WHEN THE SOLUTION MATTERS

Optimisation de serveurs web

Christian Quest

cquest@cquest.org

Optimiser: pourquoi ?

- Améliorer la qualité de service:
  - Réduire les temps de réponses
  - Réduire les temps de transfert et d'affichage
- Réduire les besoins et les coûts
  - Réduire les besoins en bande passante (coûteuse)
  - Réduire les besoins en puissance machine

Optimiser: quoi ?

- Optimiser le volume de données à transférer
  - Economies sur la bande passante nécessaire
  - Meilleurs temps de réponse
  - Temps de transfert et d'affichage réduits
- Optimiser le temps machine
  - Plus de temps pour faire d'autres choses
  - Meilleurs temps de réponse

Optimiser: où ?

- Optimisations au niveau protocole
  - le protocole HTTP (1.0 et 1.1)
  - utilisation de keep-alive
  - utilisation du "byte-range-serving"
  - transmission compressée
  - bonne utilisation des caches
- Optimisation du contenu
  - optimisation du code HTML
  - optimisation des images

Optimiser: comment ?

- Réduire le nombre de requêtes
  - Meilleure utilisation des caches
  - Alléger le design de certaines pages
  - Réutiliser au mieux des éléments statiques
- Réduire le nombre de connexions TCP
  - Mieux utiliser les connexions
- Réduire le traffic
  - Réduire le volume de données à transférer

Volume utile et volume réel

- Principe de la transmission par paquets
- Encapsulation à chaque niveau
  - HTTP (requêtes et réponses HTTP, plusieurs centaines d'octets)
  - TCP (ouverture de connexion + entête du protocole TCP = 24 octets/paquet + retransmissions)
  - IP (entêtes du protocole IP = 24 octets/paquet)

Le protocole HTTP et ses défauts

- Une session TCP par requête
  - Beaucoup de temps machine utilisé uniquement pour crée/supprimer les connexions TCP
  - Problème du «slow-start» de TCP
- Solution courante: un client ouvre plusieurs connexions
  - Inconvénient: il faut traiter un grand nombre de connexions

HTTP et HTTPS

- HTTPS = HTTP sur SSL: une couche de plus!
  - HTTP > SSL > TCP > IP
- SSL rajoute encore plus de charge lors des ouvertures de connexions
  - Négociation des clés cryptographiques
  - Transmission des certificats
- La réduction des connexions est encore plus importante qu'en HTTP "classique"

HTTP 1.0

- Utilisé encore par une majorité de navigateurs et de serveurs
  - Il est "universel"
- Une connexion par requête
  - Peu efficace, gourmand en temps machine et en bande passante
- Quelques problèmes avec les proxies

HTTP 1.0 + Keep-Alive

- Le "keep-alive" est reconnu par une majorité de navigateurs
- Fin du principe "une requête = une connexion"
  - Moins gourmand en bande passante et temps machine
  - Facile à implémenter

Le principe du "keep-alive"

- Conserver la connexion ouverte après le traitement d’une requête
- Ceci n'est possible que si l'on connaît la taille des données de la réponse (difficile pour des pages dynamiques)

Byte-Range-Serving

- Sert à ne transmettre que des parties d'un fichier
- Utilisé par exemple par le plugin PDF
  - Permet de ne charger que les parties d'un fichier PDF nécessaire à la lecture d'une page
- Peut servir à l'affichage plus rapide de la mise en page (dimensions des images)

HTTP 1.1

- Fonctionne par défaut en "keep-alive"
- Meilleur support des proxies et caches
- Bien plus complexe à implémenter en totalité qu'HTTP 1.0
- Pas supporté par tout les navigateurs

Conseils d'implémentation

- Permettre aux caches de fonctionner
  - Fournir des dates de modification et/ou des dates d'expiration
  - Limiter les pages générées dynamiquement
- Supporter le "keep-alive"
- Supporter le "byte-range-serving"

Optimisations du contenu

- But: transférer moins de données
- Optimisation au niveau design
  - Une grosse image ou plusieurs petites ?
  - Limiter les styles, préférer les feuilles de style

Optimisation au niveau HTML

- L'HTML généré par les composeurs n'est généralement pas optimisé
  - Indentation inutile
  - Balises HTML redondantes
  - Balises HTML en majuscule

Compression du contenu

- HTTP supporte la compression depuis la version 1.0 !
- Supporté par la majorité les navigateurs
- L'HTML se comprime très bien
- algorithme gzip (RFC#1950, 1951,1952) très efficace (plus que celui des modems)
- ITK_Mac2gzip, ITK_Blobgzip...

Compression du contenu

- Réduit très nettement le volume (typiquement 50% à 75%)
- Réduit le nombre de paquets IP transmis
- Réduit le temps de transmission
- Le temps de décompression est négligeable

Ajout de la compression

- Vérifier que le navigateur supporte la compression (Accept-Encoding)
- Ne compresser que les données qui ne le sont pas déjà (HTML, Texte)
- Générer du code HTML "propre" et plus compressible (balises en minuscules)
- Conserver un cache de compression: éviter de recompresser en permanence les mêmes fichiers...

Un peu de code !

- Keep-Alive
- Byte-Range serving
- Compression

- La base d'exemple ITK_ACME_Demo

Questions / Réponses

A vous la parole !

www.internet-toolkit.com