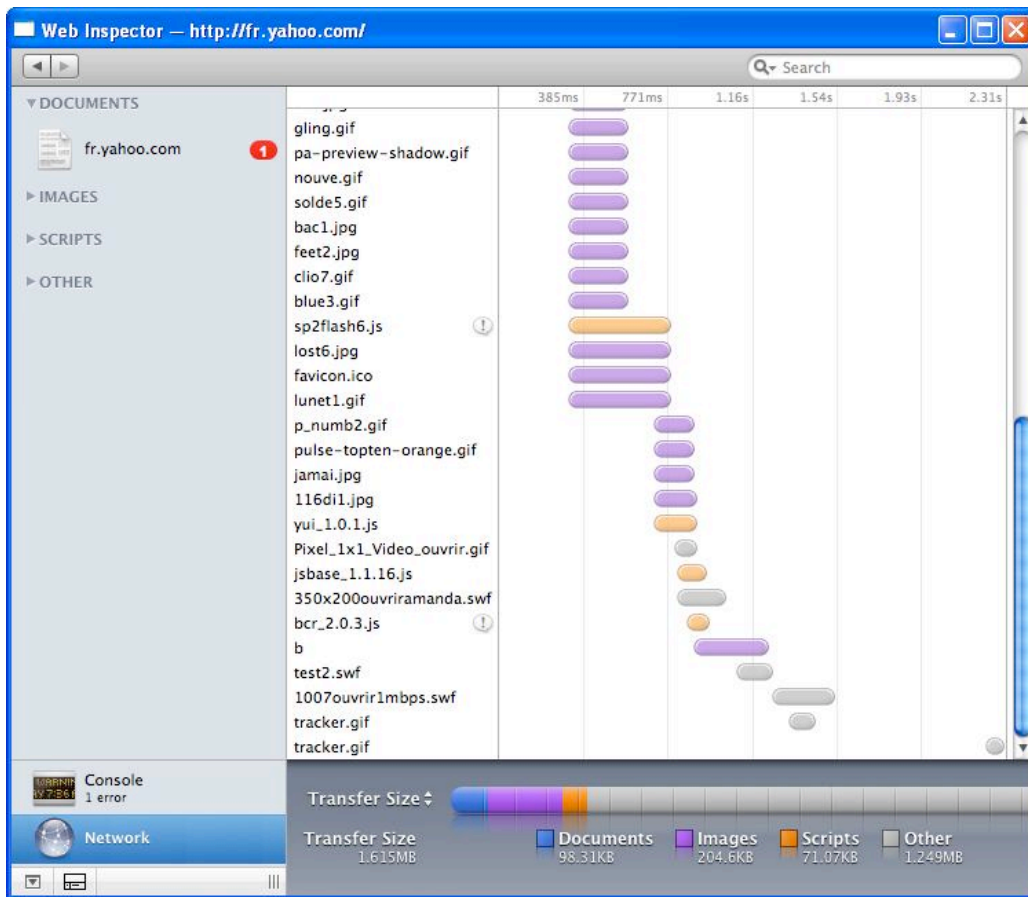
Ai je précisé que les deux outils fonctionnant avec Firefox fonctionnent de la même manière sur pc et sur mac? Pas besoin de redémarrer son mac sous windows avec boot camp ou de lancer Parallels pour tester un site.

Safari Web Inspector

<http://www.apple.com/fr/safari>

Nouveau et intéressant (tout chaud sorti du four à programmes), la beta de safari 3.0 fonctionne à la fois sur mac et sur pc (windows 2000, XP et vista). Le téléchargement est bien entendu gratuit (mais attention c'est une version beta c'est à dire en cours, et qui peut planter).

Le principal intérêt pour ce qui nous concerne vient d'un outil qui est intégré à cette version intermédiaire qui s'appelle le web inspector. Celui ci est disponible avec les dernières versions et permet d'un clic droit de la souris sur une page de visualiser plein d'informations intéressantes, comme le contenu source de chacun des fichiers de la page, le log javascript avec les erreurs rencontrées, et un onglet 'réseau' qui permet de voir comme avec Firebug les différents téléchargements, leur temps, leur ordre etc.



le web Inspector de Safari 3 beta sous Windows

Ici on le voit en action sur une page de Yahoo avec de la pub en vidéo (ce qui explique la taille gigantesque de la page). Ce que j'apprécie de prime abord sur cet outil est le rassemblement en différents items (javascripts, css, images, html) des différents objets

de la page, et la vue synthétique poids et temps de chargement ce qui permet de discerner rapidement ce qui prend du poids et du temps. Apparemment, pour rendre leur navigateur très rapide, les développeurs travaillant sur Safari ont boosté le nombre de connexions simultanées et on le voit bien ici avec toutes ces lignes ayant le même top départ.

Cet outil est encore un peu jeune et manque à mon avis de quelques informations et de quelques options. Il mérite d'être suivi. Etant open source (Webkit) vous pouvez intervenir sur les sources, voire les recompiler vous même si vous vous sentez l'âme d'un hacker. Vous pouvez surtout vous appuyer dessus pour ajouter les fonctionnalités que vous désirez.

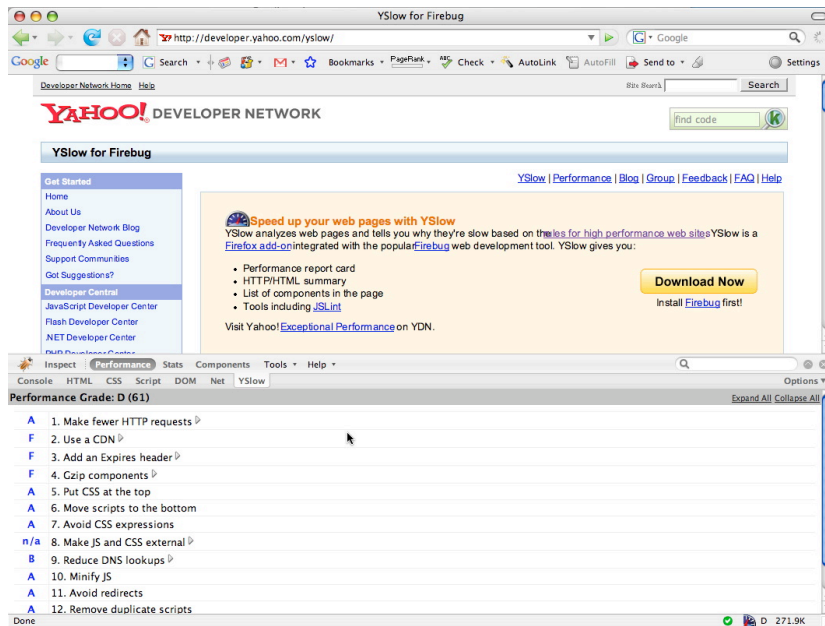
Yslow

<http://developer.yahoo.com/yslow/>

Yahoo performance group a sorti en juillet 2007 un outil qu'ils avaient depuis longtemps dans leurs cartons : Yslow.

C'est un plugin de plugin: il se branche sur Firebug pour faire des recommandations. Il vous donne des notes selon certains critères (un peu différents de ceux évoqués plus tard ici), ce qui vous permet d'évaluer la qualité de vos performances.

L'onglet performance vous permet d'évaluer votre conformance ux recommandations du Yahoo Performance Group. Personnellement je ne suis pas convaincu par toutes ces recommandations, mais elles ne sont pas plus mauvaises que d'autres ;-)



Le jugement de YSlow sur ses pages : D

Empty Cache		Full Cache	
4.2K	1 HTML document (est)	4.2K	1 HTML document (est)
19.0K	3 Style Sheet Files	0.0K	3 Style Sheet Files
3.1K	4 JavaScript Files	0.0K	3 JavaScript Files
242.9K	17 Images	0.0K	11 Images
2.5K	3 CSS Images		
271.9K	Total size	4.2K	Total size
28	HTTP requests	18	HTTP requests

Cookies: 24 bytes
 (24 bytes) B=257ge1h38o0ik&b=3&s=n6;

Longlet Stats : savoir combien on peut gagner sur le cache et les cookies

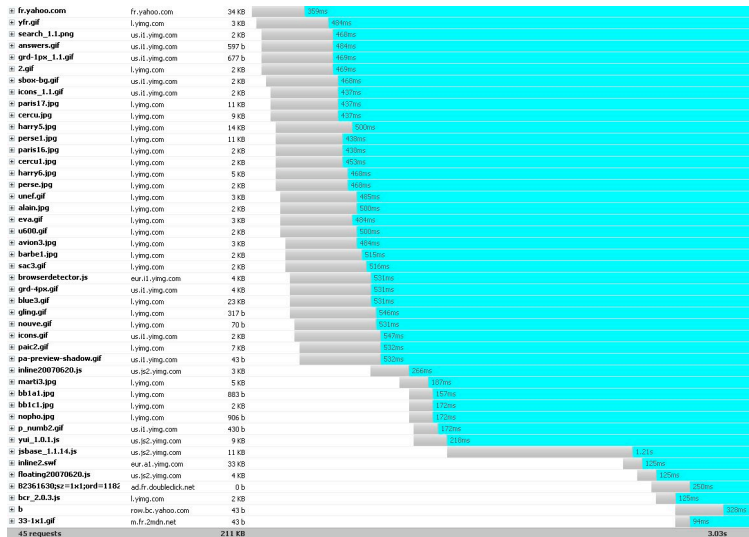
PARTIE II

Comment accélérer des pages web

Comment avoir une très bonne performance

La règle des 80/20

Dans une dérive de la loi de Pareto (ce sont 20% des clients qui génèrent 80% des plaintes, ou 20% des plus riches détiennent 80% des richesses), on peut dire que l'on consacre 80% de ses efforts sur ce qui ne représente que 20% du temps de chargement : le html. Si on regarde des exemples réels de temps de chargement, on est rapidement étonnés de voir que celui ci ne représente qu'une part ridicule du temps d'affichage :



les requêtes de chargement de la page d'accueil de Yahoo France. Le html c'est la première en haut à gauche.

Par exemple, le chargement de la page d'accueil de Yahoo prend soit 359ms, soit 3.09s selon le point de vue que l'on choisit : 359ms pour la page html seule, le reste étant dévolu au chargement de toutes les ressources qui vont permettre de construire et d'afficher la page dans son ensemble. Dans ce cas, le html ne représente que 10% et quelques du temps d'affichage de la page.

Faire porter ses efforts sur le fichier html est donc une aberration. Une aberration qui peut avoir du bon dans des cas extrêmes, comme la page d'accueil de Google, mais seulement dans ces cas :

+	www.google.fr	google.fr	2 KB	63ms
+	logo.gif	google.fr	9 KB	78ms
2 requests			11 KB	94ms

les même requêtes chez Google : minimaliste

Temps de chargement total : 94ms, dont temps de chargement du html: 63ms.

On peut varier les exemples sur toute une série de sites web :

Site web	Temps d'affichage	Temps HTML
Yahoo.fr	3.09 s	359 ms
Google.fr	94 ms	63 ms
amazon.fr	938 ms	407 ms
france2.fr	4.71 s	63 ms
orange.fr	4.3 s	32 ms
bouyguestelecom.fr	2.29 s	63 ms
sfr.fr	2.02 s	31 ms
liberation.fr	12.7 s	156 ms
flickr.com	828 ms	16 ms
lemonde.fr	8.88 s	188 ms
cyrilgodefroy.com	1.52 s	94 ms

Site web	Temps d'affichage	Temps HTML
msn.fr	3.67 s	102 ms
skyblog.com	4.75 s	89 ms
free.fr	2.53 s	1.53 s
dailymotion	6.97 s	359 ms

Bien évidemment, il n'est pas question de faire des efforts que sur le temps html puisque celui ci ne représente qu'une toute petite part du temps de chargement et d'affichage complet de la page. C'est le temps de chargement de tous les autres éléments qui va permettre de faire la différence. Imaginez si vous gagnez 10% de performance sur le html sur la home de yahoo, vous gagnez 30ms. Si vous améliorez le reste, vous gagnez 300ms : 10 fois plus. C'est encore plus vrai sur le site de France 2, ou de Libération... Comme on le verra, on peut gagner plus que 10%.

**80 à 90% du temps d'affichage d'une page est passé
côté client. Commencez par là!**

Télécharger les bons fichiers au bon moment

Certains créateurs de sites web perdent le sens des réalités à force de faire des tests sur leur machine locale ou sur un réseau très rapide. La puissance des machines peut avoir un effet masquant des problèmes les plus importants. Je me suis donc mis à la place d'un internaute moyen et j'ai cherché à reprendre le b-a-ba du chargement de page : qu'est ce qui prend du temps, qu'est ce qui bloque, comment se déclenchent les téléchargements.

J'ai pour cela créé plusieurs pages très simples, avec beaucoup d'éléments en général pour exagérer volontairement les résultats. Ces tests sont accessibles à l'adresse <http://cyrilgodefroy.com/fr/testqos/> . Si vous désirez exploiter à outrance ces tests, je vous invite à télécharger ces tests disponibles sur la même page et à les installer chez vous.

Ces tests ne sont pas forcément très durs à réaliser, et n'apportent pas toujours des découvertes phénoménales. Ils permettent de valider ce que l'on sait déjà peu ou prou.

Ces fichiers de test avec les mesures que l'on peut effectuer en les exécutant avec Internet Explorer et Firefox nous apprennent beaucoup de choses sur ces navigateurs, ou valident des choses dont nous nous doutions auparavant.

Les navigateurs sont différents

Cela paraît peut-être évident à la plupart d'entre nous : les équipes de IE et celles de Mozilla, bien qu'elles soient parties du même navigateur, Mosaic, ont divergé à plusieurs reprises, et leur stratégie est différente quant au téléchargement et au parsing des pages. Et il y a le rejeton WebKit (Safari pour le grand public), à l'origine seulement sur Mac, depuis quelques jours sur Windows.

Ces différences ne sont pas dramatiques, mais elles peuvent expliquer la différence de rapidité d'affichage de pages ou de parcours. Je me suis retrouvé dans certains cas à avoir jusqu'à 30% de différence entre Firefox et Internet Explorer sur un parcours. Les mesures avaient tendance à mettre en exergue cette différence.

Ce qu'il y a dans le header, c'est important

Les entêtes des fichiers HTML sont bien trop souvent ignorées par les développeurs.

Je ne parle même pas de la syntaxe des balises d'indication des pages. Non, je parle de ce qu'il y a dans la balise header. On a l'impression que c'est une zone de non droit où aucune règle n'est appliquée : plusieurs CSS, différents JS, des scripts en dur en plus, le tout sans logique ni réflexion.

Le cas le plus flagrant est la sérialisation du téléchargement des fichiers CSS et JavaScripts. Plutôt que de réfléchir, ou de faire du copier-coller pour concaténer le contenu de plusieurs fichiers dans un seul et sous le prétexte fallacieux de rendre modulaire, les développeurs rajoutent des liens vers des fichiers CSS et JavaScripts à tire-larigot. Un exemple ?

```
<script>
```

```

        __IS_NO_PROXY__ = true;
    </script>
    <script src="http://s3.unsiteweb.fr/js/css_ie_bug.js" ty-
pe="text/javascript"></script>
    <link rel="stylesheet"
href="http://s3.unsiteweb.fr/css/default_size.css" type="text/
css" />

    <link rel="stylesheet"
href="http://s4.unsiteweb.fr/css/default_struct.css" type="text/
css" />
    <link rel="stylesheet"
href="http://s4.unsiteweb.fr/css/default_blocks_size.css" ty-
pe="text/css" />
    <link rel="stylesheet"
href="http://s2.unsiteweb.fr/css/default_blocks_definitions.css"
type="text/css" />
    <link rel="stylesheet"
href="http://s2.unsiteweb.fr/css/default_nav_header.css" ty-
pe="text/css" />
    <link rel="stylesheet"
href="http://s2.unsiteweb.fr/css/default_nav_footer.css" ty-
pe="text/css" />
    <link rel="stylesheet"
href="http://s3.unsiteweb.fr/css/default_size.css" type="text/
css" />
    <link rel="stylesheet"
href="http://s3.unsiteweb.fr/css/common.css" type="text/css" />
    <script>
        document.cfg_default_path =
"http://s3.unsiteweb.fr/js/dlib4/";

```

```

</script>

<script src="http://s3.unsiteweb.fr/js/dlib4/dlib4.07.js"
type="text/javascript"></script>
<script type="text/javascript">
    dlib_load_extension("layers");
    dlib_load_extension("events");
</script>
<script src="http://www.unsiteweb.fr/nav/nav_js.jsp" ty-
pe="text/javascript"></script>
<script src="http://s3.unsiteweb.fr/js/nav_left.js" ty-
pe="text/javascript"></script>
<script src="http://s3.unsiteweb.fr/js/nav_top.js" ty-
pe="text/javascript"></script>
<script src="http://s3.unsiteweb.fr/js/common.js" ty-
pe="text/javascript"></script>

<script src="http://s3.unsiteweb.fr/js/scroll.js" ty-
pe="text/javascript"></script>
<script src="http://s3.unsiteweb.fr/js/m_select.js" ty-
pe="text/javascript"></script>
<noscript>Votre navigateur doit supporter Javascript pour
visualiser le site.</noscript>
<link rel="stylesheet"
href="http://s4.unsiteweb.fr/css/musique_jeux/common.css" ty-
pe="text/css" />
<link rel="stylesheet"
href="http://s1.unsiteweb.fr/css/musique_jeux/nav_left.css" ty-
pe="text/css" />

```

```
<link rel="icon" href="images/common/favicon.ico" type="image/x-icon" />
```

```
<link rel="shortcut icon" href="images/common/favicon.ico" type="image/x-icon" />
```

```
<link rel="stylesheet" href="http://mus3.unsiteweb.fr/css/unsiteweb_music.css" type="text/css" />
```

```
<link rel="stylesheet" href="http://mus2.unsiteweb.fr/css/musique_jeux/music.css" type="text/css" />
```

```
<script src="http://mus3.unsiteweb.fr/js/unsiteweb_music.js" type="text/javascript"></script>
```

```
<script type="text/javascript" src="http://mus3.unsiteweb.fr/js/lib_utils.js"></script>
```

```
<script type="text/javascript" src="http://mus3.unsiteweb.fr/js/class_ToolTipBox.js"></script>
```

```
<script type="text/javascript" src="http://mus3.unsiteweb.fr/js/class_ToolTipBoxManager.js"></script>
```

```
<script type="text/javascript" src="http://mus3.unsiteweb.fr/js/ttb_settings.js"></script>
```

```
<script src="http://mus3.unsiteweb.fr/js/navg_unsitewebmusic.js" type="text/javascript"></script>
```

```
<!--<script src="js/mouseEvtHandler.js" type="text/javascript"></script>-->
```

Désolé! Ce n'est pas très sympa de vous infliger cela. Mais c'est un exemple réel, j'ai juste honte pour eux, et je cache (un peu) leur vrai nom.

Le bon doctype pour le bon parser

Un petit aparté sur le doctype de vos fichiers. Si vous faites des fichiers qui s'appellent .html, qui sont servis par des serveurs http qui ont le mimetype de text/html réglé pour les fichiers .html, ne vous amusez pas à mettre un doctype en xhtml. Vous allez vous fatiguer pour rien, voire vous tirer dans le pied.

Cet article d'un contributeur Webkit (maciej) explique en anglais et dans des termes assez simples la question (<http://webkit.org/blog/68/understanding-html-xml-and-xhtml/>). Ce qu'il faut en retenir :

- faire un document en xhtml vous expose à des non affichages si le parser rencontre une erreur.
- Internet Explorer ne sait pas lire le xhtml.
- seul l'en tête http de MIME type application/xhtml+xml est traitée en xhtml. Les autres sont traités en html
- Il faut mettre un doctype HTML qui active le mode standard des navigateurs :
`<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">`

Ce que l'on apprend des tests mis en ligne

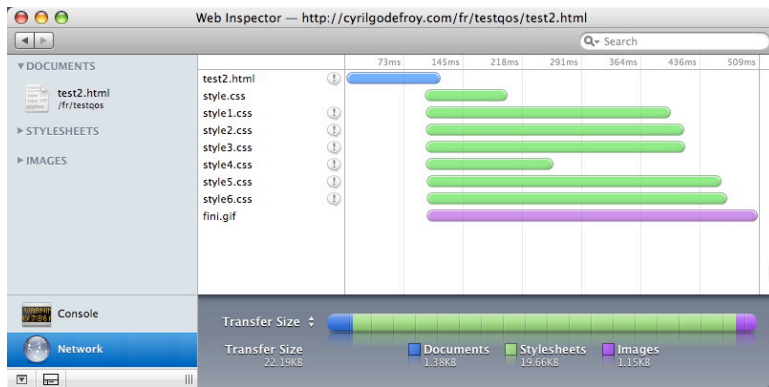
Pour travailler sur ma compréhension des navigateurs, de la façon dont ils chargent les fichiers et les utilisent, de quoi mettre où et quand, j'ai fait plusieurs pages de tests qui m'ont permis de vérifier des choses que je supposais, d'aller plus loin dans la compréhension des mécanismes des navigateurs. Ces tests sont loin d'être parfaits et je vous remercie de faire vos commentaires, d'ajouter vos observations. J'ai en effet conçu ces tests en fonction des comportements que j'ai observé, mais les choses sont supposées évoluer d'une version à l'autre d'un navigateur.

Le chargement des css

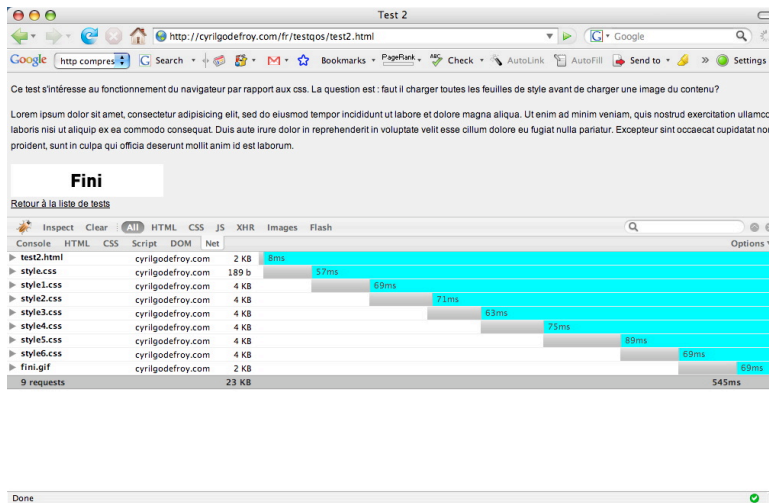
Suivant les navigateurs le chargement des css ne se fait pas de la même façon : sur IE et Safari, les fichiers css peuvent être chargés en même temps. Sur Firefox, le chargement des css est séquentiel : on finit de charger un css avant de lancer le chargement d'un autre.

David Hyatt (of WebKit Fame) a écrit une explication de ce qui se passe dans Firefox et Safari par rapport à l'exécution des styles. Je ne sais pas si c'est toujours vrai pour Safari, ni même pour Firefox, mais l'intérêt de son commentaire réside dans l'explication des affres auxquels il doit faire face concernant la question du rendu des styles.

Chargement des css ne veut pas dire utilisation des css. En effet, on peut être confronté à un autre problème quand le css est long : le flash de rendu qui se produit sous IE quand celui ci commence enfin à rendre les pages.



Sous Safari



Sous Firefox

Préoccupons nous en attendant du chargement (partie réseau) des css : je vous conseille de rassembler vos css dans le minimum de fichiers pour éviter de vous retrouver face au chargement séquentiel de Firefox. Mettez les au début de votre page html, dans l'entête du html.

Une question que je n'ai pas abordée car c'est une instruction que je n'ai pas rencontré depuis longtemps est l'usage du `@import` dans les css. Cette instruction des feuilles de style n'est activée qu'une fois que le css est complètement chargé et commence à être analysé par le navigateur. Dans ces conditions, on recrée un chargement séquentiel des fichiers css... Il y a peu, un développeur plein de bonne volonté m'a évoqué qu'il pourrait rapidement améliorer son site avec plusieurs css en insérant le deuxième dans le premier. C'était au cours d'une conférence téléphonique. Il a vite compris son erreur, quand ses collègues lui ont montré qu'il allait aggraver les choses.

Le chargement des javascripts est séquentiel

Ce comportement est prévisible car l'ordre des fonctions dans un fichier javascript a une importance vitale : je ne vais pas appeler une fonction dans un deuxième fichier javascript si je ne suis pas sûr que le premier fichier javascript qui contient cette fonction a été téléchargé. Ce chargement en séquence se produit que les fichiers javascripts se trouvent dans l'en tête html ou à l'extérieur.

Deux tests différents, les tests 8 et 9 permettent de vérifier ce que le chargement d'un javascript peut faire sur le chargement des images ou autres ressources statiques. Dans ces deux tests, j'ai mis les javascripts dans le corps du html. Jusqu'à là pas de miracle : le résultat de cet emplacement est exactement le même que celui du placement dans les en têtes. Par contre, le fait de rencontrer un fichier javascript reporte le chargement des images qui peuvent être devant ou derrière.

J'ai même poussé le vice jusqu'à faire un test 10 en mettant deux fichiers image entre deux fichiers javascript pour voir ce qu'il se passait : oh surprise, on lance bien le chargement de ces deux fichiers en parallèle, mais le fichier image qui est après les deux fichiers en question attend que le téléchargement du deuxième fichier javascript soit fini.

Ce qui se passe peut aussi conduire à écrire la règle suivante :

le chargement des javascripts retarde le chargement de tous les éléments qui se trouvent derrière

Je tire de tous ces tests une conclusion : il faut concentrer au maximum le contenu des javascripts dans un seul fichier, quitte à appeler les fonctions de ces fichiers en insérant directement les fonctions dans le code html. On risque sinon de voir tous les chargements d'images qui se trouvent après bloqués.

Le code que je vous ai montré plus haut est après ces deux séries de tests le pire que l'on puisse trouver :

- 13 fichiers de styles
- 14 appels de javascripts (presque 15)
- 1 fichier javascript avant le chargement des styles : ils doivent attendre la fin du chargement
- Des styles au milieu du lot des fichiers js

Je vous jure que cet exemple est pris sur un vrai site, vraiment public. Et ce site est censé recevoir une masse importante de trafic. Le meilleur est que dans le corps du html on trouve encore des appels à des scripts js.

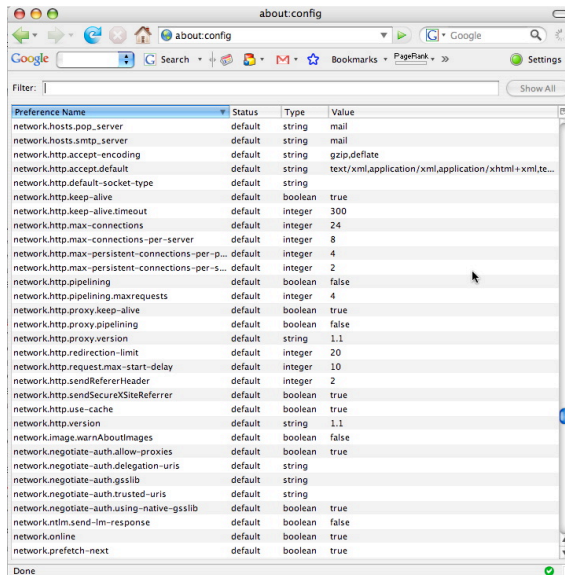
Télécharger plus vite en téléchargeant mieux

Une information dont certains développeurs de navigateurs se gardent bien de parler, et qui a une importance indéniable sur la rapidité de chargement d'une page complète est le nombre de connexions simultanées. On peut jouer sur ce paramètre pour absorber la charge sur les navigateurs qui souffrent le plus de nombre de connexions faibles.

Il est évident que si les navigateurs n'avaient qu'une connexion réseau (à un niveau applicatif, pas au niveau signal ou câble), les pages risqueraient d'apparaître beaucoup moins vite : les paquets se perdent (pas beaucoup, mais cela arrive), des contentions peuvent se produire. Le protocole TCP/IP permet de réduire les risques de perdre tout un contenu à cause d'un petit bout qui serait mal passé, mais il n'est pas parfait et ne permet pas d'assurer une qualité parfaite. D'autant que les éléments à chaque bout de la chaîne (navigateur et serveur) ont aussi des latences (des temps de retard) pour aller chercher une ressource ou pour la fournir.

Les navigateurs ont donc plusieurs connexions réseau ouvertes pour télécharger plusieurs ressources en parallèle. Firefox 2.0 vient de base avec une capacité de 8 connexions par serveur et 24 connexions en tout. Cela signifie que ce navigateur va lancer 8 téléchargements de ressources en concurrence vers un serveur par exemple `serveur1.siteweb.com`, et qu'il ne pourra pas télécharger plus de 24 ressources en même temps. Ensuite, il fait sa sauce si plus de 3 connexions serveur sont ouvertes en même temps.

Pour s'en rendre compte, on peut accéder aux configurations cachées en tapant l'url about:config dans la barre d'adresse. Le paramètre network.http.max-connections-per-server est à 8:



Un autre paramètre sur lequel le navigateur peut jouer pour télécharger plus vite des éléments est le pipelining : l'idée est de passer plusieurs requêtes dans la même connexion à la fois, et d'attendre la réponse à toutes ces requêtes avant d'en lancer d'autres. Cette méthode de téléchargement souffre de problèmes de fiabilités : si une réponse met du temps, toutes les réponses mettent du temps. Et cela mettrait à genoux les serveurs à fort trafic (imaginez devoir ouvrir une connexion et la laisser ouverte pour chaque utilisateur sans savoir quand celui ci va la refermer). Aujourd'hui, aucun navigateur n'est réglé par défaut de cette manière et ce n'est pas prêt de se produire.

On a parlé de Firefox, mais on n'a pas parlé du navigateur qui fait encore la majorité du parc : Internet Explorer 6 et 7. La version 6 d'Internet Explorer a par défaut moins de connexions ouvertes par serveur (<http://support.microsoft.com/kb/183110> et <http://support.microsoft.com/default.aspx?scid=kb;fr-fr;Q282402>). On ne peut donc pas paralléliser de la même façon le téléchargement des fichiers sur Internet Explorer et sur Firefox dans la version par défaut.

Dans le cas de Internet Explorer en https, la situation s'aggrave : le nombre de connexions simultanées en protocole https est encore inférieur. On peut voir le phénomène d'empilement des téléchargements s'aggraver. Dans HttpWatch, les téléchargements sont lancés à peu près au même moment, mais les fichiers mettent de plus en plus de temps à arriver.

J'ai mis sur le serveur <http://cyrilgodefroy.com/fr/testqos> des tests permettant d'observer le phénomène: essayez avec vos différents navigateurs pour voir comment se passe le chargement de fichiers différents dans la configuration par défaut.

Faire du chargement parallèle sans devoir tout casser

Pour faire du chargement parallèle de ressources sans devoir aller changer les configurations sur les navigateurs des visiteurs (ce qui n'est pas tout à fait faisable), il faut un autre serveur. On a en effet vu que le navigateur ouvre un certain nombre de connexions par serveur. Si on augmente le nombre de serveurs, on multiplie donc le nombre de canaux de téléchargement.

Je pourrais donc vous inviter à créer un serveur img.monsiteweb.com pour stocker vos ressources statiques images dessus, et de les mettre dessus. Pas forcément facile à faire en production, surtout si vous partez ensuite sur l'idée de faire des serveurs phy-

siquement différents, ou de stocker les fichiers dans des arbres différents de répertoires.

La solution qui vous permet de rapidement rajouter un serveur sans changer votre hébergement, en modifiant peu votre configuration serveur? Utilisez un CNAME dans le serveurs de noms de votre domaine. Le CNAME est un alias qui fait pointer notre serveur `img.monsiteweb.com` vers votre serveur `www.monsiteweb.com`. Il faut alors que votre serveur http soit capable de répondre à toutes les requêtes vers une adresse de type `*.monsiteweb.com` avec les mêmes réglages et le tour est joué. C'est cette technique que j'utilise notamment pour les pages de tests sur

<http://cyrilgodefroy.com/fr/testqos/>

Les limites du téléchargement parallèle

Ce travail d'utilisation de plusieurs noms de serveurs a des limites : en effet le fait d'ouvrir une connexion a un coût : résolution du nom de machine, ouverture de la connexion réseau, réutilisation après un téléchargement prend un petit peu de temps à chaque fois. Il y a une limite autour de 4 noms de serveurs à partir de laquelle le temps gagné à paralléliser est supplanté par le temps de toutes ces opérations.

Ainsi, il ne faut pas tomber d'un extrême à l'autre. Cela m'est arrivé sur un site : en 6 mois, on est passé d'un site qui utilisait deux domaines à 11 domaines, avec des agrégations de contenus dans tous les sens. L'utilisation de plusieurs sous domaines est alors devenue un écueil.

Là encore, j'ai préparé plusieurs séries de fichiers disponibles à partir de l'url `http://cyrilgodefroy.com/fr/testqos` qui permettent de valider avec son navigateur cible le comportement idéal à reproduire pour une page.

Améliorer le cache

Certains amis de monsieur La Palisse diraient que le moyen le plus rapide de fournir à un internaute un fichier c'est encore de faire en sorte qu'il l'ait déjà.

Ils n'ont pas complètement tort: c'est justement pour cela que les navigateurs ont inventé le cache. Celui ci est utilisé pour stocker les ressources déjà utilisées sur l'ordinateur ou le téléphone de l'internaute, en mémoire ou sur disque, et recharger ces ressources quand elles sont redemandées. Suivant les navigateurs, la taille du cache, le fonctionnement du cache est différent (Safari cache énormément par exemple). Il faut donc essayer de remplir au maximum le cache des navigateurs. Celui ci fournit aux navigateurs le moyen d'utiliser des fichiers sans qu'il soit nécessaire de les télécharger : une page dont toutes les ressources sont dans le cache se charge très très vite.

Là encore, j'ai fait plusieurs pages de tests avec des directives différentes pour vous donner les directives les plus efficaces et les techniques qui permettent d'améliorer le plus vos temps d'affichage.

Avant d'aborder la question des directives, réglons la question des meta headers. J'ai un secret pour vous: Ces balises à insérer dans le code html ne sont pas efficaces, et il vaut mieux ne pas les utiliser. Plusieurs raisons :

1. Que voulez vous cacher? Des fichiers html, ou des ressources qui sont utilisées au long d'un parcours? Mettre des pages html différentes dans le cache ne permet aucune

économie... en effet il y très peu de chance que vous reveniez sur la même page HTML au cours d'un parcours (même si c'est possible et malin).

2. Elles ne sont pas prises en compte dans le protocole http 1.1, la version actuelle. Seuls les entêtes http sont utilisés.

Donner des directives de cache le long de la chaîne

La version simple d'une requête http suppose qu'entre un navigateur web et le serveur web, il y a des équipements 'passifs' : routeurs, routeur ADSL, switches. Mais l'infrastructure peut être plus élaborée : présence de load balancers pour s'adresser à plusieurs serveurs web, existence de reverse proxies pour contrôler l'accès à certaines ressources mais pas d'autres, firewalls pour limiter le trafic réseau à certains ports.

Deux équipements sont importants dans la gestion du cache et viennent s'intercaler dans le flux entre le navigateur et le serveur :

- le proxy est un équipement qui se trouvera sur votre réseau local et contrôlera l'accès à internet (pas possible d'aller voir YouTube ou DailyMotion), tout en fournissant un cache local.
- le reverse-proxy fait la même chose mais au niveau de votre plate forme d'hébergement : contrôle d'accès et cache des objets statiques demandés aux serveurs web pour réduire la latence due au réseau et alléger la charge des serveurs.

Ces deux types d'équipements peuvent aussi exister chez votre FAI : l'usage de celui ci est alors transparent ou non. Il vise à réduire un peu la bande passante utilisée sur les liens inter-opérateurs et à fournir les objets plus vite.

Ces équipements peuvent accueillir vos ressources statiques. Il faut leur demander gentiment, être capable de leur fixer des règles pour qu'ils le fassent de la bonne manière pour les bonnes ressources. Sinon, vous pouvez vous retrouver à montrer (ou à voir) une page destinée à un autre internaute, parce que ces équipements ont 'trop' caché. Ou avec des ressources trop vieilles, parce que la durée fixée pour le cache est trop importante et aucune revalidation n'est demandée.

Les directives à utiliser pour le cache

Il existe plusieurs directives de cache.

La première et la plus importante est celle qui permet de ne pas avoir de cache. Elle est double (je vous conseille d'utiliser les deux directives, qui peut le plus peut le moins), car il y a la directive du http 1.0 et celle du http 1.1.

`Pragma: no-cache`
et

`Cache-Control: no-cache, no-store`

Les autres directives permettent de contrôler précisément le niveau et la durée de cache que l'on veut préconiser aux navigateurs et aux caches intermédiaires.

`max-age=`

La directive la plus vitale sert à préciser la durée de cache à appliquer par un navigateur à une ressource. La durée précisée est en secondes, ce qui explique que le nombre que l'on trouve derrière est parfois très important. Si `Cache-control: max-age=3600` apparaît dans une réponse par exemple, le navigateur va considérer la ressource à

jour pendant une heure. Cette directive est plus importante que la directive Expires, si les deux infos sont présentes dans l'en tête http.

private

private indique aux caches qu'il ne doit pas stocker la réponse au navigateur. Cela vaut pour les caches intermédiaires, comme les proxies et les reverse-proxies.

public

public indique au cache que la réponse peut être cachée alors qu'elle ne devrait pas l'être (comme par exemple parce qu'elle est authentifiée).

no-store

no-store indique que la ressource ne doit pas être stockée dans aucun type de mémoire non volatile. Cela permet d'assurer le maximum de confidentialité, hors chiffrement.

must-revalidate

Indique qu'un cache partagé ne doit renvoyer une ressource dont la date d'expiration est dépassée qu'après avoir validé sa validité auprès du serveur d'origine.

D'autres directives encore plus précises existent, que je vous invite à aller découvrir directement dans la RFC 2616 : elles sont peu utilisées, et n'ont pas lieu d'être décrites ici.

Comment utiliser les directives de cache?

Pour vous éviter des heures fastidieuses de recherche dans la littérature spécialisée et sur internet, voici quelques infos pour régler les directives de cache avec Apache.

Pourquoi Apache? Parce qu'il sert plus de 50% des pages web dans le monde. Il aurait été un peu idiot de les faire pour lighttpd (le serveur web que j'utilise pour mes sites), qui représente 0.005% des serveurs http. De plus, Apache définit un peu le mode standard de réglage de tous ces paramètres, étant lui même conçu et développé par des développeurs qui cherchent à respecter les standards, normes et recommandations.

Avec Apache, on utilise deux modules différents pour modifier les entêtes http concernant le cache: le module expire et le module header. Vous devriez vous référer à la documentation de ces modules (<http://httpd.apache.org/docs/2.0/mod/>) pour avoir le détail de leur fonctionnement. Le premier module contrôle deux infos des en têtes : Expires et Cache-Control: max-age . Le deuxième module permet de manière générale de modifier les entêtes http, nous nous en servons pour régler l'entête Cache-Control de manière spécifique.

Ces modules doivent être activés dans votre configuration Apache. Ensuite, vous pouvez utiliser leurs réglages de manière différente en fonction de vos besoins : dans le fichier principal de configuration d'Apache, dans un autre fichier inclus, répertoire par répertoire (section <Directory>), ou dans des fichiers .htaccess si vous avez activé ce réglage. Je vous conseille de les tester d'abord dans un fichier htaccess puis de les intégrer progressivement dans un fichier de réglages spécifique en include du fichier de configuration principal.

Expire fonctionne avec les directives ExpiresDefault, ExpiresByType, ExpiresActive.

Technique 0

Comment ne pas cacher

Pour ne pas cacher, il faut régler les entêtes http Pragma et Cache-Control:

```
<Directory "/home/website/cgi-bin/">  
    Header Set Pragma "no-cache"  
    Header Set Cache-Control "max-age=0, no-store, no-cache"  
</Directory>
```

Cela peut paraître bizarre de commencer par là, mais c'est certainement la directive de cache la plus importante pour vous, et le problème que vous vous poserez le plus souvent : comment ne pas cacher. Cela concerne tous les fichiers que vous voulez pouvoir changer rapidement, pour une raison (mauvaise) ou pour une autre (mauvaise aussi).

Cette directive doit surtout être rajoutée dans le cas d'un contenu généré dynamiquement et qui est forcément amené à être rafraîchi à chaque instant.

Technique 1

Faites un répertoire avec les images qui changent tous les 10 ans et mettez des directives agressives.

Il y a peu de chances pour que votre logo change tous les 3 jours ou que certaines images de votre site soient modifiées même tous les mois. Ces ressources peuvent être cachées sans limite de durée par les caches, quels qu'ils soient. N'hésitez pas à ajouter des directives agressives du type :

- mettre les ressources de ce répertoire en cache avec une date d'expiration dans 10 ans
- mettre toutes les ressources d'un type en cache avec une date d'expiration dans 10 ans

Pour le premier cas, cela donne ceci:

```
<Directory "/home/website/images/s">  
  Options FollowSymLinks MultiViews  
  AllowOverride All  
  Order allow,deny  
  Allow from all  
  ExpiresDefault "access plus 10 years"  
</Directory>
```

La deuxième stratégie donne

```
<Directory "/home/website">  
  Options FollowSymLinks MultiViews  
  AllowOverride All  
  Order allow,deny  
  Allow from all  
  ExpiresByType image/gif "access plus 10 years"  
  ExpiresByType image/jpg "access plus 10 years"  
  ExpiresByType image/png "access plus 10 years"  
  ExpiresByType application/x-shockwave-flash "access plus 10  
years"  
</Directory>
```


Technique 2

Changez le nom des ressources que vous voulez mettre à jour. En effet, si vous utilisez un cache de longue durée, il y a éventuellement un risque que certains équipements gardent les ressources en mémoire. Pour éviter ce risque, vous devez changer de ressource : en changeant de nom, vous mettez automatiquement à jour la ressource.

Par exemple, la méthode utilisée par les plus grands sites est de faire référence à des numéros de version ou de build dans leur noms de fichiers javascript ou autres. Ainsi, Yahoo a à l'instant où j'écris ces lignes utilise des scripts yub1.2.1.js_ bcr_2.0.4.js etc

Technique3

Mettez un cache court pour s'adapter à un parcours: une heure ou deux. Cela évitera les revalidations. Cette technique est illustrée sur le test des deux pages accessible sur cyrilgodefroy.com. Dans ce cas, j'ai mis un cache de 1 heure sur toutes les ressources du répertoire cache1 : les fichiers html, les fichiers css, les images... Ce qui passe en conséquence c'est qu'au premier accès, on télécharge tous les fichiers, et qu'en arrivant sur la page 2 du test, on cherche à télécharger à nouveau uniquement le fichier html (et encore, seulement sous Firefox).

Le cache n'est pas tellement utilisé

Le résultat de l'implantation des directives de cache peut sembler décevant quand on regarde la bande passante, et le nombre de requêtes totales sur une ressource censée être cachée. Yahoo a réalisé et partagé une étude (<http://yuiblog.com/blog/2007/01/04/performance-research-part-2/>) sur la question.

Cette étude montre que sur un site à très important trafic comme Yahoo, même sur une longue période, il reste 40% des utilisateurs qui arrivent avec un cache complètement vide.

La mise en place d'un très bon cache reste toutefois essentielle pour des parcours sans couture, utilisant au mieux les ressources d'un site web, et rendant inutile le téléchargement systématique des ressources. On doit pas s'appuyer là dessus en espérant que les visiteurs vont arriver avec un cache rempli. Il faut faire en sorte qu'au cours d'un parcours, un minimum de ressources soient rechargées et que les ressources communes soient parfaitement cachées.

Cacheability

<http://www.ircache.net/cgi-bin/cacheability.py>

C'est un outil automatique qui vérifie l'état de chacune des ressources d'une page. Cet outil ne fonctionne que sur internet et vérifie les en têtes http des ressources d'une page. Il peut prendre un peu de temps à fonctionner, car il doit télécharger et analyser chacune des ressources de la page que vous soumettez, et qu'il le fait séquentiellement.

Son principal défaut se situe au niveau de l'analyse des fichiers css : il n'en fait aucune.

Le temps de téléchargement c'est aussi du temps de requêtes

Bien sûr, vous pensez comme tout le monde qui a un peu de bon sens : ce qui prend du temps c'est la construction de la page et le temps de chargement des objets qui la composent (images, javascripts et autres). Vous seriez étonné de vous rendre compte de la masse d'informations qui font le trajet inverse: de votre navigateur vers le site web. Voici un exemple sur des parcours sur quelques sites :

Site	Upload	Download	Rapport D/U
Google	1286	30105	23,409797823
Bouygues Télécom	60037	692996	11,542815264
SFR	107744	457038	4,2418881794
Paypal	54459	264946	4,8650544446

Sur les pages d'accueil en général pas de problème. Mais commencez à rentrer dans le coeur du sujet, ajoutez des articles dans votre panier et authentifiez vous : vous verrez grimper votre trafic remontant vers le serveur.

Voici par exemple le résultat d'un parcours d'achat chez Amazon :

Download 852415

Upload 125708

(Achat du premier produit trouvé sur une recherche, avec compte existant, mais sans valider la commande).

Gérer ses cookies avec précision par domaine et sous domaine

Un cookie, c'est une information envoyée par le serveur web (en général par l'application qui est derrière) au navigateur, et que ledit navigateur doit ensuite renvoyer suivant certaines règles. Les cookies sont utilisés partout : pour la pub, pour l'identification, pour gérer un caddie, etc. Tout ce qui suppose la persistance de la session ou de l'identification nécessite en général un cookie. L'autre méthode pour identifier une session est un peu trop compliquée à mettre en oeuvre et pose des problèmes de référencement de contenu.

Le cookie simplifie la vie du développeur. Mais le développeur se simplifie en général trop la vie. Il peut en effet choisir certains paramètres pour le cookie comme le nom de domaine auquel le cookie doit être renvoyé par le navigateur et le chemin. Inutile de dire que le développeur choisira normalement le domaine le plus large (yahoo.com ou cyrilgodefroy.com par exemple) et le chemin lui ouvrant le plus de possibilités (/ en l'occurrence). Ce qui fait que toutes les requêtes envoyées par le navigateur une fois que le cookie est là contiendront le même cookie.

Un cookie peut avoir différentes tailles : la taille d'un identifiant, la taille d'un identifiant de session (plus gros), la taille d'un identifiant d'identité (encore plus gros). Donc la taille peut varier de 0 à 1000 ou 2000 octets. Et cette taille de cookie est reproduite à chaque requête sur une ressource du domaine et du chemin définis pour le cookie.

Site	Taille de requêtes
Paypal	1001 octets
Yahoo	356 octets
Google	438 octets
TF1	292 octets
Voila	246 octets
SFR	385 octets
Bouyges Télécom	685 octets
Orange	357 octets
MySpace	452 octets

Le tableau ci-dessus montre que suivant les sites web , les requêtes ont des tailles différentes, pour certaines assez importantes (Paypal, BouyguesTelecom).

Pour vous conforter dans l'idée que les cookies peuvent être lourd, en voici quelques uns:

- MySpace

```
MYUSERINFO=; domain=.myspace.com; expires=Wed, 19-Jan-2005
08:28:17 GMT; path=/ MYUSERINFO=; domain=myspace.com; expi-
```

res=Wed, 19-Jan-2005 08:28:17 GMT; path=/
MSCulture=IP=82.238.217.64&IPCulture=fr-FR&PreferredCulture=fr-FR&Country=FR&timeZone=0&ForcedExpiration=0&USRLOC=QXJlYUNvZGU9MCZDaXR5PVBhbGFpc2VhdSZDb3VudHJ5Q29kZT1GUiZDb3VudHJ5TmFtZT1GcmFuY2UmRG1hQ29kZT0wJkxhdG10dWRlPTQ4LDCxNjcmTG9uZ2l0dWRlPTIsMjUmUG9zdGFsQ29kZT0mUmVnaW9uTmFtZT1BOA==; domain=.myspace.com; expires=Sun, 15-Jul-2007 21:27:29 GMT; path=/

Je dis bravo : deux fois le même cookie, sur deux domaines différents...

- SFR

JSESSIONID=590A163AEFFD806BDE9ECEf37B3AF76F;
PSWsessionID=230925066.0.0000; s_cc=true; s_sq=%5B%5BB%5D%5D

- Paypal

KHcl0EuY7AKSMgfvHl7J5E7hPtK=GgdN_CkjiZ241UgTtTkkyGHJM6UzLHab3odd6dWHWlUwus3ZMsh3BL3RK8-YwzEvSzZ0Ay9BESomyLjW; cookie_check=yes; s_pers=%20s_favsn_paypalglobal_1%3D7628769626047%7C1499205713501%3B; login_email=...; upct=96; Apache=8////.....20999118236932791; SzWDZLlTw4729rjVwzEIbk4eZ7u=CMZp3C6FrsJHDUEHRNyKHszAfPrEuljq34IlvDlbUJxmO40wLsydPieCiNt_JSEZKXB6VnSMYNRd1OTTU_KLHTI-6k9asA1UZ3XKux2G; OL6S9fXi8y9NkAzRyU8cgsycsz0=FE-j1sULASmKiFjgOnM69JHCw0AXPa-tVDGQm9D4V5Uz4TNYBRlJnG3jQM9kYPRg6GGqIY0VVpyIWjW47xCZeyJUnxro4_z2j_2sv6Z1n8lfb21P; s_sess=%20s_cc%3Dtrue%3B%20s_refresh%3DHomepage%257E%255B1%255D%3B%20s_sq%3Dpaypalglobal%253D%252526pid%25253DHomepage%252526pidt%25253D1%252526oid%25253DLog%25252520In%252526oidt%25253D3%252526ot%25253DSUBMIT%3B; HaC80bwXscjqZ7KM6VOxULOB534=9uS_zYnvvEWYb_Ji7k7TDu3-UTNooFTK_h4M7JCWs-YMRfLDTX83BOGFgvEnJB0b-DkUWtrGh-nLXzaptAua3CI8Et8lndlp3lCF

```
Tvr9RFhdjRXjsti3raztZg704vksJ5cWBG; cookie_contact_phone=; cookie_welcome=  
- Google
```

```
PREF=ID=a4abafc91afcc710:TM=1176054507:LM=1176054507:S=MQ0FDkeKA  
B16M1PV
```

Comment se débarrasser des cookies

Il est plus facile pour un développeur de créer un cookie que pour vous de vous en débarrasser : une fois que ce genre d'information existe, il a tendance à se rendre indispensable et à conserver son emprise. On ne peut donc pas se débarrasser des cookies comme cela. Il faut éviter au maximum de les utiliser avant de les enlever.

La deuxième méthode à utiliser est de ne mettre les cookies que sur des sous domaines très précis, par exemple `www.yahoo.com`, ou `mail.google.com`. Ainsi quand on va sur `movies.yahoo.com` ou `ww.google.com`, on n'envoie pas ces cookies avec la requête. Si on crée un sous domaine `img` pour héberger uniquement les images (par exemple), les requêtes envoyées vers ce serveur `img.monsiteweb.com` ne contiendront pas de cookie déposés sur `www.monsiteweb.com`.

La troisième méthode consiste à se créer un autre domaine pour servir les images et les ressources statiques. C'est dans le cas où vous ne pouvez vraiment pas vous passer de mettre un cookie sur l'intégralité du domaine, sur `*.monsiteweb.com`. Cette méthode pose d'autres problèmes : alertes de sécurité qui peuvent survenir pour vous prévenir de cross site scripting (une technique de hacker pour passer des infos d'un site à l'autre). Par exemple `sfr` a `s-sfr`, et `yahoo` a `yimg`, quant à `wanadoo-Orange`, ils ont `woopic` (?!).

Réduire la taille des pages sécurisées

Un facteur que certaines personnes ont tendance à oublier : Faire un site au protocole https est compliqué et lourd. Un site https aura par ailleurs tendance à consommer plus de ressources serveur et client et plus de bande passante (à cause du chiffrement des données). Si vous envoyez des requêtes lourdes à cause des cookies, celles ci seront encore plus lourdes à cause du chiffrement.

Le https c'est le mal

Faire intervenir un changement de protocole au milieu d'un parcours peut être dramatique pour la performance de celui ci: tous les efforts faits dans les pages précédentes pour cacher le contenu sont réduits à néant car les ressources ont une adresse différente (<https://www.monsiteweb.com/ressource1.gif> au lieu de <http://www.monsiteweb.com/ressource1.gif>), car on demande au serveur et au navigateur de chiffrer et de déchiffrer toutes les ressources, car l'on chiffre toutes ses requêtes aussi (le chiffrement se fait dans les deux sens).

J'ai vécu l'expérience personnelle de changement de protocole sur des parcours, et le résultat était systématique : 100% des mesures sur les pages de changement de protocole étaient hors seuils (et pas qu'un peu).

Le https est indispensable

Je vous rappelle d'abord la loi : Tout responsable de traitement informatique de données personnelles doit adopter des mesures de sécurité physiques (sécurité des lo-

caux) et logiques (sécurité des systèmes d'information) adaptées à la nature des données et aux risques présentés par le traitement.

Le non-respect de l'obligation de sécurité est sanctionné de 5 ans d'emprisonnement et de 300 000 € d'amende. (art. 226-17 du code pénal)

Seules les personnes autorisées peuvent accéder aux données personnelles contenues dans un fichier.

La divulgation d'informations commise par imprudence ou négligence est punie de 3 ans d'emprisonnement et de 100 000 € d'amende. (art. 226-22 du code pénal).

Evidemment, la CNIL n'a pas les moyens de faire respecter ces lois et directives.

Utiliser le protocole https est donc indispensable:

- Le certificat que votre serveur vous authentifie et prouve votre identité. Quand on sait ce qui peut se passer sur internet, et qu'on connaît l'ingéniosité des escrocs, on se méfie.
- il y a des informations que votre visiteur vous envoie qui doivent rester secrètes: mot de passe, numéro de carte bleue...
- vous devez préserver l'identité de vos client et protéger leurs données. C'est une obligation légale qui si elle n'est pas respectée vous expose à des poursuites pénales. Donc dès que vous affichez des données personnelles (adresse, nom, points, facture...), vous devez chiffrer le trafic.

Pas question dans ses conditions de se passer du https. Il faut donc se poser la question de l'amointrissement de son impact et des raisons pour lesquelles il ralentit autant vos pages.

Gérer des ressources rares

En allant visiter les pages de tests avec une connexion sécurisée à <https://cyrilgodefroy.com/fr/testqos/>, on peut découvrir ou valider certains faits concernant l'usage du https.

Avant que vous ne vous jetiez sur l'url, sachez que je n'ai pas de certificat contresigné par une autorité de certification: si vous êtes sérieux, vous ne faites pas entièrement confiance aux requêtes que vous ferez. Rien de secret entre nous toutefois: vous pouvez accepter ce certificat, au moins pendant la session.

https = pas d'astuces d'amélioration

Le https fait échec à certaines mesures d'améliorations que nous avons évoquées:

- le cache est considéré comme non sécurisé sur le disque. Il a en effet pu être touché en dehors de la connaissance du navigateur. Donc le navigateur ne met pas en cache les ressources sécurisées sur le disque. Cela ne veut pas non plus dire qu'il recharge tout à la fois: il y a un cache de session en mémoire vive. Quand vous fermez le navigateur, tout disparaît.
- il est déconseillé d'utiliser différents sous domaines, voire différents domaines. Dieu merci, cette recommandation n'est pas bloquante: vous pouvez continuer à servir des images depuis un nom de domaine qui n'a pas de cookie.

- il y a peu de connexions sur internet explorer: pas de parallélisation du téléchargement des ressources. En tout cas, moins

Comment réduire l'impact des pages https

Tout d'abord, il faut que vous vérifiiez que votre plate forme https est à la hauteur : cartes accélératrices, choix du package de sécurité SSL, etc jouent énormément dans la performance pure de votre chargement d'éléments en protocole https. Comparez alors la vitesse de chargement d'une page en http et une page en https.

Ensuite, vous pouvez minimiser l'impact du passage du http au https grâce à plusieurs techniques et dans certaines situations.

Des frames? Oui encore un peu, merci..

La première technique, héritée des années 90, est d'utiliser des frames: mettez vos éléments de navigation dans une frame ou plusieurs frames, par exemple en haut et en bas de votre page, et servez ces pages en mode http. Limitez l'usage du https au contenu qui a besoin d'être protégé pendant le transfert et faites que toute la frame et toutes ses ressources soient en https.

Plusieurs inconvénients à cette technique. Tout d'abord, toute la page n'est pas sécurisée, et le navigateur le montrera avec quelques indicateurs, dont le cadenas (cassé).

ensuite, c'est des frames, avec tout ce que cela implique de difficultés, d'énervement et de travail inutile et bête.

In fine, ce n'est pas une bonne solution.

Faire des redirections

La technique suivante pour limiter les pages https est d'utiliser des redirections client après les formulaires fournis en https pour revenir en http : un formulaire d'authent par exemple n'a pas besoin d'être sur une page https, mais il faut que la requête POST quand on clique sur OK soit en https. Ensuite on peut faire une redirection client pour revenir sur du https. C'est par exemple ce que fait bytel.

Est ce que tout ce qui doit être sécurisé l'est bien avec cette technique? Vous seuls pouvez le dire... et la CNIL.

Les vraies techniques qui vont permettre de réellement améliorer votre affichage en https sont les mêmes que pour le http:

- peu d'éléments par page, notamment css et js
- des cookies de petite taille en allant chercher des ressources sur un domaine ou un sous domaine différents de celui qui crée les cookies.
- un cache et des directives efficaces et une utilisation intelligente de vos ressources. Le cache session n'est pas un vain mot. Exploitez le à fond.

Faites des pages hyper simples en https. Au mieux pour une page en https, il suffit d'une ressource : la page. Je suis sur que vous pouvez en avoir une qui s'affiche en moins de 1 seconde, https compris. Alors la prochaine fois qu'un designer vous fait une page d'espace client, de paiement, etc, désinstallez photoshop et illustrator de son ordinateur avant qu'il commence, ou fixez lui des règles très strictes en appelant à son intelligence: il n'est normalement pas là pour vous mettre des batons dans les roues.

Compresser pour aller plus vite

Comment fonctionne la compression?

Une des premières solutions pour rendre un site web plus rapide que les développeurs me citent, c'est cette solution, de compression du contenu. Je ne sais pas d'où ça vient, quelle est cette maladie... Non je rigole.

Le fonctionnement est le suivant : plutôt que de faire passer sur le réseau un fichier de plusieurs dizaines de kilo octets, le serveur web le compresse, et envoie une instruction au navigateur pour que celui ci sache que le fichier doit être décompressé. Le navigateur décompresse alors le fichier qui redevient tout à fait normal.

On peut faciliter la vie du serveur en compressant à l'avance les fichiers. La charge pour celui ci est alors réduite. Par exemple, le module `compres` de `lighttpd` conserve des versions compressées des fichiers qu'il envoie.

Le WebInspector de Safari note les ressources qui pourraient bénéficier de ce mode de compression avant transmission pour vous préconiser de les compresser.

Quels éléments compresser

Il n'est pas astucieux de compresser tous les éléments qui sont envoyés par le serveur. Je vous rappelle que la compression est une opération mathématiquement exigeante qui prend du temps processeur : compresser tous les éléments sur un serveur à fort trafic augmente donc encore la charge sur ce serveur.

Par ailleurs, les éléments qui sont déjà compressés ne gagnent rien à être encore compressés. Parfois même, le fichier compressé est légèrement plus gros que le fichier d'origine! Les fichiers qui sont déjà compressés (gif, jpeg, png, swf... notamment) ne doivent donc pas être compressés. Il faut limiter la compression aux ressources qui sont au format texte : html, css et javascript.

Tous les navigateurs ne sont pas égaux devant la compression http : il existe en effet deux modes de compression, et tous les navigateurs ne les implémentent pas, voire certains ont des bugs dans leur implémentation. Premier élément, le navigateur doit dire au serveur s'il accepte la compression dans les entêtes : il utilise l'instruction Accept-Encoding : gzip, deflate. Certaines versions de Netscape Navigator 4 disent supporter la méthode zlib alors qu'ils ne le font pas, etc : c'est une zone de configuration assez délicate, et qui souffre de nombreux écueils.

Par exemple, des fichiers PDF n peuvent pas être compressés car ce n'est pas le navigateur qui les exploite, mais directement le lecteur PDF, qui ne sait pas décompresser des fichiers.

Enfin, la plupart des navigateurs ne lisent les fichiers compressés que s'ils ont explicitement demandé à en avoir.

Comment compresser avec Apache

On peut compresser à plusieurs niveaux. On peut compresser par le serveur web, par le serveur d'application ou le langage de script.

Comme plus haut pour les directives de cache, je vais présenter la configuration qui peut être mise en place avec Apache 2, car c'est certainement le serveur web le plus utilisé.

Apache2 vient avec un module spécifique pour la compression, le module `mod_deflate`. Celui-ci est utilisé en appliquant la directive

`SetOutputFilter DEFLATE`

Ce module fonctionne ensuite par inclusion de certains types de fichiers, exclusion de fichiers, et peut aussi travailler avec des `BrowserMatch` (pour identifier des navigateurs un peu grincheux).

Voici une configuration typique:

```
<Location />
    SetOutputFilter DEFLATE
    DeflateCompressionLevel 9
    SetEnvIfNoCase Request_URI \
        \.(?:gif|jpe?g|png)$ no-gzip dont-vary
    SetEnvIfNoCase Request_URI \
        \.(?:exe|t?gz|zip|gz2|sit|rar)$ no-gzip dont-vary
</Location>
```

Toute cette compression peut avoir un impact sur la performance de votre serveur web : compresser peut en effet utiliser une part importante de vos ressources machine.

Compresser avec PHP

Php bénéficie d'un module de base pour compresser le contenu qui est renvoyé en html. Votre php doit être compilé avec le module `zlib` qui va bien. Vous pouvez utiliser la commande `phpinfo` pour avoir une page web vous indiquant tous les modules compilés dans votre php. Pour utiliser ce module, il suffit de l'appeler :

```
<?php
```

```
ob_start( 'ob_gzhandler' );  
?>
```

Comme il faut compresser toute la réponse et positionner cette instruction en premier, faites attention à positionner cette instruction en haut de page. Vraiment au premier caractère.

Compresser (minifier) les ressources javascript

Le javascript est rempli de vide et de contenus inutiles. Si, si. C'est un fait que tous les langages, avec les commentaires, les retours chariot, les espace etc, sont remplis de rien, mais de rien qui a du poids.

Or envoyer des contenus qui ont un poids trop important est inutile. On va donc essayer de réduire la taille des javascripts.

Il existe pour ce faire un programme souvent cité pour sa simplicité d'utilisation, l'efficacité du résultat obtenu (compression de 10 à 50% de la taille), et le fait que l'on peut toujours lire le code même une fois qu'il est minifié.

D'autres programmes peuvent être utilisés, comme celui qui vient avec dojo, ou d'autres qui brouillent le contenu du code. Leurs résultats pour le moment sont moins bons.

JSMIN, The JavaScript Minifier (<http://www.crockford.com/javascript/jsmin.html>)

Utiliser un CDN

CDN ques aquo?

Un CDN est un Content Delivery Network, un réseau de livraison de contenu, une entreprise qui a créé son propre réseau à l'intérieur d'internet pour tenter de rapprocher le contenu des utilisateurs. Ces entreprises sont apparues dans les années 90, et ne sont pas très connues du grand public, mais sont souvent utilisées par les gros groupes pour rendre possible et efficace la diffusion de gros contenus aux utilisateurs.

Le principe est simple : offrir un cache du contenu le plus proche possible de l'utilisateur final. La réalisation de cet objectif est moins simple, puisqu'il faut déterminer quel serveur est proche de l'internaute, comptabiliser la requête (pour être payé et connaître le trafic), rapatrier le bon fichier au bon moment (mieux vaut trop tôt que trop tard), gérer sa durée de vie etc.

Ces fonctionnalités font quand même qu'utiliser les services d'un réseau de distribution de contenu est généralement réservé aux sites à très fort trafic et ayant des ressources importantes.

Je connais de nom 3 réseaux : Akamai, Savvis et Limelight Networks. Akamai est le gros leader du marché, assurant 60% des contrats de CDN et représentant 25% du trafic internet à lui tout seul. Ce géant américain a déployé près d'une centaine de serveurs de cache chez les fournisseurs d'accès internet français et dans les points nodaux d'internet, les centres de peering, pour être capable de couvrir la majorité des internautes français. Il a en plus un autre niveau de serveurs de cache chez quelques gros fournisseurs, pour réduire encore le risque d'indisponibilité.

Toute la technologie et tous les serveurs de ce réseau servent uniquement à fournir le plus rapidement possible à vos utilisateurs tous les fichiers dont ils ont besoin. Si vous distribuez de petits fichiers et que vous êtes capables de réduire le nombre de fichiers nécessaires pour une page, vous n'aurez pas besoin d'un CDN.

Si par contre, vous êtes déjà gros consommateur de bande passante, que vous aimez mettre beaucoup d'images sur vos pages et faire de grosse pages, un CDN vous permettra de réduire énormément la latence réseau. Gare à l'embonpoint qu'il risque de provoquer, et à ne pas mettre des ressources plus nombreuses et plus importantes chaque jour, au point de vous retrouver définitivement tributaire de ce service.

Pour mettre en place un réseau de distribution de contenu, c'est en général assez simple : une modification DNS suffit. Par contre, pour exploiter toutes les possibilités, l'existence de plusieurs sous domaines et un minimum d'agilité s'avèrent très utiles. Par exemple, utiliser un sous domaine spécifique pour l'hébergement des pages sécurisées est une bonne idée : vous pouvez alors régler dans le DNS les destinations des requêtes et l'usage ou non du CDN pour les requêtes sécurisées (par essence pas cachables).

PARTIE III

Appliquer les meilleures pratiques sur son site

Comment analyser et améliorer son propre temps de chargement

Nous avons vu de manière générale les outils à disposition, les erreurs les plus fréquentes, les techniques pour améliorer le temps d'affichage, mais qu'est ce qui va vous faire gagner du temps à vous? Comment allez vous procéder?

Fixer des objectifs réalistes

Vous n'arriverez pas à afficher en 4 secondes une page contenant 300 images et faisant appel à beaucoup d'éléments Ajax ou simplement à du javascript un peu coton. Vous n'arriverez pas non plus à empiler toutes les fonctionnalités que vous désirez dans le design et le look and feel que vous voulez sans souffrir un peu sur le temps d'affichage (car après tout il faut beaucoup de ressources pour faire un joli site). Il faudra que vous fassiez la part des choses et que vous vous fixiez des objectifs réalistes

Il y a quelques années, on considérait qu'une page devait se charger (ou au moins commencer à apparaître) en 2 secondes. Ce précepte est toujours assez vrai. Je l'étendrais en disant qu'il faudrait qu'une page s'affiche en 5 secondes complètement. On peut augmenter ce seuil pour des pages qui deviennent compliquées, mais dans ce cas il faut les prévoir et mettre une temporisation rendant plus agréable ou plus perceptible le fait que le site fonctionne, et qu'il fait quelque chose de dur.

Connexion sécurisée

Si cette page reste affichée plus de 5 secondes, [cliquez ici pour l'actualiser](#).

PayPal. Votre mode de paiement et de réception de paiements en ligne rapide et sécurisé.



*Paypal cherche à se souvenir du solde à 9 chiffres de mon compte, cela prend du temps,
mais au moins je suis prévenu que ce sera long*

Ces objectifs doivent porter sur les pages d'accès à votre site web (home mais aussi pages d'accueil de catégories lors de promos ou de pubs), et sur les parcours complets qu'il y a derrière : c'est moins grave si votre page d'accueil met 6 secondes à se charger si cela vous permet derrière de fournir les autres pages en moins de 3 secondes.

Ces objectifs doivent surtout porter sur les pages d'accès aux services : accueil, accès au compte etc. Cela ne sert à rien d'avoir des pages derrière la page d'accueil qui se chargent en 3s si les utilisateurs sont partis sur la home car elle était trop lente.

Est ce que je viens de dire deux choses qui allaient dans des directions contraires?
C'est pour vous dire qu'il faudra revoir vos objectifs en fonction des résultats et des succès que vous aurez obtenus.

Le critère qui vous permettra de vous positionner par rapport au temps de chargement d'une page, c'est son poids total : si elle est très lourde (plus de 700Ko), aucun espoir de descendre sous les 4 secondes.

Rassembler les données des parcours les plus importants

Il faut que vous vous restreignez : vous ne pourrez pas obtenir rapidement des résultats magiques sur toutes les pages. Il faut donc travailler en premier lieu sur les parcours les plus importants. Plus importants parce qu'ils sont quotidiens pour vos utilisateurs, ou plus importants parce qu'ils font du business : ils sont critiques car il s'agit de l'achat, de l'inscription, etc.

Ensuite, rassemblez les données : non pas un parcours, pas 10, mais 50 ou cent. Vous ne pouvez pas baser vos analyses sur des tests unitaires. Construisez vos parcours en numérotant les pages, en faisant des captures des pages par lesquelles vous devez passer, de manière à toujours effectuer le même parcours. Au démarrage de chaque parcours, videz votre cache, effacez vos cookies et relancez votre navigateur pour vous mettre dans la peau de votre internaute.

Construisez un tableau de résultats avec le temps d'affichage complet de chaque page du parcours. Notez les infos au fur et à mesure. Pour déterminer le temps d'affichage d'une page, comptez à partir du moment où vous actionnez le lien, et ne vous fiez pas

aux chronos qu'on peut trouver avec les navigateurs : ceux ci peuvent se remettre à zéro au cours de l'affichage d'une page.

Prenez des traces de vos parcours, pour pouvoir les analyser a posteriori. L'outil que je vous conseille pour cela est HttpWatch : les traces conservées sont réutilisables telles quelles. Sauvegardez au fur et à mesure les traces de vos mesures de parcours en utilisant le numero du tests que vous venez d'effectuer.

Une fois que vous avez des traces conséquentes sur votre parcours, concentrez vous sur les pages les plus lentes à afficher en moyenne : faites une moyenne du temps d'affichage des pages d'un parcours, et prenez les plus lentes. Allez ensuite vérifier les traces http de ces pages. Http Watch ouvre les fichiers de type hwl, alors vous pouvez conserver toutes vos mesures sous ce format, qui est plus complet et peut éventuellement vous permettre d'aller voir aussi les requêtes, réponses et les headers.

Voici ensuite les choses que vous devez aller vérifier.

1. Repérer les pages de type text/html qui prennent du temps
2. Faites la chasse aux redirections
3. Comparez la taille des requêtes et leur nombre à la taille des réponses
4. Vérifiez qu'au cours d'un parcours les ressources sont servies par le cache
5. Page par page, vérifiez quelles ressources sont téléchargées en premier
6. Identifiez les pages qui doivent être refondues

Repérez les pages de type text/html qui prennent du temps

Certaines pages sont infailliblement lentes : elles font appel à un système distant, elles créent un rapport dans un format inhabituel pour le web, elles sont mal conçues... Ces pages doivent être débusquées dans les situations réelles (et pas seulement dans la phase de développement). Elles devraient déjà être connues du développeur ou de l'intégrateur de la partie applicative du site. Si il n'est pas capable de vous les citer, vous pouvez faire la liste pour lui. Tout ce qui dépasse une seconde en moyenne est suspicieux, tout ce qui dépasse les 3 secondes est carrément trop lent.

Comment agir face à ces pages? Suivant les justifications qui peuvent être faites, vous avez deux options :

si les systèmes derrière sont réellement complexes (système de réservation de transports aériens, catalogues de prix de plusieurs sites web...), vous devez traiter l'attente comme quelque chose d'inéluctable et prévenir l'internaute de celle ci. La capture de la page Paypal au début du chapitre est un bon exemple.

si la page est lente alors qu'elle ne devrait pas l'être d'après le développeur, renvoyez le vérifier son code et ses flux sur la production. Les flux doivent être de bout en bout : depuis l'entrée de la requête dans l'infrastructure d'hébergement jusqu'au renvoi de la réponse à la sortie de cette infrastructure en passant par les requêtes aux bases de données.

Faites la chasse aux redirections

Une façon assez nulle de perdre du temps que l'on rencontre trop souvent est la redirection. Toutes les requêtes ayant pour réponse des codes 300 et quelques (essentiel-

lement 302 et 301) sont des redirections. Ces redirections s'insèrent de purement séquentielles dans le chargement de vos pages : impossible d'accéder à la page finale sans passer par les pages de redirection et de subir le temps de réponse de ces pages.

L'existence de ces redirections n'est pas forcément innocent, motivé par la volonté de protéger l'application d'utilisateurs qui soumettraient plusieurs fois le même formulaire, ou de leur éviter une navigation en protocole https sécurisé inutile. Le problème est quand les redirections sont au nombre de deux ou plus successivement, ou quand elles pallient des mauvaises configurations.

Par exemple, on accède souvent aux url sans taper le / de fin pour indiquer l'accès à un répertoire. Le navigateur répond alors par une redirection. Or on peut le forcer à répondre directement sans effectuer la redirection. Evitez vous 400ms de round trip totalement inutile.

Comparez la taille des requêtes et des réponses

Dans HttpWatch, il y a un onglet 'summary' qui fait automatiquement le travail de somme du poids des requêtes et du poids des réponses. Vous pouvez aussi copier le contenu des colonnes dans un tableur, faire vous même la somme des colonnes requêtes et réponses. Cette somme vous permet d'avoir une première appréciation du trafic entre votre navigateur et le serveur dans les deux sens.

Si vos requêtes sont supérieures en poids aux réponses, il y a un problème évident (sauf dans le cas d'un upload). Si vos requêtes dépassent 10% de vos réponses, il faudra faire la chasse aux requêtes trop lourdes.

La vision synthétique ne doit pas vous empêcher d'aller faire le même travail dans le détail sur une page. Vous verrez assez vite les requêtes de statistiques et de publicité : ce sont souvent les plus lourdes. Vous verrez si vous risquez d'avoir un problème à cause de la taille des requêtes assez vite. Ne sous estimez pas l'influence de ce facteur : il n'est peut être pas évident sur votre connexion en cœur de réseau 100 Mb/s, il est plus évident sur une connexion ADSL à 1 Mb/s asymétrique disposant de 128 Kb/s sur le sens montant (du navigateur vers le serveur).

Vérifiez l'usage du cache

Cette recommandation suppose que vous avez mis en place des directives de cache, évidemment. Si vous n'en mettez pas, ne vous attendez pas à des miracles de ce côté. Le navigateur fera du best effort... mais celui ci peut être très limité. Par contre vous devez vérifier que vos directives de cache sur vos ressources statiques sont bien transmises, qu'elles sont appliquées et qu'elles sont efficaces.

L'efficacité se traduit dans HttpWatch par ce mot magique : (Cache) et par le chiffre qui va avec : 0. Il peut aussi se traduire par une ligne absente. Si vous avez une page avec votre logo mais que vous ne voyez pas de requête pour celui ci, tout va bien : il vient du cache.

Par contre si vous voyez ces codes : 200 ou 304, c'est que vous n'avez pas bien fait votre travail, juste à moitié. Pourquoi à moitié? Parce que le navigateur ne croit pas en la validité de la ressource dont il dispose. Il a donc besoin de revalider celle ci. Cela nous pose deux problèmes de performance : un parce que l'on refait un roundtrip (une boucle requête réponse), et que ce roundtrip peut être sur un fichier bloquant (type javascript). Deux parce que chaque requête pèse, bien entendu... Vous devez regarder dans le détail les requêtes et réponses sur la ressource en question :

- la durée du cache est elle bonne
- y a-t-il une directive du type must-revalidate
- y a-t-il un cookie envoyé dans la requête de la ressource

Quelles ressources sont téléchargées en premier?

L'ordre de téléchargement des ressources a une importance grande, on l'a déjà vu. Surtout les ressources de type javascript (mais pas seulement). Commencez par vérifier que les css sont chargés en premier. S'il y en a trop (plus de deux), vous pouvez noter de faire un progrès de ce point de vue.

Ensuite vérifiez l'usage du javascript. Le téléchargement d'un fichier externe javascript au milieu du corps de la page doit être proscrit. Le mieux serait de le charger à la fin (même si ce n'est pas toujours possible). En effet comme cela il ne bloque pas le chargement des autres ressources. Au pire, essayez de rassembler les fonctions dans un ou deux fichiers javascript, un pour les fonctions statiques et un deuxième pour les fonctions dynamiques, puis utilisez ces fonctions par l'usage de javascript inline, c'est à dire du javascript directement dans le html.

Vérifiez que vous n'avez pas d'aberrations dans le chargement : @import dans les fichiers css de style, des fichiers javascript et css intercalés les uns avec les autres.

Au fur et à mesure que vous vérifiez ces ressources texte statiques, vérifiez qu'elles sont minifiées (pour les javascripts) et passées en gzip ou en deflate (compression) pour toutes les ressources texte. Le WebInspector de Safari vous signale rapidement toutes les ressources qui pourraient bénéficier de compression mais n'en ont pas.

Identifiez les pages qui ont besoin d'être refondues

Certaines pages ne passeront jamais sous le seuil que vous leur avez fixé à la base : il est même impossible qu'elles le fassent en sortie de plateforme d'hébergement. Ces pages sont simplement trop lourdes, trop compliquées en nombre d'éléments à charger, trop compliquées en structure (tableau dans un tableau dans un tableau ...).

Ne vous éreintez pas avec ces pages. Sachez les identifier rapidement pour y revenir plus tard. Elles vont juste réussir à épuiser vos ressources, votre patience et votre ingéniosité, et vous n'arriverez pas à un bon résultat. Dans ce cas, vous êtes perdant. Concentrez vous d'abord sur les pages les plus faciles à corriger. Vous y gagnerez plus avec celles ci en gratification immédiate.

Les pages les plus dures, n'hésitez pas à les remettre sur la planche à dessin : planche à dessin graphique, html, applicative, ergonomique. Changez leur seuil pour fixer un objectif réaliste : une page de 900 Ko pour 100 éléments mettra fatalement plus de 5 secondes à se charger sur une connexion à 1 Mb/s ($800/(1024/8) = 7$ s au mieux pour un fichier unique).

Une home qui fait snap!

Votre page d'accueil est certainement le premier contact que font les internautes avec votre site. Or on n'a qu'une seule chance de faire bonne impression : il n'existe pas de deuxième première rencontre. Votre premier objectif de superperformance est donc d'avoir une home qui fait snap!

J'ai rencontré plusieurs personnes qui ne voyaient la home que comme la première page d'un parcours. A ce titre ils pensaient qu'elle devait avoir la même apparence et la même construction que les autres pages, notamment pour "mettre dans le cache les éléments à réutiliser sur les autres pages".

Ce genre d'attitude est dommage à mon sens : 20% du trafic du site en question se fait sur la page d'accueil, et cette tendance doit s'accroître puisque cette page d'accueil est devenue la page d'accueil d'un fournisseur d'accès internet, un portail vers de nombreux contenus.

La page d'accueil de votre site est la plus importante et la plus spéciale. Elle doit être traitée comme un joyau unique et mérite tous vos efforts. Souvent elle sera la seule page visitée de votre site. Même s'il repart, l'internaute doit rester sur une bonne impression. Alors ne la traitez pas comme les autres pages.

Encore une fois, inspirez vous des meilleurs. Regardez la home de Google, 2 fichiers et 94 ms. Moins de temps qu'il ne faut pour se rendre compte qu'on a appuyé sur entrée. Vous allez me ressortir: oui mais c'est un moteur de recherche, c'est pas pareil. Si. Un fichier html et une image: c'est tout ce qu'il faut charger.

Regardez la home de Yahoo, si vous n'êtes pas convaincu. Ne me dites pas qu'elle est comme celle de Google: des images, des blocs, des photos, de la pub. De la pub avec de la vidéo! Si vous trouvez quelque chose de moins performant en web 1.0, dites le moi. Les mêmes règles de construction sont appliquées.

Tout envoyer en une requête

On l'a vu tout au long des pages précédentes: demander des fichiers à charger, c'est bien trop long, et il faut faire son possible pour éviter toute requête supplémentaire.

On peut envoyer moins de choses, alléger les choses, ou tout envoyer en une fois. C'est ce qu'on essaie de faire avec le pipelining: envoyer plus de contenus dans la même requête.

Pour notre page d'accueil, nous allons donc rassembler tous les fichiers qui peuvent l'être: html, css et javascript. Plusieurs images deviendront une seule image.

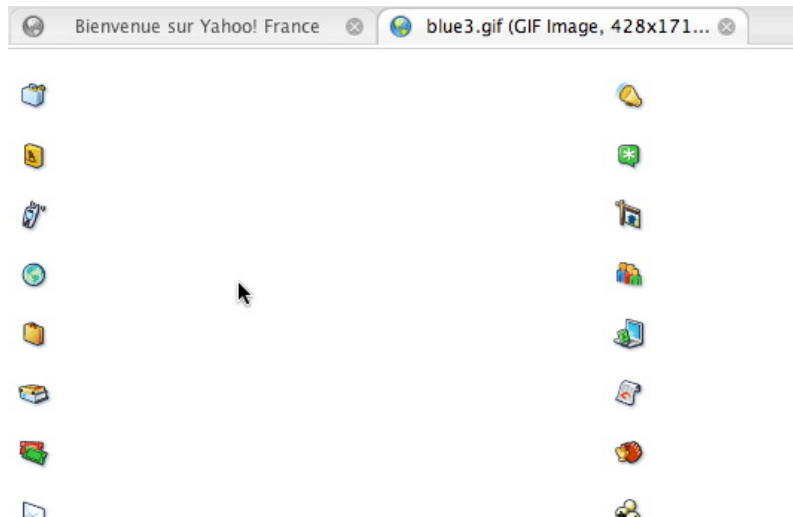
Prenez votre éditeur de texte, ouvrez votre home et votre feuille destyle, votre javascript (vous ne devriez déjà plus en avoir qu'un). Puis intégrez ces deux fichiers en interne :

Vous venez de gagner deux requêtes, séquentielles qui plus est.

Ensuite, si votre design et votre animation éditoriale le permettent, agrégez plusieurs images en une. Par exemple, les catégories du menu de gauche de la page d'accueil de Yahoo! sont une seule image.

Vous n'êtes pas obligés de faire une grosse image map, vous pouvez aussi utiliser des sprites css (<http://www.alistapart.com/articles/sprites>). Cette technique vous permet

d'utiliser une grosse image contenant de nombreux éléments à afficher , et d'afficher juste une partie de cette image, un carré à l'intérieur de celle ci.



Les icônes de Yahoo sont toutes dans un seul fichier image ou presque..

Un exemple des résultats en appliquant une partie des recommandations, et sans faire de nettoyage dans les scripts ou les styles:

<http://cyrilgodefroy.com/fr/testqos/snap/avant.html>

<http://cyrilgodefroy.com/fr/testqos/snap/apres.html>

Et le web 2.0 dans tout ça?

On n'a pas beaucoup évoqué les applications web 2.0, utilisant à foison les interfaces riches construites grâce aux scripts javascripts comme prototype, JSON, scriptaculous, DOJO, etc. Et pour cause : ces applications sont plutôt nouvelles et peu de gens réalisent aujourd'hui de telles choses. C'est le travail de spécialiste à l'intérieur du travail de spécialiste.

Il ne faut pas trop s'inquiéter de ces applications : les règles qui prévalent à l'existence de pages très performantes sont les mêmes pour les applications web 2.0. (et 3.0) : le javascript se chargera toujours séquentiellement, les css aussi sur Firefox, les requêtes sont aussi lourdes. En étant rigoureux comme pour une application normale, vous obtiendrez des temps d'affichage équivalents et aussi bons.

Deux différences méritent notre attention : le principe d'une application riche embarquée dans le navigateur, utilisant une seule page pour fonctionner fait disparaître la notion de temps d'affichage des pages. Quand j'arrive sur ma page docs.google.com, mon webmail yahoo ou netvibes, j'ai peut être un temps d'affichage un peu plus important, mais je suis prévenu et je n'ai ce temps de chargement qu'une fois. Le reste du temps, toutes mes requêtes à l'application ont un temps d'exécution assez rapide, sans doute beaucoup plus rapide que si j'étais passé par un cycle requête/réponse complet.

L'autre différence concerne le XMLHttpRequest : ce drôle d'objet qui sert à faire de l'Ajx (du chargement asynchrone de pages web) ne se comporte pas tout à fait de la même façon que le navigateur de base.

J'ai récemment eu à auditer la performance d'une page qui faisait du web 2.0 : un multi contenu agrégeant des flux extérieurs et un menu de contenus configurables. L'utilisation de js et de requêtes asynchrone était assez colossal : sur la même page on passait de 65 requêtes à 115 requêtes en ajoutant ces deux éléments. La performance s'est bien sûr écroulé. On est passé d'une page qui s'affichait en moins de 3 secondes à une page qui s'affichait en plus de 7 secondes.

Pourquoi tant de décalage? A cause des suspects habituels d'abord : trop de javascripts, insérés en cœur de page plutôt que d'appeler leurs fonctions en inline, des requêtes à tire larigot (une par contenu caché), qui chacune était courte, efficace, mais qui agglomérées donnaient une performance désastreuse.

J'ai fait mes recommandations. N'étant pas maître du développement il m'a fallu composer avec une équipe de développement réticente à refaire le travail (difficile) qu'elle avait déjà fait, mais sans prendre en compte des objectifs de performance d'affichage.

J'ai aussi eu des expériences désagréables avec Dojo : leur système (malin) de localisation fait que les fichiers sont en plusieurs bouts. On télécharge donc beaucoup de fichiers javascripts. C'est un problème en soi. Je conseille donc de se méfier de Dojo, voire de faire sa propre version mono fichier.

Si vous faites de l'Ajx, pensez à bien maîtriser vos fichiers javascript, à les versionner et à leur mettre des durées de vie dans les caches importantes, minifiez les et compressez les : c'est dommage de passer deux secondes ou trois secondes à charger du javascript sur une page d'accueil

Conclusion

Le sujet de la performance des pages web est un sujet riche, qui se caractérise par l'excellence dans tous les domaines de la création web : excellence ergonomique, applicative, html et bien sûr http. C'est un travail de spécialiste méticuleux et précis, ayant une culture diverse et profonde de la chose web : cache, https, javascripts font partie de domaines différents, un profil rare sinon inexistant. Ce travail est d'autant plus intéressant qu'il fait donc intervenir différentes spécialités, et permet de remettre tous les acteurs de la création de site web autour de la table, ou de la page : vous ne réussirez à mettre de la performance dans vos pages que si vous communiquez auprès de tous les intervenants et que si vous avez le courage de refaire des choses qui pourtant marchent 'bien'.

Dans ce contexte, une approche agile du développement de sites et d'applications web sera préférable et vous permettra d'atteindre vos objectifs. Une approche compartimentée est le meilleur moyen de voir les intervenants se renvoyer la balle et se défausser mutuellement l'un sur l'autre.

C'est aussi un sujet d'expertise assez récent. Il est donc en évolution. Je vais faire l'effort évident de continuer à travailler sur les causes de lenteurs de sites web, et à analyser le comportement des navigateurs par rapport à plusieurs comportements. Je mettrai à disposition les mises à jour des informations disponibles dans ce livre au fur et à mesure.

Les sites à tester

Top sites of France Alexa

Vous pouvez aussi trouver une liste des 30 groupes faisant le plus de trafic sur Le Journal Du Net

Les articles de référence

Des articles sans intérêt écrits par des gens qui ne savent pas de quoi ils parlent

Yahoo Performance Research La task force performance de Yahoo
(<http://yuiblog.com/blog/2006/11/28/performance-research-part-1/>).

Serving Javascript fast Un développeur de Flickr
(<http://www.thinkvitamin.com/features/webapps/serving-javascript-fast>)

Mnot's web log Mark Nottingham est gourou chez Yahoo, il a beaucoup travaillé chez Akamai, 2 tit' boites (<http://www.mnot.net/blog/Caching/index.html>)

<http://www.websiteoptimization.com/> Des gens qui vendent aussi de l'optimisation de site web

http://www.die.net/musings/page_load_time/ Un ingénieur qui travaille sur les applications Web 2.0 d'une start up : Google.

14 rules for fast web pages Encore un gars de chez Yahoo. Il faut croire que cette entreprise se pose des questions sur la performance

(http://www.skrenta.com/2007/05/14_rules_for_fast_web_pages_by_1.html).

<http://code.google.com/p/minify/> Une librairie pour éviter de faire des erreurs avec ses css et ses javascripts en php.

<http://www.webperformance.org/compression/gzip-compress.html> pour savoir un peu mieux comment utiliser le mode de compression de Apache, et bien sûr, mod_deflate - Apache HTTP Server (http://httpd.apache.org/docs/2.0/mod/mod_deflate.html)

Yahoo Performance Group un groupe de personnes bien intentionnées

(<http://developer.yahoo.com/performance/>).

Flash, attention au cache

Par cyril, mardi 23 octobre 2007 à 14:17 [Maximum Performance #87](#) [rss](#)

les fichiers flash sont une plaie pour le poids des pages, mais ça vous le saviez déjà. Ils sont lourds, difficilement modifiables, jamais remplaçables. Mais ils plaisent au marketing et aident à vendre plein de mobiles, de lecteurs mp3, des portefeuilles ou des montres sur mesure où vous gravez votre nom. Difficile de les refuser...

Le plus gênant est qu'ils sont faits par des gens qui n'ont aucune considération pour le cache, et je sais de quoi je parle (vous voulez voir toutes mes réalisations en Flash?). Notamment, une méthode très utilisée pour passer des paramètres à un flash est de mettre comme url de source le nom de fichier suivi d'un point d'interrogation puis tous les paramètres. Attention danger! Ce n'est pas cachable.

C'est même à l'origine le meilleur moyen utilisé par les développeurs pour être sûrs de charger le flash ou une autre ressource sans cache : forcer un paramètre variant à chaque requête. Quelque chose comme `compteur.swf?now=2543985`.

Vous pouvez mettre une directive de cache sur le fichier swf, cela ne changera malheureusement pas le comportement du navigateur au premier chargement : il téléchargera le fichier flash plusieurs fois comme s'il était différent à chaque fois.

Il existe une solution pourtant bien documentée qui permet d'éviter ce comportement et de pousser le flash dans le cache. Quand vous passez du stade développement au stade recette ou production, pensez à utiliser les flashvars. C'est l'autre moyen de passer des paramètres au lecteur Flash, sans utiliser l'url. C'est tellement [bien documenté](#) que des objets comme [swfobject](#), la librairie la plus utilisée pour intégrer les flash a des méthodes pour faire ce passage de paramètres.

Maintenant, si vous utilisez un énorme fichier ou que vous avez besoin d'informations toujours garanties fraîches, vous devrez utiliser d'autres méthodes pour améliorer le temps de chargement. Je vous conseille le réseau de distribution de contenu.

La vérité sur les ETags

Par cyril, dimanche 30 septembre 2007 à 22:53 [Maximum Performance #82 rss](#)

Un des critères de notation utilisés par l'excellent [YSlow](#) pour apprécier la qualité de votre site est l'utilisation des ETags. D'ailleurs, si on consulte [les commentaires](#) laissés à ce sujet, on voit l'équipe de Yahoo obligée de se justifier.

Je vais essayer de traduire le consensus sur la question et de vous expliquer en termes simples ce que sont ces ETags.

1. Qu'est ce qu'un ETag, à quoi ça sert?

Un ETag est un code renseigné par le serveur web sur une ressource, et qui est fourni avec l'en tête de la ressource. Cela ressemble à ça:

"1ed39b-9a6-45ce4f9c", ou à "-346776727645". Ce code identifie 'de manière unique' la ressource dans le temps et dans son chemin.

C'est donc là pour valider l'unicité d'une ressource ou d'une représentation.

2. Comment sont fabriqués les ETags

Les ETags sont créés par le serveur qui renvoie la ressource. Celui ci utilise différents critères pour créer l'ETag.

Apache utilise par défaut la combinaison de trois éléments pour créer l'ETag : l'inode du répertoire où est stocké la ressource, le timestamp unix (une mesure des millisecondes) et la taille du fichier.

IIS utilise une autre façon de générer les ETags. Le format par défaut est Filetimestamp:Changenummer où Changenummer est un compteur dans le serveur.

Mon serveur lighttpd une autre encore. C'est toutefois encore une combinaison de node-value (emplacement sur le disque), un timestamp et la taille

3. Où sont les ETags?

Les ETags sont dans les entêtes des ressources qui sont envoyées par les serveurs http. Lancez votre firebug et regardez une ressource statique sur le site du meilleur opérateur de la planète (qui n'a pas l'iphone toutefois). Et regardez l'entête d'une image. Vous y verrez quelque chose comme "df0fd-1ca1-44fc2b84".

4. Quand sont ils utilisés?

Quand le cache veut utiliser un objet qu'il détient déjà, et qu'il veut l'utiliser comme réponse à la requête d'un client, il doit d'abord vérifier avec la source (ou avec un cache intermédiaire qui a une donnée plus fraîche) pour voir si sa ressource mise en cache est utilisable (HTTP 1.1 spec 13.3).

Ce cas de figure peut se produire de plusieurs façons:

- vous n'avez pas mis d'informations de cache, comme max-age:36000, si bien que votre cache n'a pas beaucoup d'informations pour juger de la fraîcheur de ses ressources. Votre ressource n'est pas fraîche, le cache doit s'assurer de la fraîcheur avant de vous la donner.
- la date de validité calculée de votre ressource est dépassée. Votre agent doit alors savoir s'il dispose de la représentation la plus fraîche de la ressource.

Bien sûr, quand je parle de cache, je parle à la fois de votre navigateur et d'un cache intermédiaire (rproxy, proxy, cdn).

5. Est ce que ça marche?

Pour la plupart des sites, cela fonctionne PARFAITEMENT dans la distribution et les réglages de base. En effet pour la plupart des sites, quand un cache a besoin de valider la fraîcheur d'une ressource en envoyant une ressource de type If-None-Match, le serveur renvoie une réponse de type 304 : Not Modified.

Le but des ETags est alors rempli : au lieu de faire un aller-retour avec une réponse complète contenant toute la ressource, on a une réponse ne contenant que quelques octets : la transaction est normalement beaucoup plus rapide.

Pour les sites plus gros qui ont besoin de plusieurs serveurs web pour servir leurs fichiers statiques, cela peut devenir plus délicat à utiliser. En effet dans ce cas, il est probable que l'ETag généré pour une ressource (pourtant la même) soit différent suivant le serveur qui la génère, ne serait ce que parce que le inode est différent d'un serveur à l'autre.

Oui, c'est un problème de riche.

6. Comment éviter d'avoir des problèmes à cause de ses ETags

Première solution : n'utiliser aucun cache. Oui, cela va à l'encontre de tous les principes, mais si on ne met pas de directive de cache et qu'on envoie pas d'ETag, on ne risque pas d'avoir le problème évoqué plus tôt.

Deuxième solution : faites en sorte que vos ETags ne soient jamais nécessaires, avec des ressources qui ont des durées de mise en cache très importante. Pas très malin non plus, et difficile à appliquer à tous les fichiers.

Troisième solution : ne pas utiliser les ETags. Il existe un autre mécanisme de validation (la validation légère) qui repose sur l'entête 'Last-Modified'. Bien que cette validation soit moins précise, elle conviendra dans 99,99% des cas. Bien sûr, il faudra que cette information soit la même sur vos différents serveurs.

Quatrième solution : faites en sorte que vos ETags soient les mêmes sur tous vos serveurs. Pour cela, rabattez vous sur les deux arguments qui peuvent être les mêmes d'un serveur à l'autre : la taille et le timestamp de dernière modification.

7. Les cas limites

Les fichiers gzippés sont différents à chaque fois qu'ils sont régénérés : l'ETag est différent d'une requête à l'autre. Dans ces conditions, l'ETag n'a servi à rien car on ne peut jamais valider (même si la ressource qui est gzippée n'a pas changé).

Les 10 règles d'or des pages les plus rapides

Par cyril, mardi 31 juillet 2007 à 20:39 [Maximum Performance](#)

.(.....) a été très riche sur la question de la performance et j'ai pu théoriser certaines connaissances ou idées que j'avais sur la performance. J'ai développé mes connaissances avec des tests, des expériences, la recherche de dizaines d'articles, l'usage de plusieurs outils.

J'ai obtenu des super résultats, j'ai connu des échecs cuisants.

J'en ai tiré une série de règles à respecter pour améliorer de manière fulgurante l'affichage des pages web :

1. Faire la chasse aux redirections
2. Concentrer les css et les javascripts en un minimum de fichiers
3. Faire confiance au cache
4. Compresser les contenus textuels
5. Surveiller les cookies
6. Ne pas rompre le parcours avec du https
7. Utiliser plusieurs sous domaines
8. Surveiller ses doctypes et sa syntaxe
9. Réduire le nombre d'éléments par page
10. Rendre les pages applicatives plus rapides

Ces 10 règles sont accompagnées de la règle d'or: 80% à 90% du temps d'affichage est passé dans le chargement des éléments autre que html, c'est là dessus que l'on gagne le plus et le plus facilement de la performance.