La distribution de contenu dans l'Internet
2 – Usage des proxies dans le Web

Christophe Deleuze

Christophe.Deleuze@free.fr

ENST Paris

janvier 2004

Proxy

élément intermédiaire dans une communication client-serveur

- passerelle
  - ✓ protocoles
  - ✓ réseau public/privé
  - ✓ paramètres réseau
- filtrage
- transformation de contenu
- cache

autre élément intermédiaire

- tunnel

Importance dans le web

web : système centralisé ➠ peu *scalable*

- indispensable pour la performance
- groupes de travail IETF
  - ✓ WREC *(web replication and caching)*
  - ✓ WEBI *(web intermediaries)*
- mais complexe à gérer
  - ✓ spec HTTP ➠ 35 pages sur 176 (20/114)
  - ✓ éliminer le besoin de requêtes : expiration
  - ✓ éliminer le besoin de réponses complètes : validation
  - ✓ mais préserver la "transparence sémantique"

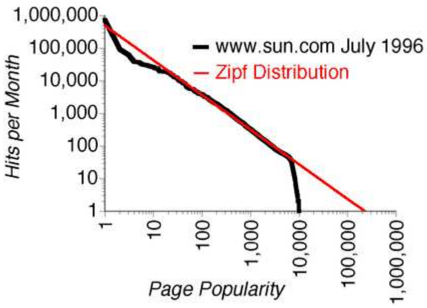
support HTTP

- en têtes
  - ✓ *end-to-end* cachés
  - ✓ *hop-by-hop* liste fixée plus ceux dans `Connection`
- CONNECT transforme un proxy en tunnel
- Authentification
  - ✓ ne cachent pas sauf si explicitement autorisé
  - ✓ ont leur propre auth

```
⇐ 407 Proxy Authentication Required
  Proxy-Authenticate: ...
⇒ Proxy-Authorization: ...
```

- négociation transparente

Pourquoi le cache est utile

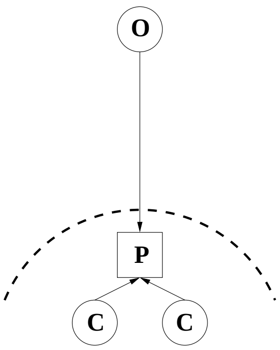


- 1 cache = 10–100 go, Internet = 300 tera-octets (10^{12})

Scénarios

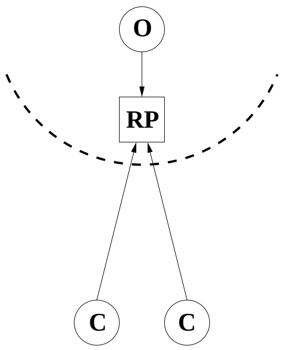
Forward Proxy Caching

- un *proxy* sert le contenu au client
- explicite/forcé/transparent
- géré par le FAI
  - ✓ + débit
  - ✓ + délai
  - ✓ + charge origine
  - ✓ – stats origine
  - ✓ – fraicheur



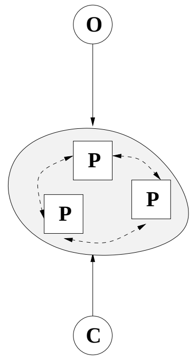
Accélérateur web (reverse proxy)

- origine caché derrière un proxy
- géré par le fournisseur de contenu
  - ✓ + charge origine
- variante : *web switch/server farm*



Cluster

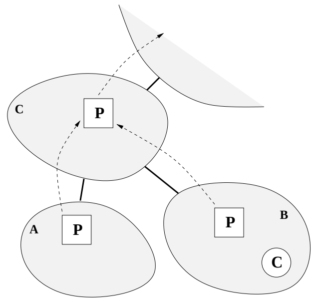
- groupe de caches localisé
- eqv système plus puissant



Hiérarchie

projet Harvest '94

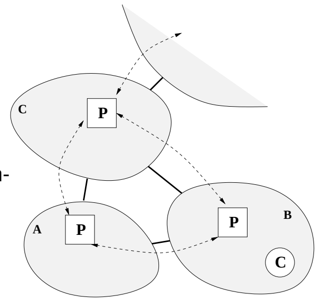
- récursion du principe
- hiérarchie ➠ scalabilité
- conf. statique
- topologie du réseau



Maillage

- *Cache mesh*
- organisation hybride

➠ topologie des flots de données



Maintien de la cohérence

Problème général

- architecture
  - ✓ plusieurs écrivains
  - ✓ fortes contraintes sur CPU et mémoire
- bases de données distribuées
  - ✓ garanties de transaction
- systèmes de fichiers distribués
  - ✓ similaire mais web $10^n \times$ plus gros + hétérogène

Types de cohérence

- forte (bémol)
- delta
- faible
- explicite
- mutuelle
- $\neq$ besoins pour $\neq$ objets et $\neq$ utilisateurs
- côté client (proxy)/serveur

Expiration

- date spécifiée par le serveur : `Expires`
- heuristique "adaptive TTL"
  - ✓ ex. Cisco Content Engine
    TTL = (CurrentDate - LastModTime) $\times$ FreshnessFactor
    (spec décourage FF > 10 %)
- expiration si age > TTL
  - ✓ origine fournit `Date`
  - ✓ proxy fournit `Age`

Validation

- origine associe un "validateur" à la réponse
- req. cond. avec validateur
  - ✓ si OK ➠ `301 Not Modified`
- validateurs
  - ✓ `Last-Modified` (`If-Modified-Since`) HTTP/1.0
  - ✓ `ETag` ➠ validateur "opaque"
- validateurs faibles et forts
  - ✓ faible : reste valide si chgt "not semantically significant"
    - ☞ `GET subrange` interdit
  - ✓ fort : invalide pour tout chgt
- subtilités avec `Last-Modified`
- *browser* reload/shift-reload

Cachabilité

HTTP/1.0 `Pragma: no-cache`
HTTP/1.1 `Cache-Control:`

requêtes	réponses
`no-cache`	`public`
`no-store`	`private`
`max-age`	`no-cache`
`max-stale`	`no-store`
`min-fresh`	`no-transform`
`no-transform`	`must-revalidate`
`only-if-cached`	`proxy-revalidate`
`cache-extension`	`max-age`
	`s-maxage`
	`cache-extension`

Cohérence : résumé

support pour :
- explicite (`Expires`)
- forte non scalable (revalidation systématique)
- faible OK (adaptive TTL)
- hétérogénéité si explicite
- hétérogénité client

manque : invalidation/mise à jour
- délicat à intégrer dans HTTP
- état serveur
- fiabilité/scalabilité
- pb administratifs

Protocoles inter-caches

Internet Cache Protocol - ICP - 1

- '95 cache *Squid*
- conçu pour HTTP/0.9
- défini dans rfc2186/7
- relations
  - ✓ parents
  - ✓ frères *(siblings)*
- msg UDP : 20 octets en-tête + URL

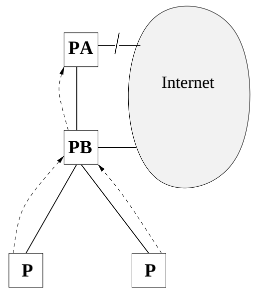
ICP - 2

1. envoie
   - `ICP_QUERY` aux pairs
   - `ICP_SECHO` à l'origine
   - `ICP_DECHO` aux pairs non ICP
2. reçoit `ICP_HIT` ou `ICP_MISS`
3. - 1er `HIT` gagne
   - attend max 2 secondes
   - pas de hit : parent le plus proche

cas des pairs "morts"

ICP - 3

- `ICP_MISS_NOFETCH`
- `ICP_HIT_OBJ`
  - ✓ + rapide
  - ✓ –
    - ☞ + lent qu'un `HIT`
    - ☞ fragmentation
    - ☞ infos en-tête ?
    - ☞ *cache poisoning*
  - ✓ ➟ déconseillé



ICP - 4

parents ou frères ?
- parents mutuels pour contenu dans A et B
- frères pour contenu ailleurs
- ...

multicast ?
- ne réduit pas le nb de réponses
- pb sécurité

ICP - 5

network probe
- cache construit une table (origine, RTT)
- inclut son RTT vers l'origine dans `ICP_REPLY`
- sélection du cache le plus proche (peut être soit-même) !
  - ✓ plus complexe
  - ✓ attendre toutes les réponses

problèmes de ICP
- scalabilité
- sécurité
- en têtes HTTP/1.1 (mauvaise prédiction)

Hyper Text Caching Protocol - HTCP

rfc2756 HTCP/0.0 - expérimental

- msgs sur UDP (TCP optionnel)
- en-têtes complètes
- envoi d'infos générales
- authentification (signatures)
- opérations
  - ✓ `TST` objet présent ?
  - ✓ `MON` "tiens-moi au courant"
  - ✓ `SET`
  - ✓ `CLR` oublier entité

Adaptive Web Caching

- gestion des *hot-spots*
- groupes dynamiques de caches (se chevauchant)
- adaptabilité, auto-organisation
- Cache Group Management Protocol (CGMP) : vote + retour
- Content Routing Protocol (CRP)
  - ✓ *URL routing table*
  - ✓ cache hier. par contenu
- communications multicast
- pb : frontières administratives

Cache Digests

Les protocoles précédents ajoutent des délais

- *Cache digest* : résumé du contenu d'un cache
  - ✓ basé les *Bloom filters*
  - ✓ compression avec pertes et recherche (hash)
  - ✓ impossible effacer une clé unique
- les caches pairs s'échangent leur CD
- màj périodique
- 1 Mo pour 16 Go

Système similaire : *summary cache*

CARP

Cache Array Routing Protocol

- spec. draft '98, Microsoft
- répartition pondérée des URL entre les caches
  - ✓ URL $\Rightarrow$ (C1, C2, ...)
- pour proxy ou client
- ...

WCCP

Web Cache Coordination/Communication Protocol
propriétaire CISCO, spec. publique en '00

- router $\Leftrightarrow$ caches
- caches transparents
- hiérarchie
- *cluster* de caches (hachage sur IP dest)
- *monitoring*
- *plug and play*
  - ✓ démarrage froid

WCCP

- partage de *clusters*
- routers redondants
- *overload bypass*
- *dynamic client bypass*
- *reverse proxy caching* (hachage IP source + port)

Approches avec répertoires

- "Distributed Internet Cache"
  - ✓ hiérarchie
  - ✓ seules les feuilles cachent
  - ✓ autres sont des répertoires
- CRISP
  - ✓ répertoire centralisé

Efficacité

Efficacité : que mesurer ?

- faciles à mesurer
  - ✓ BP économisée
  - ✓ *hit-rate* (max 40–55 %)
  - ✓ *byte hit-rate* (max 20–35 %)
  - ✓ temps de remplissage
- plus difficiles
  - ✓ gain en temps de réponse
- difficultés
  - ✓ qu'est-ce qu'un hit ?
  - ✓ ...
- taux d'utilisation par les usagers
- web dynamique peu cachable

Efficacité : impact de TCP

- l'impact de TCP : ex. modem 28 kbps
  - ✓ BP : coût des ABORT annule les gains
  - ✓ réduction de délai = 3 %
  - ✓ sauf si connexions persistentes
  - ✓ proxy = cache de connexions

Efficacité : fournisseurs

pratique du *cache busting* pour compter le nb de hits

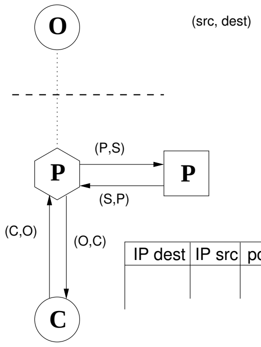
- rfc2277 "Simple Hit-Metering and Usage Limiting for HTTP"
- extension HTTP, compte les hits en "best-effort"
- en-tête `Meter` *(hop-by-hop)*
- formation d'un arbre de mesure
  - ✓ chaque nœud compte les *use* et *re-use*
  - ✓ reporte à son parent au prochain GET
  - ✓ nœuds feuilles font du *cache busting*
- autres approches : méthodes statistiques

Efficacité : conf. clients

- explicite
  - ✓ manuelle
  - ✓ à partir d'un fichier "AutoConfig"
  - ✓ forcée
- aucune
  - ✓ interception (cache "transparent")
    - ☞ éventuellement sélective
- automatique
  - ✓ WPAD

Interception niveau transport

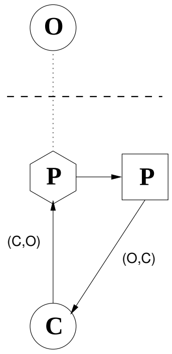
- switch L4 = proxy TCP transparent
- examen du premier paquet (`SYN`)
  - ✓ adresses IP (C,O)
  - ✓ port TCP
- triangulation possible
- WCCP



IP dest	IP src	port	S

Interception niveau transport avec triangulation

- canal descendant
- réseau d'accès
- réseau du serveur origine
- *spoofing*
  - ✓ P et S directement connectés
  - ✓ configuration spéciale de S



Interception niveau application

- URL
- en-têtes HTTP
- proxy transparent *in-path*
  - ✓ utilisé dans les *Access Content Networks*
  - ✓ – point de faiblesse pour **tout** le trafic
- proxy filtrant redirecteur
  - ✓ triangulation possible (mais délicat)

Pb. avec interception

- pas très catholique
- si pb proxy, accès coupé
- authentification/personalisation
- recherche DNS inutile
- TCP non persistant

WPAD

Web Proxy Auto-discovery Protocol

- trouver l'URL du fichier de conf `http://<host>:<port><path>`
- chaos de mécanismes d'auto-conf
- solution court terme
- essayer dans l'ordre :
  1. DHCP
  2. SLP (Service Location Protocol rfc2608)
  3. DNS a) SRV b) TXT c) A
- 1 et 3c obligatoires
- le fichier précis en fct de `User-Agent`

Efficacité : pages dynamiques

problème préoccupant

- encourager le *client side*
- déplacer les fcts serveur ➠ proxy
- Dynamai
  - ✓ ex. site d'enchères
  - ✓ uniq. en *reverse proxy*
- Active-Cache
  - ✓ URL associée à une applet
  - ✓ applet exécutée dans le proxy à chaque hit

Active Cache

- en tête HTTP `CacheApplet`
- proxy exécute l'applet ou fait suivre la requête
- applications
  - ✓ journal d'accès
  - ✓ rotation bandeau pub.
  - ✓ autorisation (vérification certificat)
  - ✓ expansion SSI, compression Delta etc
- sécurité ...
  - ✓ Java
  - ✓ env. sécurisé

Prefetching

Comment augmenter le *hit rate* ?

- en anticipant les requêtes
  - ✓ doc. populaires sur le serveur
  - ✓ prédiction sur l'utilisateur
  - ✓ trois situations
    - ☞ client – serveur
    - ☞ proxy – serveur
    - ☞ client – proxy (cas des modems)

Gestion de l'espace de stockage

- traditionnels
  - ✓ LRU, LFU
  - ✓ Pitkow/Recker (LRU sauf si tous le m̂ jour)
- basés sur des clés
  - ✓ taille
  - ✓ LRU-min $\{x, taille(x) \geq S\} \cap \emptyset \rightarrow LRU\{x, ...\}$ sinon S := S/2
  - ✓ LRU-threshold, objets de taille > S jamais cachés
  - ✓ Lowest latency first
- fcts de coût
  - ✓ GreedyDual-Size – coût, choisit min(coût/taille)

caractéristiques du trafic

Efficacité : performance

- taille disque : 10-100 go (pas linéaire)
  - ✓ dépend de l'environnement
- tps de "chauffe" n jours, voir semaines ...
- nb req. simultanées (ress par req/ress totales)
  - ✓ + durée moy. ➠ nb. req/s
  - ✓ dépend fortement de l'env.
  - ✓ ex. cache de Satellite

Efficacité : implémentation

- arch. spécialisée micro-noyau / classique

points fondamentaux

- index des objets en mémoire
- gestion efficace du DNS
- accès disque performants
  - ✓ Squid : organisation du disque ➠ *cache-dir*s
  - ✓ accès disque
    - ☞ un `diskd` par cache-dir
    - ☞ échange de messages
    - ☞ mémoire partagée

Exemple de cache : Squid

- développé par le NLANR
- proxy cache pour HTTP, FTP et d'autres
- proxy pour SSL
- support des hiérarchies
- support de ICP, HTCP, CARP, Cache Digests
- cache transparent
- WCCP v1 (v2 plus tard ?)
- reverse proxy
- SNMP
- cache les résultats DNS

Utilisation

- IRCACHE
  - ✓ 10 caches maillés, ICP
  - ✓ servent hiérarchie de 1000 caches
  - ✓ Squid
  - ✓ matériel Intel, 512 Mo, 24 Go
- ENST
  - ✓ ? ? ?

En conclusion

- ce qu'on veut :
  - ✓ faibles délais
  - ✓ robustesse
  - ✓ transparence
  - ✓ scalabilité
  - ✓ efficacité
  - ✓ adaptativité
  - ✓ stabilité
  - ✓ équilibrage de charge
  - ✓ hétérogénéité
  - ✓ ... simplicité !
- choix sur :
  - ✓ architecture
  - ✓ coopération des caches
  - ✓ docs. à cacher
  - ✓ partage
  - ✓ cohérence
  - ✓ ...
- difficultés
  - ✓ cohérence forte
  - ✓ adaptativité
  - ✓ contenus dynamiques