Cache Web: un état de l'art des techniques et prototypes

Jean-Christophe Mignot

Laboratoire de l'Informatique du Parallélisme

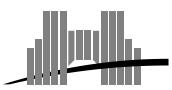
École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON
n° 5668

Cache Web: un état de l'art des techniques et prototypes

Jean-Christophe Mignot

Novembre 1999

Research Report N° RR1999-52



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

Cache Web: un état de l'art des techniques et prototypes

Jean-Christophe Mignot

Novembre 1999

Abstract

This paper presents a survey of the state-of-the-art techniques and prototypes for Web caches. The basic principles are presented, the most important hardwares and softwares approaches are described.
This work was partially supported by the French Ministry of Industry within the CHARM Project, contract n° 99.2.93.0116

Keywords: Web caches, distributed systems

Résumé

Ce document présente un état de l'art des prototypes et techniques des caches WEB actuelles. Les principaux matériels et logiciels disponibles sur le marché sont présentés. Les principales approches dédiées aux machines parallèles sont décrites.
Ce travail a été partiellement financé par le ministère de l'Industrie dans le cadre du contrat CHARM, convention n° 99.2.93.0116

Mots-clés: Caches Web, systèmes distribués

1 Introduction : proxies et caches Web

L'explosion de l'utilisation de l'Internet crée de sérieux challenges pour les prestataires de services Internet et les opérateurs réseaux. A une heure de pointe, l'accès à un document ou la soumission d'une requête à un moteur de recherche peut prendre un temps considérable qui constitue un frein à l'utilisation interactive de l'Internet, un phénomène souvent désigné sous le nom de "World Wide Wait".

Ce problème est principalement lié à un accroissement quasi exponentiel de la demande en bande passante suite à la multiplication des utilisateurs et à la banalisation des informations multimédia. Malheureusement, même si les solutions basées sur des satellites permettent des gains importants sur les solutions terrestres, augmenter la bande passante des réseaux longue distance n'est possible qu'au prix d'investissements considérables. Dans un futur proche, des solutions de type ADSL (Asymmetric Digital Suscriber Line) et MMDS (Microwave Multiwave Distribution System) permettront des débits élevés en utilisant respectivement des boucles locales téléphoniques ou hertziennes.

Les utilisateurs ne pourront bénéficier des performances accrues de ces réseaux que si les informations accédées sont hébergées sur un serveur directement connecté à la boucle locale haut débit. Ceci implique que des fonctions de cache Internet doivent être implantées dans la tête de réseau par un serveur suffisamment puissant pour qu'il ne constitue pas à son tour un goulet d'étranglement. Ce besoin en puissance est tout particulièrement sensible avec l'avènement de documents incluant des flux audio et vidéo et la multiplication des requêtes à "valeur ajoutée" nécessitant l'évaluation de scripts et de méthodes d'indexation. Un serveur de cache de forte capacité, extensible, et à haute disponibilité apportera aux abonnés d'une boucle haut débit un confort d'utilisation accru en réduisant les temps d'attente et en leur offrant la possibilité d'utiliser des applications incluant des éléments multimédia riches (audio, vidéo, etc.). Ces avantages pour les utilisateurs constituent une valeur ajoutée très intéressante pour les opérateurs et les prestataires de service Internet. Pour les exploitants des boucles locales haut débit, il faut souligner les économies de bande passante sur les liaisons avec la dorsale Internet induites par le cache WEB et les fonctionnalités d'indexation.

Dès le début des années 90, les premières études étaient menées sur la caractérisation du trafic en réseau que ce soit en LAN ou sur l'Internet et sur l'impact des caches. Les buts étaient déjà ceux qui nous préoccupent actuellement : détourner la charge des serveurs surchargés (désignés dans la littérature anglophone par "hot spots"), réduire l'utilisation de la bande passante du réseau, diminuer les temps de latence devenant rédhibitoires pour les utilisateurs et aussi protéger le réseau des clients qui bouclent par erreur et génèrent des requêtes répétitives ([12, 1]).

De nombreuses études ont été consacrées à l'analyse des spécificités du trafic Web (voir [5, 6, 16]). Danzig, Schwarz et Hall ([17]) montrent que l'utilisation de caches hiérarchiques pourrait permettre de réduire de moitié la bande passante utilisée par les transferts `ftp` longue distance, diminuant ainsi de 21% la charge du NSFNET. Braun et Claffy ([8]) présentent des résultats similaires concernant le trafic WEB. Alonzo et Blaze ([4]) et Munz et Honeyman ([28]) montrent que concernant les réseaux locaux, le type de cache utilisé est important (cache hiérarchique *versus* cache "à plat", voir section 3) étant donné la nature statique

des informations accédées.

Dans la suite de cette introduction, nous décrivons en détails les notions de serveur proxy et de cache. La section 2 décrit les principaux outils de cache utilisant l'approche "boite noire". La section 3 décrit les principaux logiciels disponibles sur le marché et leurs particularités. La section 4 est dédiée aux logiciels de cache parallèles. La section 5 aborde les problèmes posés par l'étude des performances des outils de cache WEB.

1.1 Les proxies

Le premier rôle d'un proxy (mandataire en français) est de permettre l'accès à l'Internet depuis un réseau derrière une machine coupe-feu (voir Figure 1). Typiquement, dans un réseau sécurisé, toutes les machines sont reliées entre elles mais une seule est connectée à l'Internet et les communications avec cette machine sont restreintes et contrôlées. Le but est de protéger l'intérieur du réseau d'éventuelles attaques venues de l'extérieur (voir [25]).

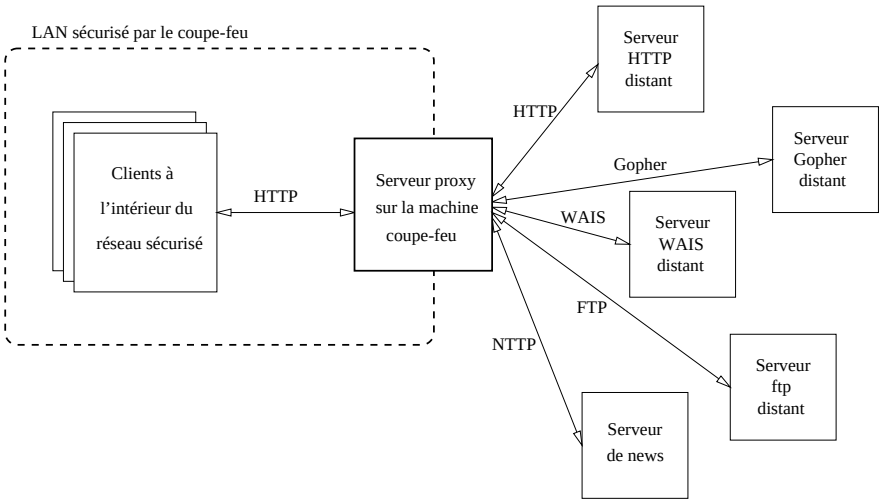


FIG. 1: Proxy et réseau sécurisé : le serveur proxy est exécuté soit sur une machine coupe-feu (cas présenté sur la figure), soit sur une machine du réseau ayant un accès à Internet, soit sur n'importe quelle machine du réseau ayant la possibilité d'accéder au monde extérieur par l'intermédiaire d'une couche logicielle *ad hoc* comme SOCK

Un proxy est un serveur `HTTP` spécial qui est typiquement exécuté sur une machine coupe-feu. Le proxy attend une requête issues du réseau sécurisé, transmet la requête au serveur distant à l'extérieur du réseau local, lit la réponse et la renvoie à l'initiateur de la requête. Il rend donc le coupe-feu perméable de manière sûre, sans créer de trou de sécurité qui permettrait à d'éventuels pirates d'accéder à l'intérieur du réseau local.

Pour les clients Web, les modifications à apporter pour leur faire supporter les proxies sont minimes. Il n'est pas nécessaire de recompiler une version des clients avec une librairie de coupe-feu : un client standard peut être configuré pour devenir un client proxy. L'utilisation des proxies est un moyen standard

pour passer à travers les murs coupe-feu plutôt que d'avoir n versions de clients adaptées à chaque type de mur coupe-feu. C'est d'autant plus intéressant que la plupart des clients du commerce ne sont pas fournis avec le code source.

Il n'est pas nécessaire pour les utilisateurs de disposer de clients `ftp`, `Gopher` et WAIS spécialement modifiés pour passer à travers un mur coupe-feu. Un simple client Web et un serveur proxy suffisent. L'utilisation d'un proxy permet aussi de standardiser la forme des requêtes `ftp` et `Gopher` entre clients par rapport à une situation où chaque client aurait son propre gestionnaire de `ftp` ou `Gopher`.

Un proxy permet au programmeur de clients Web de ne pas avoir à gérer tous les protocoles du réseau pour se consacrer à des spécifications du client plus importantes. Il est possible de n'utiliser que des client "légers" ne reconnaissant que le protocole `HTTP`, les autres protocoles étant gérés de manière transparentes par le proxy. L'utilisation du protocole `HTTP` entre client et proxy ne fait perdre aucune fonctionnalité à l'utilisateur puisque `ftp`, `Gopher` et les autres protocoles du Web sont parfaitement représentés dans les méthodes `HTTP`.

Les client n'ayant pas de DNS peuvent quand même avoir accès au Web, l'adresse IP du proxy est la seule information dont ils ont besoin. Les réseaux utilisant un espace d'adresses privé (comme les réseaux de classe A comme 10.*.*.*) peuvent quand même accéder à l'Internet pour autant que le proxy soit visible aux deux réseaux, par exemple par deux interfaces distinctes.

L'utilisation des proxies permet un enregistrement de haut niveau des transactions des clients incluant l'adresse IP du client, la date, le temps, l'URL, le nombre d'octets et les codes retournés. Tous les champs et méta-informations des transactions `HTTP` peuvent être enregistrés. Ceci n'est pas possible avec les enregistrements des niveaux IP et TCP. Il est en outre possible de filtrer les transactions des clients au niveau du protocole de l'application. Le proxy peut contrôler l'accès aux services des différentes méthodes, les hôtes, les domaines, etc.

D'une manière générale, les développeurs n'ont aucune raison de développer une version coupe-feu de leur client Web. Avec l'utilisation des proxies, ils ont une incitation : les possibilités de cache. Enfin, un proxy est plus simple à configurer qu'une couche logicielle de communication comme SOCK et fonctionne quelle que soit la plate-forme.

1.2 Les caches

L'idée de base des caches est simple : enregistrer les documents collectés sur le réseau dans un fichier local de manière à ne pas avoir à se reconnecter aux serveurs lors de requêtes ultérieures aux mêmes documents. Ce principe est illustré par les figures Fig. 2 et Fig. 3.

La Figure 2 illustre la manière dont le document demandé est récupéré sur le serveur distant (some_host.far.away) puis enregistré localement sur le serveur proxy (www_proxy.my.domain) pour une utilisation ultérieure. La Figure 3 illustre un succès de cache sur le proxy : un document demandé était déjà enregistré dans le cache, c'est la version enregistrée qui est fournie au demandeur, il n'est pas nécessaire de se reconnecter au serveur distant.

Dans la plupart des cas, le même proxy est utilisé par toutes les machines d'un réseau local. Cette situation d'intermédiaire obligatoire pour toutes les machines rend le proxy très attractif pour jouer le rôle de cache des documents

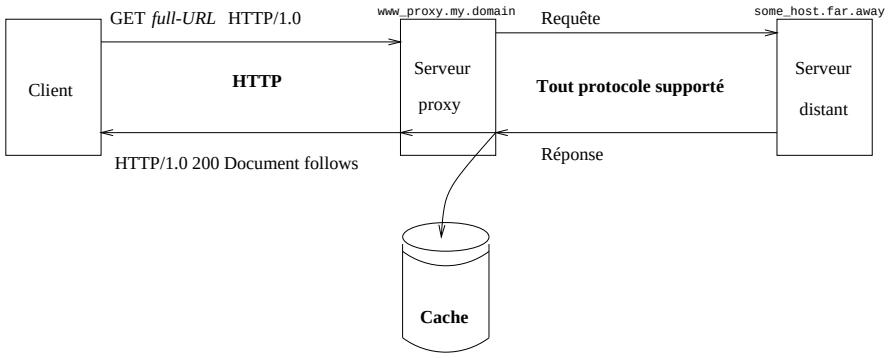


FIG. 2: le principe des caches : première requête

demandés par les clients. Notons que cette possibilité est intéressante même pour les machines qui ne nécessitent pas d'être derrière un coupe-feu afin de diminuer les temps d'accès et de réduire l'utilisation de la bande passante disponible. Le cache permet d'avoir accès à des ressources comme les sites `ftp` ou `Gopher` même quand ceux-ci sont en panne. Les caches peuvent aussi servir pour effectuer des démonstrations en des lieux où Internet n'arrive pas, ou encore, pour lire tranquillement, et "off line", des documents préalablement chargés dans le cache. Notons que ces cas peuvent demander une configuration particulière du cache spécifiant de ne pas vérifier la fraîcheur des documents délivrés (version 1.1 de `HTTP`).

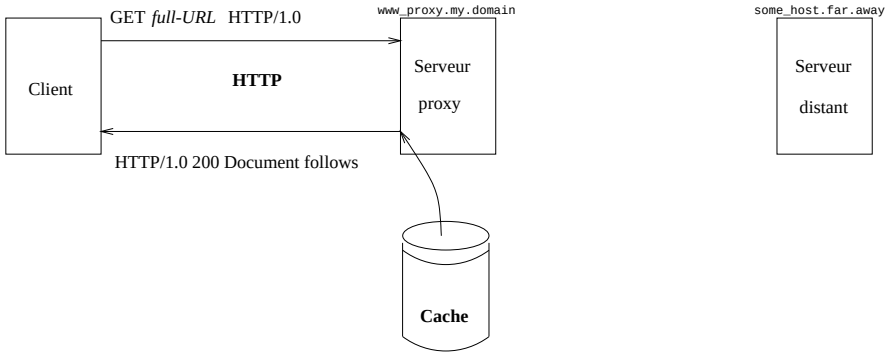


FIG. 3: le principe des caches : requêtes suivantes

Les caches sont plus efficaces sur un proxy que chez chaque client. L'utilisation des disques en est diminuée puisqu'une seule copie des documents est cachée. Le cachage des documents souvent référencés par plusieurs utilisateurs est plus efficace car le gestionnaire de cache peut mieux prédire quels documents doivent être gardés longtemps dans le cache et quels documents doivent être rapidement enlevés.

Il est possible de collecter automatiquement certaines pages depuis leurs serveurs d'origine. On utilise souvent le terme de pré-chargement (pre-fetching en anglais) pour désigner cette méthode. La mise en œuvre du pré-chargement est facilitée quand le nombre de clients est grand car les méthodes statistiques mises

alors en œuvre sont plus fiables. L'utilisation des caches pose problème : combien de temps est-il possible de garder un document dans le cache tout en assurant qu'il soit à jour ? La gestion de l'expiration des documents a été pressentie dans le protocole `HTTP` qui contient un entête d'objet spécifiant la date d'expiration. Cependant très peu de serveurs fournissent actuellement cette information et tant que cet état de fait persistera il faudra utiliser des heuristiques comme par exemple utiliser un estimateur grossier du temps à vivre ("time to live", souvent trouvé dans la littérature sous la forme TTL) des objets.De manière plus importante, les documents collectés sur le WEB sont souvent des documents "vivants" et leur affecter une date d'expiration n'est pas toujours facile. Un document peut rester inchangé pendant très longtemps puis devenir subitement modifié très fréquemment. Ce changement peut ne pas avoir été pressenti par l'auteur des pages. Il ne sera donc pas reflété de manière précise dans les dates d'expirations envoyées avec les versions précédentes du document. Si la date d'expiration envoyée avec la version initiale était, par exemple, d'un mois, et si après un certain temps, elle passe à un jour, tous les caches ayant reçu la première version vont être incorrectement informés.

Quand il est essentiel que la version du document soit à jour, il est nécessaire de se connecter au serveur d'origine pour chaque requête de type `GET`. Le protocole `HTTP` implémente une méthode nommée HEAD qui permet de ne récupérer que l'entête d'un document. Elle permet de vérifier si le document a été modifié depuis le dernier accès. Cependant, dans le cas où le document a été modifié, effectuer une seconde connexion au serveur serait inefficace car le surcoût pour l'établissement d'une connexion est très lourd. Le protocole `HTTP` a donc été enrichi par un entête de requête If-modified-since qui rend possible de faire un `GET` conditionnel. La requête est la même excepté qu'il faut y adjoindre la date du document caché. Si le document n'a pas été modifié depuis les date et heure fournies, il n'est pas retourné au client, seuls sont renvoyées les méta-informations importantes de l'entête de l'objet, comme la nouvelle date d'expiration. Si le document a été modifié, la nouvelle version est retournée, comme pour un `GET` normal.Le `GET` conditionnel permet d'optimiser certains outils comme les outils de mirroring. Le mirroring (de l'anglais mirror, miroir) est une méthode qui consiste à dupliquer systématiquement un ensemble de fichiers prédéterminé depuis un site d'origine. On parle de sites miroirs, ou de miroirs, pour désigner les copies. Cette méthode est la méthode la plus utilisée pour dupliquer de lourdes archives comme certains sites `ftp`. Les miroirs sont entièrement réactualisés régulièrement occasionnant de longues opérations de transfert. Rien ne garantit qu'un miroir soit à jour, la réactualisation étant à la charge des administrateurs du miroir. Un serveur proxy-cache pourrait facilement être utilisé pour automatiser le rafraîchissement d'un miroir, régulièrement, pendant les périodes d'inactivité des clients, en plus des rafraîchissements occasionnés par les réactualisations explicites.

La plupart des serveurs actuels implémentent les `GET` conditionnels. De plus, le protocole `HTTP` est défini de telle manière que si un champ de l'entête d'une requête n'est pas reconnu, il est tout simplement ignoré. Donc, si un utilisateur demande, avec un `GET` conditionnel, un document à un serveur n'implémentant pas cette fonctionnalité, le champ de l'entête correspondant est tout simplement ignoré et le document entier est retourné.

Le mécanisme de cache est basé sur les disques. Il est donc par essence persistant, ce qui signifie qu'il survit aux arrêts du processus proxy ou même de

la machine serveur. Cette spécificité a ouvert la porte, dans le cas où le client et le cache sont sur la même machine, à l'utilisation "off line" du WEB.

1.3 Avantages et inconvénients des caches

Les avantages généralement reconnus de l'utilisation des proxies-caches sont :

- les temps de réponse endurés par les utilisateurs sont diminués,
- la bande passante consommée par les utilisateurs est diminuée,
- les serveurs surchargés sont soulagés,
- les clients buggés ne peuvent pas perturber l'ensemble du réseau.

Dans la pratique, les choses peuvent être un peu différentes. Des études récentes (voir [35]) ont montré que l'utilisation d'un cache ne réduit pas toujours les temps de réponse du côté utilisateur. En effet, si le serveur de proxy-cache est saturé, ses temps de réponse peuvent être très longs, en particulier plus longs que le temps nécessaire pour accéder au document directement depuis le serveur d'origine. Si de plus, le document n'est pas dans le cache, et qu'il faut finalement le récupérer sur le serveur d'origine, l'intérêt du cache peut sembler moindre. De plus, la détermination de la taille optimale du cache peut être difficile et influer sur les performances de manière non négligeable.

La cohérence des informations fournies par un cache peut être problématique : si les objets cachés restent longtemps dans le cache, le client risque de se voir servir une vieille version, si les objets cachés restent peu de temps dans le cache, il y a peu de chances qu'ils servent à un autre client. Si la dernière version du protocole `HTTP` permet de résoudre en partie le problème, les versions antérieures ne fournissaient aucun moyen de spécifier la durée de validité des objets. Actuellement, la plupart des serveurs ne fournissent pas de durée de validité précise aux pages qu'ils servent et il faut donc utiliser des heuristiques pour déterminer la durée de séjour dans le cache. La cohérence des informations fournies par les caches est donc de type faible. Des propositions ont été faites qui permettraient de garantir au Web une cohérence forte ([10]).

1.4 Les différents dispositifs

De nombreux outils de cache sont disponibles. Deux approches sont utilisées :

- Approche propriétaire : la machine de cache achetée est intercalée entre le réseau local et l'accès à Internet. C'est elle qui gère les documents reçus de manière transparente pour les utilisateurs et administrateurs. Il y a peu/pas de possibilité de visualisation des performances et d'optimisation. La modularité est faible. La littérature se limite le plus souvent aux documents du constructeur et peu d'informations sont fournies sur le fonctionnement interne de la machine.
- Approche logicielle : le cache est réalisé par un des nombreux logiciels de cache disponibles sur le marché. Les choix matériels et les réglages du logiciel sont à la charge de l'administrateur et peuvent être modifié à tout moment. Si un logiciel libre est utilisé, il peut être modifié à volonté pour améliorer les performances, visualiser certaines données, etc. De nombreuses études sont publiées concernant le fonctionnement des différents logiciels de cache disponibles, leurs performances et les manières de les adapter à différents besoins.

2 Approche propriétaire

L'approche propriétaire est basée sur la fourniture d'une machine clef en mains pour le cache. On qualifie souvent ce type de machine de "boite noire". D'une manière générale, on appelle "boite noire" un dispositif dont le fonctionnement interne est masqué à l'utilisateur. L'idée est similaire à celle du "plug-and-play" : soulager l'utilisateur de connaissances et de manipulations dont il n'aurait pas besoin. Appliqué au domaine des caches WEB, cela donne un dispositif que l'on branche sur le réseau et que l'on manipule à l'aide d'un logiciel fourni. Une fois le dispositif branché et configuré, tout devrait marcher automatiquement et l'utilisateur ne devrait plus avoir à se préoccuper du dispositif installé.

Citons parmi les principaux outils disponibles actuellement les "Cache Engine" de la société CISCO (`www.cisco.com/warp/public/751/cache`), les machines "CacheFlow" de la société Cache Flow Inc. (`http://www.cacheflow.com`) ou encore les machines DynaCache de la société Infolibria (`http://www.infolibria.com/products/f-prod.htm`).

Le principal avantage de cette approche réside certainement dans l'aspect "clef en main" de l'outil. Si les logiciels comme Squid peuvent demander une certaine expertise informatique pour être mis en œuvre, les machines dédiées se présentent sous forme d'un équipement à installer et du logiciel de gestion associé.

Les principaux inconvénients de ce type d'outils résident dans leurs manques de flexibilité et de visibilité. Si les logiciels de cache peuvent être facilement modifiés pour corriger des imperfections ou s'adapter aux évolutions d'un contexte particulier, les machines de type "boite noire" sont la plupart du temps figées. Elles ne pourront donc pas être adaptées aux éventuels nouveaux protocoles de communication, aux enrichissements des standards, etc. De plus, la plupart du temps, en cas d'augmentation du nombre de clients à servir, la seule possibilité offerte par le constructeur est d'acheter le modèle supérieur dans la gamme. Dans le contexte du WEB, où les techniques évoluent à un rythme soutenu et où le nombre d'utilisateurs croit constamment, ce manque de flexibilité peut sembler rédhibitoire.

Les seules interactions avec le matériel sont celles rendues possibles par les logiciels fournis. Or les logiciels fournis n'implémentent pas toujours toutes les fonctionnalités que l'on pourrait attendre d'un cache tant au niveau de la configuration du cache que de la visualisation de ses résultats. Ce manque de visibilité a pour conséquence d'augmenter la difficulté à analyser finement les performances de la machine et à optimiser son utilisation. Cette difficulté est encore renforcée par la réticence des constructeurs à fournir des détails sur le fonctionnement interne de leurs machines.

D'un point de vue moins technique, la logique propriétaire adoptée par ces solutions semble aller à l'encontre de l'évolution du marché. Un organisme ayant acquis une machine de marque lambda ne pourra maintenir et enrichir son système qu'avec du matériel lambda, aux prix fixés et imposés par le constructeur.

Les performances obtenues par de tels systèmes sont très variables et difficilement comparables du fait du manque de standards pour l'évaluation de performance des caches WEB (voir plus loin, section 5). Les performances annoncées par les constructeurs ne sont pas fiables. Début Mars 1999, le premier "bake-off" a été organisé par le groupe Ircache (NLANR) pour tes-

ter les produits de cache HTTP à l'aide d'un outil ouvert et indépendant de tout constructeur dans des conditions les plus réalistes possible (voir [14], http://bakeoff.ircache.net/bakeoff-01). Seuls quelques constructeurs se sont portés volontaires, tous n'ont pas accepté la publication de leurs résultats. Les résultats sont variés et il est souvent difficile de dire qu'une machine est meilleure qu'une autre : toutes ont des forces et des faiblesses et répondent à des exigences différentes. Le seul constructeur de boites noires pour les caches WEB à avoir accepté la participation au bake-off et la publication de ses résultat est InfoLibria. A titre d'exemple, les prix des machines InfoLibria-L et InfoLibria-S, utilisées pendant le bake-off, étaient respectivement de $200.000 et de $130.000.

En conclusion, ce type de machine n'est pas intéressant pour des études expérimentales comme celles qui seront menées dans le cadre du projet CHARM du fait de leur manque de transparence et d'ouverture. Il est certainement plus adapté au marché très concurrentiel des ISP de par la simplicité d'utilisation et la rapidité de mise en œuvre.

3 Les logiciels

Dans cette partie nous décrivons les principaux logiciels disponibles sur le marché. Certains sont gratuits, d'autres payants. L'offre en la matière évolue très rapidement et de nouveaux logiciels de cache apparaissent régulièrement. Nous ne serons donc pas exhaustifs mais chercherons plutôt à dégager les principes de base utilisés actuellement par les différentes solutions logicielles.

Le premier proxy-cache à voir le jour fut le CERN HTTPD (voir [25, 26]) développé par le World Wide Web Consortium (http://www.w3.org). Ce serveur souffrait de quelques erreurs de conception (un processus par requête, utilisation du système de fichier Unix inadaptée), particulièrement sensibles quand le serveur était très chargé. Il est possible de bâtir une hiérarchie de cache avec le CERN HTTPD mais de manière limité. Le Harvest Cache a été mis au point pour pallier ses défauts.

Le Harvest Cache ([13]), devenu par la suite le Squid cache ([37]), logiciel gratuit de cache hiérarchique, est le logiciel de cache le plus utilisé de part le monde et le plus étudié. La partie qui suit est consacrée à sa présentation. Squid a levé quelques problèmes scientifiques et nombre de solutions ont été proposées, faisant évoluer le cache hiérarchique Squid vers un cache distribué. La partie 3.4 décrit les évolutions les plus pertinentes proposées pour Squid.

3.1 Présentation de Squid

Squid est, de loin, le logiciel de cache le plus utilisé au monde, c'est aussi sans doute le plus vieux. Il est utilisé sur quasi totalité du réseau universitaire français, RENATER. Il résulte du travail conjoint des universités de Californie du Sud et du Colorado. Le nom initial du projet était le Harvest Project Group. Ce groupe a initié la campagne "cache now" de sensibilisation à l'utilisation des caches sur le WEB (voir http://squid.nlanr.net). Son immense succès est sans doute tout autant dû à sa gratuité qu'à ses performances, nettement au dessus de celles de ses concurrents à sa sortie.

Le simple fait que RENATER utilise le logiciel Squid pourrait suffire à démontrer son succès, mais mieux encore, voici une justification de ce choix issue de la page WEB de RENATER : "Les deux serveurs cache les plus répandus (il y a aussi Spinner et Lagoon mais vous pouvez voir à Yahoo s'il y en a d'autres) sont le serveur du consortium W3 (ex-serveur CERN) et Harvest. Nous avons choisi ce dernier, bien plus perfectionné et mieux maintenu." Il est à noter que Squid est non seulement utilisé pour les serveurs universitaires mais que RENATER souhaite aussi faire participer les prestataires Internet commerciaux à la stratégie de caches coopérants au niveau de la France.

3.2 Fonctionnement de Squid

Squid utilise une structure hiérarchique de caches coopérants comme représentée Figure 4 et 5 afin de réduire la demande en bande passante sur les liens grande distance et de diminuer la charge sur les serveurs d'information Internet. Les échecs de cache sont résolus grâce aux caches situés plus hauts dans la hiérarchie. En plus de la relation fils-père, le cache utilise la notion de fratrie : les frères sont les caches de même niveau dans la hiérarchie. Ils sont utilisés pour répartir la charge de cache. Chaque cache dans la hiérarchie décide indépendamment de récupérer la référence demandée depuis le site d'origine ou depuis les caches père ou frères en utilisant un protocole de résolution très simple nommé ICP. Le principe de fonctionnement est décrit en détails dans la suite de cette section.

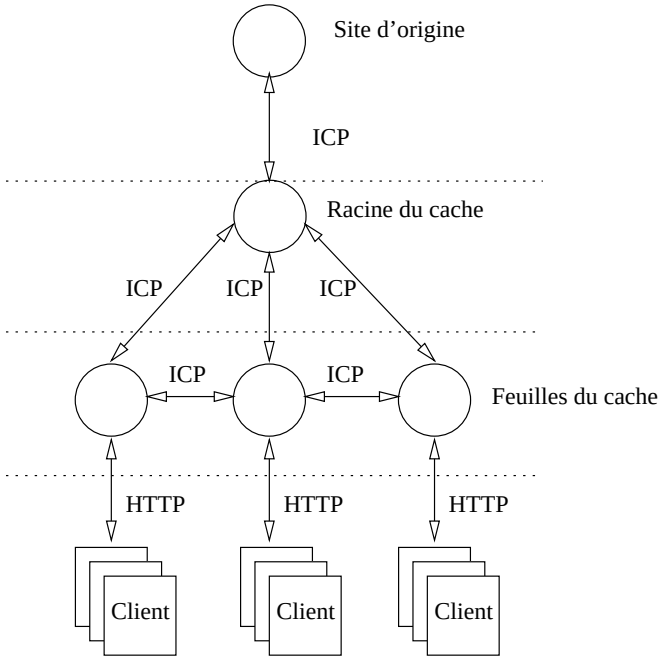


FIG. 4: L'architecture hiérarchique et les communications de Squid

Le texte de l'URL est d'abord analysé. Si il contient une des chaînes de caractères contenues dans une liste paramétrable de chaîne de caractères, alors

l'objet est récupéré directement depuis le serveur d'origine plutôt que via la hiérarchie de caches. Cette possibilité est offerte pour forcer le cache à résoudre les URL non cachables ("cgi-bin") et les URL locales directement depuis le serveur d'origine. De la même manière, si le nom de domaine de l'URL correspond à une entrée d'un liste paramétrable de chaînes de caractères, alors la référence est résolue via le parent dédié à ce domaine.

Sinon, quand un cache reçoit une requête pour une URL qui échoue, il effectue un appel de procédure distante à tous ses frères et parents afin de déterminer si l'un peut répondre à la requête. Si oui, le cache récupère l'objet depuis le site qui a répondu le plus rapidement. Une option permet d'enrôler le site d'origine dans le processus de résolution de la requête. Quand cette option est positionnée, le cache envoie un "hit" au port UDP echo (celui qui répond aux "ping" des utilisateurs Unix) du serveur d'origine. Quand cette machine renvoie ce message, il est vu par le cache comme un "hit" identique à ceux que pourrait retourner un des frères ou père en cas de succès (le frère ou père possède l'objet localement). Cette option permet de retrouver l'objet depuis son site serveur dans le cas où celui-ci est plus proche que n'importe lequel des frères ou père.

Un cache résout une référence avec le premier frère, père ou site d'origine qui retourne un paquet "Hit" ou avec le premier père qui retourne un paquet "Miss" dans le cas où tous les caches échouent et où le serveur d'origine n'a pas renvoyé de réponse avant 2s. Cependant, le cache ne va pas attendre la fin du time-out du serveur d'origine, il va commencer à transmettre dès que les parents et frères auront tous répondu. Le but de ce protocole est de résoudre les références en utilisant la source la plus efficace. Ce protocole est bien une heuristique car si une réponse rapide à un "ping" indique une faible latence, la bande passante peut être un facteur plus important dans le cas des objets volumineux.

Quand un objet est récupéré depuis son site d'origine, le nœud l'ayant collecté le copie dans son cache local et le transmet au nœud fils demandeur. Celui-ci, à son tour, copie le document dans son cache local et le transmet au nœud fils demandeur. Ce processus est réitéré jusqu'au client demandeur de l'objet.

Les auteurs de Squid préconisent dans [12] de n'utiliser que trois niveaux de caches pour n'ajouter que peu de latence par rapport à un accès sans cache. Le seul cas où le cache ajoute un délai de latence notoire est quand son père est en panne et que le fils ne le sait pas encore. Dans ce cas, les références aux objets sont différées de 2s, le time-out du cache père-vers-fils, et Squid n'attendra temporairement plus de réponse de ce nœud. Quand on augmente la hauteur de la hiérarchie, le nœud racine devient responsable d'un nombre croissant de clients et peut devenir un goulet d'étranglement. Il est recommandé de terminer la hiérarchie là où la bande passante est abondante.

Les requêtes entre les nœuds du cache hiérarchique utilisent le protocole ICP (Internet Cache Protocol). Ce protocole (voir [23, 39, 38, 40]), initialement développé pour le Harvest cache, est un protocole léger orienté datagramme, i.e. il se peut que le message envoyé n'atteigne jamais son destinataire et aucun chemin de connexion n'est établi entre l'émetteur et le récepteur pour d'éventuels autres messages.

3.3 Des problèmes avec Squid

De nombreuses études se sont intéressées à l'amélioration des performances de Squid. L'une des premières a été proposée par Povey et Harrison dans [29] et l'idée de base reprise dans de nombreuses approches ([18, 35, 33, 21, 36]). Elle se propose de résoudre les problèmes apparus avec l'utilisation de Squid à grande échelle. En effet, si Squid répond correctement aux attentes de réduction d'utilisation de la bande passante du réseau et de diminution de la charge des "hot spots", les temps de latence utilisateurs ne sont pas forcément réduits.

Il apparaît à l'utilisation que si le nombre d'utilisateurs de Squid devient important, les caches des nœuds intermédiaires deviennent rapidement surchargés et constituent des goulets d'étranglement. La figure 5 permet de mieux comprendre le processus. Les nœuds marqués "CN" représentent les nœuds clients, ceux marqués "L1", les caches de niveau 1, ceux marqués "L2", les caches de niveau 2, "L3", le dernier nœud de la hiérarchie. L'espace disque consacré au cache est représenté par les disques marqués "C". Pour qu'un nœud maintienne un taux de succès optimal, et fournisse le maximum de réduction des transferts redondants, son cache local doit contenir l'intersection des contenus de tous ses caches fils. Pour les nœuds L2 et L3, les espaces disque requis peuvent être très importants. En théorie ([29]), les nœuds intermédiaires devraient avoir un espace de stockage proportionnel au nombre total d'utilisateurs pour maintenir un taux de succès en octets (byte hit rate) constant. Si la taille des caches est insuffisante, le taux d'échec de cache augmente ainsi que la consommation de bande passante, etc. L'efficacité de l'approche hiérarchique diminue donc avec le nombre d'utilisateurs et la taille des fichiers, ce qui est plutôt mal adapté aux évolutions du marché.

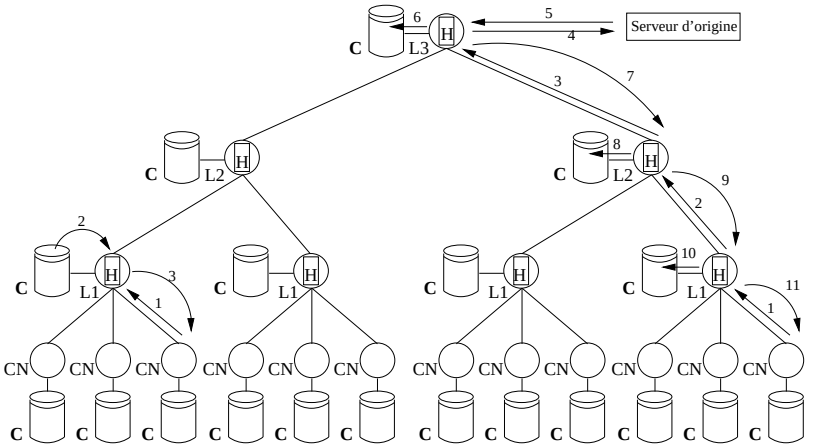


FIG. 5: Répartition des données dans un cache hiérarchique. La transaction de gauche est un succès de cache sur le premier père, celle de droite un échec

Un autre problème apparu avec Squid est celui de la maintenance de la hiérarchie. En effet, chaque nœud doit maintenir des informations sur la localisation de tous les voisins afin de pouvoir rediriger les requêtes vers eux. Si cette tâche est simple quand les différentes machines de la hiérarchie sont locales, elle devient complexe quand les différents nœuds sont distants, gérés par

des administrateurs distincts et autonomes. La nature dynamique de l'Internet rendra par exemple très compliqué pour les administrateurs de tenir compte des ajouts et suppressions de serveurs cache. Le problème est similaire à celui de la maintenance des tables de correspondance nom-adresse des serveurs de l'Internet résolu actuellement par l'utilisation du Domain Name System (DNS, voir [2]).

L'utilisation du protocole ICP a souvent été critiquée (voir [18, 33, 35]). Ce protocole est typiquement utilisé entre les nœuds de la hiérarchie de caches pour interroger leurs contenus. Les messages ICP sont transmis via UDP. Un message est constitué d'un entête de 20 octets et d'une URL. Un cache envoie un message "ICP_QUERY" à ses voisins. Ceux-ci répondent par un "ICP_HIT" ou "ICP_MISS" pour indiquer la présence ou l'absence de l'objet nommé par l'URL dans leurs caches. Le problème vient principalement de ce mode d'échange en question/réponse. Les requêtes ICP sont traitées très rapidement et le temps de latence est très légèrement supérieur au round-trip time du réseau. Mais ces temps peuvent être très longs pour des serveurs distants. Outre les problèmes liés aux conditions du réseau, ICP est peu extensible. Dans la plupart des configurations, l'interrogation de n voisins est réalisée en envoyant n requêtes ICP. Aucune optimisation n'est mise en œuvre. La bande passante utilisée et les charges CPU induites par le protocole sont d'autant plus grandes que le nombre de nœuds est grand. Une étude expérimentale utilisant Squid ([18]) a mis en évidence une réduction importante des performances induite par ICP pour un nombre de voisins aussi petit que 4 et en utilisant un réseau rapide à 100Mb/s.

Enfin, le coût des sauts dans la hiérarchie de cache n'est pas négligeable contrairement à l'idée souvent admise (voir [35] pour une étude expérimentale et [30] pour une approche formelle). Aussi bien les succès sur les caches intermédiaires que les échecs subissent des délais additionnels dus à la propagation en mode "store-and-forward" des requêtes dans le cache hiérarchique. De plus, quand les nœuds intermédiaires doivent servir un grand nombre de clients distribués sur un large territoire, les délais du réseau peuvent être très importants et donc diminuer l'intérêt, pour l'utilisateur, des succès sur les caches distants.

3.4 Les solutions actuelles

Pour pallier ces problèmes, les solutions actuelles proposent de ne pas cacher les documents au niveau de tous les nœuds intermédiaires. En revanche, ces nœuds peuvent stocker des tables contenant des informations relatives à la place des pages cachées. Un exemple est représenté Fig. 6 où les "H" représentent les tables de localisation (Hint en anglais). L'arrangement hiérarchique du cache permet de traiter les recherches efficacement et de répartir la charge sur les différents nœuds. On parle de caches distribués par opposition aux caches hiérarchiques précédents, on trouve aussi l'expression "séparation des données et des méta-données". Les différentes solutions proposées varient essentiellement par la manière dont les requêtes sont traitées et par la diffusion des informations dans la hiérarchie.

3.4.1 Distributed cache

Dans le système proposé par Povey et Harrison ([29]), seules les feuilles cachent les documents. Les autres nœuds de la hiérarchie sont utilisés pour

stocker des informations relatives aux contenus des caches. Le problème posé par l'important espace de stockage nécessaire pour les nœuds intermédiaires est donc levé. Une requête est transmise de fils en père comme avec Squid (les communication avec les frères ne sont pas utilisées), sauf que dans le cas d'un succès de cache, le nœud intermédiaire ne retourne plus le document, mais un pointeur référençant le document dans un autre nœud feuille où l'émetteur ira le chercher. Quand un nœud feuille rapatrie/détruit un document, il en informe les nœuds intermédiaires par messages.

Le principal avantage de cette méthode par rapport à Squid est de ne pas exiger d'immenses espaces disques sur les nœud supérieurs de la hiérarchie. De plus, la gestion de la hiérarchie du cache est simplifiée puisque les nœuds n'ont plus à connaître la localisation de leurs voisins. Enfin, les auteurs soulignent la facilité avec laquelle leur système pourrait être adapté pour prendre en compte la duplication de type miroir (mirroring). Les auteurs ont réalisé une étude de performance du Distributed Cache basée sur une simulation pour différentes structures de la hiérarchie. Leur étude met en évidence un gain de performance minime par rapport à Squid. D'autres auteurs ([31]) ont montré dans une étude théorique des résultats similaires. Les auteurs du Distributed Cache concluent à la supériorité de leur approche du fait de son économie et de sa simplicité.

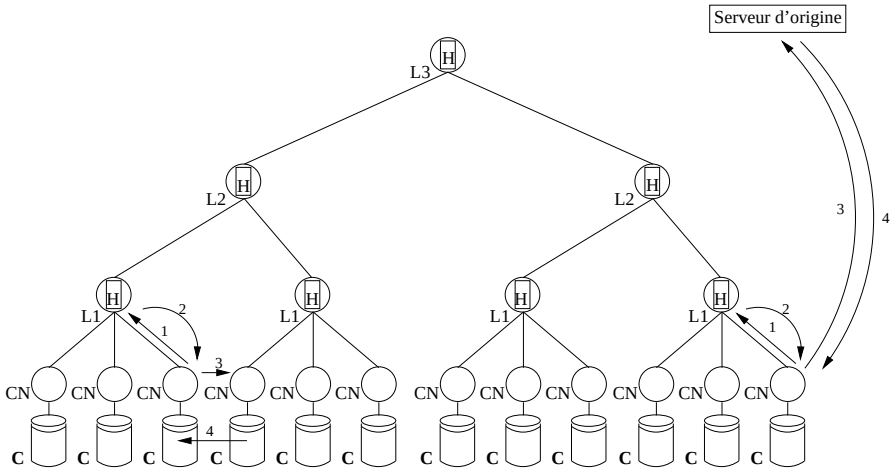


FIG. 6: Répartition de données dans un cache distribué. La transaction de gauche est un succès de cache et le document est récupéré dans un cache voisin, celle de droite un échec

3.4.2 Le cache de l'Université du Texas

Tewari, Dahlin, Vin et Kay ([35]) proposent un schéma où les feuilles du cache hiérarchique, censées typiquement représenter un ISP, cachent les documents issus de l'Internet tout en ayant une table de localisation des documents du cache (voir Fig. 6). Ce schéma représente fidèlement la réalité où les ISP ne sont pas prêts à fournir un accès direct à Internet aux clients mais préfèrent leur imposer de traverser leur cache. Le design proposé tient compte de la distance des caches entre eux : les tables de localisation ne gardent que les copies les plus proches. Quand un client ne possède pas un objet dans son cache, il demande au

cache L1 auquel il est connecté. Celui-ci examine son cache local et sa table de localisation. Si l'objet n'est ni dans son cache local ni dans sa table de localisation, la requête est transférée au site d'origine. Si l'objet est dans le cache, il est transféré. Si la table de localisation indique qu'un des nœuds feuille a l'objet, la requête lui est transférée et le cache retourne l'objet. La hiérarchie est seulement utilisée pour maintenir et propager les informations de localisation.

L'originalité de cette approche est de combiner une hiérarchie extensible des tables de localisation avec le cachage de ces tables près des clients. Les auteurs affirment obtenir des temps de réponse améliorés d'un facteur 1.27 à 2.30 par rapport à un cache hiérarchique classique. Les auteurs proposent aussi de combiner des algorithmes de "push" aux caches afin de diminuer les temps de latence des succès sur des caches éloignés.

3.4.3 CRISP

Pour CRISP (voir [21, 20]), les auteurs proposent une alternative à la structure hiérarchique et à la diffusion des interrogations aux autres caches. Les caches CRISP sont constitués d'une collection de serveurs de cache et d'un service commun de localisation. Les serveurs de cache partagent les répertoires de leur caches via le service de localisation. Ce service peut être interrogé en un seul message par les différents serveurs pour connaître le contenu des autres caches. En cas de succès (l'objet a été trouvé dans l'union des différents répertoires), l'objet est transmis directement par le serveur de cache. En cas d'échec, le serveur d'origine est contacté. Le service de localisation est maintenu à jour par les serveurs de cache qui l'informent des objets récupérés et des objets détruits. Les serveurs peuvent être configurés pour dupliquer tout ou partie du répertoire global afin d'équilibrer les coûts, latences et succès de cache en fonction de la taille et de la dispersion géographique du cache collectif. Dans la version initiale, le service de localisation est constitué d'un ensemble de serveurs de localisation. A chaque serveur est affecté un sous-ensemble de l'espace des URL grâce à une fonction de hachage.

Ce principe est proche de celui de CARP (voir plus loin et [36]) et est particulièrement adapté aux besoins des prestataires de service Internet dont les serveurs bénéficient a priori d'une bonne connectivité. Une seconde version a été développée pour répondre aux exigences des serveurs géographiquement distants. Dans cette version, le répertoire global est dupliqué sur certains serveurs de localisation de manière asynchrone. Les requêtes sont traitées localement et la cohérence du répertoire n'est pas assurée. Notons que ce dernier point n'affecte pas la validité du cache mais sa performance. En effet, si une mauvaise entrée du répertoire global dupliqué est utilisée, l'erreur est par la suite corrigée et le serveur d'origine contacté. Une troisième version de CRISP permet de réduire le coût de la duplication du répertoire global. L'idée de base est de repérer un sous-ensemble du contenu du cache qui permettra d'obtenir le plus grand nombre de succès et de ne dupliquer que ces entrées sur les différents serveurs de localisation. Les auteurs ont testés les différentes versions de CRISP contre différentes versions de Squid (voir [19]) mettant en évidence la scalabilité et la simplicité de la première approche.

3.4.4 Cache digest

Pour Cache Digest ([33], `http://squid.nlanr.net/Squid/CacheDigest`), les auteurs proposent que les caches puissent échanger des tables de leur contenus. Si le cache A possède une copie de la table du cache B, il connaît les documents susceptibles d'être dans B. Il peut donc décider de récupérer un objet du cache B sans avoir à contacter celui-ci auparavant. Les requêtes ICP de Squid sont en quelque sorte remplacées par des consultations de tables, nettement plus rapides.

Les tables sont basées sur les hash-code des URL des documents. Les collisions étant autorisées dans ces tables (elles sont supposées rares), elles peuvent indiquer un succès de cache alors que le document cherché ne s'y trouve pas. Le résultat de leur consultation est donc moins précis que celui obtenu par la batterie de requêtes ICP de Squid. En revanche, les délais associés aux requêtes ICP sont éliminés. Les temps de réponse des clients sont donc diminués et les besoins en bande passante peuvent être améliorés. Les tables sont échangées via `HTTP`. La collection de tables est maintenue à jour par l'utilisation des entêtes d'expiration des requêtes `HTTP` associées à chaque table. Une version de Cache Digest a été insérée dans Squid. Comparés aux résultats obtenus avec ICP, les résultats expérimentaux de Cache Digest mettent en évidence une diminution de la latence pour les utilisateurs et une diminution de la bande passante consommée. En revanche, on constate un accroissement de l'espace mémoire consommé, le nombre de faux succès de cache est important et les nœuds du cache doivent avoir des connexions permettant des transferts rares mais volumineux (1Mo toutes les heures).

3.4.5 CARP

Défini en collaboration par l'Université de Pennsylvanie et Microsoft, CARP (Cache Array Routing Protocol, voir [36]) est un protocole permettant de répartir l'espace des URL entre les différents serveurs d'un cache faiblement couplé. Le protocole définit une structure de table d'appartenance à la liste des proxies du cache et une fonction de hachage associée. La fonction de hachage permet de déterminer lequel des proxies de la liste devrait être le possesseur de l'URL si elle a été cachée. La fonction de hachage remplace en quelque sorte les requêtes ICP, les Caches Digests et autres tables de localisation. Quand un client veut un objet, il sait immédiatement où aller le chercher en hachant l'URL. Un serveur ne cache que les documents répondant à sa fonction de hachage, évitant ainsi les doublons. Le comportement d'un tel système à une grande échelle où les temps de réponse peuvent être très variables n'est pas clair : la répartition des URL est indépendante de la connectivité du réseau. La gestion de la table d'appartenance à la liste des proxies du cache pourrait être problématique à grande échelle, de la même manière que la hiérarchie de Squid.

3.4.6 SUMMARY CACHE

Pour Summary Cache ([18]), Fan, Cao, Almeida et Broder proposent que tous les serveurs cache aient un résumé des URL cachées par les autres serveurs. Les résumés sont mis à jour périodiquement et utilisent une représentation économique. Les résumés ne sont donc pas toujours à jour et peuvent générer deux types d'erreurs : les faux échecs et les faux succès de cache (comme la plupart

des méthodes utilisant des tables). Un faux échec apparaît quand un document caché par un autre serveur n'était pas indiqué dans le résumé et que le demandeur est allé le récupérer auprès du serveur d'origine. Un faux succès apparaît quand un objet censé être dans un certain cache n'y est pas. L'erreur est facilement détectée, le serveur d'origine est contacté mais le temps de latence utilisateur est altéré. Dans les deux cas, la validité du cache n'est pas affectée, seuls sont concernés les taux de succès et trafic inter-proxy. Les résumés sont implémentés sous forme de filtres de Bloom ([7]) qui permettent une représentation peu coûteuse avec un faible taux d'erreurs (faux succès de cache). Les résumés sont modifiés quand le pourcentage de taux d'erreur dépasse un seuil choisi par l'utilisateur.

3.4.7 Vers des caches entièrement auto-adaptatifs

Récemment, Zhang, Floyd et Jacobson ont proposé l'utilisation d'une organisation des caches Web entièrement auto-adaptative basée sur des groupes de diffusion (voir [27, 42]). Ces travaux n'ont pour l'instant donné lieu à aucune implémentation.

4 Les caches parallèles

Les études portant spécifiquement sur les caches WEB parallèles sont encore rares sinon inexistantes. Les études portant sur les serveurs `HTTP` parallèles sont plus nombreuses ([9, 43]). Les travaux relatifs aux serveurs vidéo parallèles peuvent aussi être intéressants pour le domaine des caches parallèles ([22, 15]) surtout dans le cadre du projet CHARM.

Il est concevable d'implémenter la plupart des caches présentés dans la section 3 sur un cluster de machines. Les auteurs de CRISP ([21]) estiment que la première version de leur algorithme (Partitioned Synchronous Directory) pourrait être implémentée sur un cluster de proxies avec succès. Le protocole ICP pourrait aussi être utilisé car la plupart des reproches qui lui ont été faits sont surtout applicables aux réseaux grande distance. Les auteurs de Squid notent dans ([12]) que les problèmes posés par l'implémentation du Harvest Cache sont les mêmes que ceux qui ont été résolus auparavant dans le domaine des systèmes de fichiers distribués. L'architecture du Harvest Cache est d'ailleurs très proche de celle du Alex file system ([11]).

Une proposition a même été faite ([34]) pour remplacer le protocole `HTTP` par le système de fichiers distribués AFS (Andrew File System). Il est clair que les systèmes de fichiers distribués fournissent de meilleurs résultats en terme de cache, duplication, gestion et sécurité que le Web actuel. La raison principale pour laquelle des systèmes comme AFS n'ont pas été réutilisés tient sans doute à leur lourdeur et à leur manque d'ouverture. Mais une raison plus fondamentale est que les systèmes de fichiers sont de mauvaises abstractions pour les systèmes d'information actuels. Ils réunissent un ensemble de fonctionnalités qui les rendent rédhibitoires pour certaines applications et la seule manière de modifier ces fonctionnalités est de modifier le système d'exploitation ou de fournir des mécanismes permettant aux applications de spécifier le type de comportement attendu. Par exemple, la diffusion par flot (streaming en anglais) n'a pas d'équivalence quand on adopte le point de vue des systèmes de fichiers distribués

actuels. La communauté de l'Internet a préféré un moyen plus souple pour sélectionner des caractéristiques prédéfinies (par exemple voir une vidéo streamée) : le choix à la carte parmi un ensemble d'outils disponibles sur le marché plutôt qu'un système de fichiers omnipotent. En revanche, dans le cadre d'un cache en cluster, le choix d'un système de fichiers tel AFS est tout à fait concevable. C'est l'option retenue par les auteurs d'un prototype au NCSA décrit ci-après.

Katz, Butler et McGrath (voir [24]) du National Center for Supercomputing Applications (NCSA) ont développé un serveur `HTTP` parallèle et leurs choix sont représentatifs de nombreux travaux sur le domaine. Ils proposent d'utiliser un cluster de serveurs `HTTP` configurés à l'identique. Un contrôleur central distribue les requêtes aux différents nœuds de manière circulaire. Ce contrôleur, appelé aussi distributeur de charge, est une version du Berkeley Internet Name Domain modifié pour affecter un même nom (du type www.ncsa.edu) aux différentes machines du cluster. Le système de fichiers parallèle ASF est utilisé pour maintenir un ensemble de documents cohérents entre les nœuds. ASF permet aux serveurs `HTTP` de voir le système de fichiers comme local, quelle que soit sa localisation effective. Un système de cache copie les fichiers demandés sur le disque local. Les données ont aussi été dupliquées à l'intérieur de ASF afin qu'en cas de défaillance d'un des serveurs les données soient encore disponibles... Il est à noter que BIND et ASF étant portables sur n'importe quelle plate-forme, le cluster ainsi formé est modulable tant au niveau du nombre de machines que de leurs types. Un problème lié à l'utilisation du round-robin DNS est que celui-ci ne tient pas compte de la charge des serveurs : un serveur surchargé recevra autant de requêtes que les autres.Un système similaire est proposé par IBM Ressearch. Il est basé sur un SP-2 ou un cluster de RS6000 et fait aussi office de serveur multimédia. Il utilise un routeur TCP pour distribuer les requêtes vers les différents nœuds du serveur, le système de fichiers multimédia Tiger Shark et le système de fichiers parallèle Calypso pour le stockage des données (voir http ://www.research.ibm.com/webvideo pour de plus amples informations).

5 Mesure de performance de caches WEB

Plusieurs types de mesures peuvent être considérées pour évaluer les performances d'un cache WEB. Les mesures les plus utilisées concernent le temps de délivrance des objets contenus dans le cache et le nombre d'URL servies par seconde. Mais on peut aussi s'intéresser aux nombre maximal de connexions simultanées possibles, à la relation entre temps de réponse et charge, à la relation entre taux de succès et charge ou encore à la fraîcheur des documents.Un problème récurrent pour l'analyse des performances des caches Web est celui posé par la reproduction de conditions réalistes lors du test. Le plus souvent, la solution retenue repose sur l'utilisation de traces réelles collectées soit auprès de prestataires de services Internet, soit auprès d'universités, le plus souvent américaines. Les fichiers de traces réelles sont utilisés pour générer un profil de requêtes semblable à celui des utilisateurs réels. De nombreux fichiers de traces sont disponibles sur le Web. De nombreux serveurs proxy-cache permettent de générer les traces d'utilisation de leurs clients. Le format de trace adopté comme standard de fait est celui de Squid.Les constructeurs de cache (matériel ou logiciel) proposent des outils permettant d'évaluer les performances d'un cache Web, mais ceux-ci sont malheureusement trop souvent partiels et partiaux.

Certaines études tentent d'analyser les performances des caches WEB en calculant la complexité des algorithmes mis en œuvre. Dans cette optique, Rodriguez, Ross et Biersack (voir par exemple [31]) ont comparé les coûts des connexions et des transferts de données pour des caches hiérarchiques et des caches distribués. Ils proposent ensuite un modèle hybride permettant de bénéficier du meilleur des deux algorithmes. Malheureusement, de par la complexité et le nombre des mécanismes mis en œuvre, de telles analyses restent limités à des modèles de la réalité très simplifiés.

On trouve dans la littérature de nombreuses descriptions de protocoles d'analyse réalisés sur des plates-formes expérimentales. Les mesures sont souvent réalisées manuellement, au coup par coup, sans que soient développés d'outils logiciels pour analyser les performances du cache WEB automatiquement.

A titre d'exemple, le dispositif mis en place pour analyser les performances du Summary Cache ([18]) utilise 10 Sparc-20 connectés par un réseau Ethernet à 100 Mb/s représenté par la figure 7. Quatre stations sont configurées en proxy-cache, quatre autres stations sont utilisées pour simuler la charge, chacune exécutant 30 processus client. Les clients se connectent aux différents proxies et émettent des requêtes sans temps d'attente entre elles. La taille des document demandés suit une distribution de Pareto avec $\alpha = 1.1$ et $k = 3.0$. Deux stations sont utilisées comme des serveurs, chacune avec 15 serveurs connectés sur différents ports. Chaque serveur "fork" un nouveau processus pour gérer une nouvelle requête `HTTP`. Le processus attend 1s avant de répondre à la requête afin de simuler la latence du réseau. Les expériences sont réalisées pour des taux de succès de 25% et 45%.

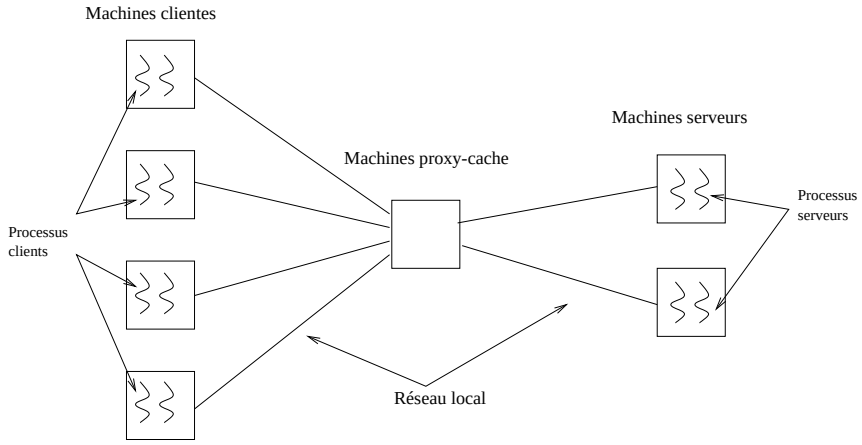


FIG. 7: Architecture pour l'analyse de performance de cache Web

Souvent des traces réelles sont utilisées pour générer une charge réaliste. Il est cependant à noter qu'il faut prendre ses mesures avec beaucoup de précautions : les traces générées par des utilisateurs au travail peuvent être très différentes de celles générées par des utilisateurs (éventuellement les mêmes) à la maison. Le taux potentiel de succès d'un fichier de trace est donc à prendre en considération lors de l'évaluation des résultats. C'est une des raisons pour lesquelles des outils ont été mis au point qui permettent de faire varier le taux de succès associé aux requêtes.

Les tests et outils de tests indépendants sont rares et récents. Almeida et Cao ont développé le Winsconsin Proxy Benchmark ([3]) pour l'analyse des performances du Summary Cache. La structure utilisée est toujours celle représentée Fig. 7. Le Winsconsin Proxy Benchmark comprend un processus maître qui gère l'ensemble du test, des processus client qui génèrent les requêtes, des processus serveur qui simulent les serveurs de pages Web existants sur le réseau. Les auteurs proposent un protocole de test et une manière de l'interpréter. Les limitations de ce produit sont encore grandes : le conditional-GET n'est pas pris en compte, les connexions persistantes non plus, etc. Le premier test à grande échelle indépendant, organisé par le NLANR (National Laboratory for Applied Network Research), date du début de l'année 1999. Il devrait se reproduire sur la base de deux tests par an. C'est le "bake-off" décrit plus en détails ci-après.

5.1 Le premier bake-off

Début Mars 1999, le premier "bake-off" a été organisé par le groupe Ircache (NLANR) pour tester les produits de cache `HTTP` à l'aide d'un outil ouvert et indépendant de tout constructeur dans des conditions les plus réalistes possible (voir [14], `http://bakeoff.ircache.net/bakeoff-01`, `http://polygraph.ircache.net`).

Les outils logiciels du Web Polygraph, `polysrv` et `polyctl` ont été utilisés. Ils permettent respectivement de simuler des serveurs et des clients. Le cache est situé entre les clients et les serveurs simulés. Pendant l'expérience, `polyctl` envoie au cache un flot de requêtes répondant à certains critères, `polysrv` répond aux requêtes envoyées par le cache. L'expérience est découpée en trois phases : une phase de démarrage (remplissage des caches), une phase de plein régime et une phase d'arrêt. Les mesures sont effectuées pendant la phase de plein régime. Les outils `polysrv` et `polyctl` permettent de faire varier la taille des réponses, la proportion d'URL cachables, les temps de latence des serveurs, la popularité des objets, etc. Le but est de pouvoir simuler des taux de charges représentatifs des utilisateurs du réseau.

Il n'existe pas de mesure absolue des performances d'un cache Web. D'aucuns attacheront le plus d'importance au débit en sortie, d'autres à la réduction de la bande passante, d'autres au taux de succès obtenu, d'autres enfin au temps de réponse aux requêtes. Les auteurs du test préconisent aux évaluateurs de baser leur choix sur une somme pondérée des différentes valeurs obtenues. Le choix des pondérations étant laissé aux évaluateurs en fonctions de leurs besoins.

Cependant ces outils semblent encore difficiles d'accès, en attestent les problèmes rencontrés lors de l'étude réalisée par Network Computing magazine (voir [41]) et les réactions de l'équipe d'irc (voir [32] et `http://polygraph.ircache.net/slides` pour l'utilisation de Polygraph). De nombreux paramètres sont à prendre en compte, de nombreux résultats à interpréter et il faut encore un expert en mesure de performance pour mettre en œuvre un test cohérent.

Il est à noter que ce test ne concerne que le protocole `HTTP` et que les standards récents, en particulier ceux utilisés pour les techniques de streaming (RTSP, RTP), ne sont pas pris en compte. Ils ne sont de toute manière pas cachés.

6 Conclusion

Dans ce document nous avons tenté de présenter un état de l'art des caches Web. Loin de prétendre à l'exhaustivité, nous avons plus particulièrement insisté sur le fonctionnement des différents logiciels actuels, sur les rares études concernant les caches Web parallèles et sur la manière d'estimer leur performance.

La suite de nos travaux prévoit la mise au point d'un cache Web parallèle. La plupart des algorithmes de cache présentés ont été intégrés dans Squid. Il est donc probable que dans la suite de nos travaux Squid soit utilisé et que les analyses de performances qui seront menées s'inspireront du bake-off. Sans préjuger des travaux de recherche qui vont suivre, la mise au point du cache Web parallèle fortement couplé pourra s'appuyer sur les travaux réalisés dans le domaine des systèmes de fichiers distribués, des serveurs web parallèles et des serveurs vidéos parallèles.

De nombreux objets multimédia ne sont pas cachés par les logiciels actuels, or leur importance devient de plus en plus grande. Nous nous attacherons par suite à trouver des solutions novatrices pour pallier ces insuffisances particulièrement pour les vidéos streamées du type RealVideo.

De même, les fichiers volumineux ne sont pas cachés du fait de la place trop importante qu'ils occuperaient en cache. Nous étudierons les impacts des méthodes utilisant des miroirs à haut débit afin de réduire les pénalités endurées par les utilisateurs de tels fichiers.

Les possibilités d'indexations des documents stockés dans le cache ou le miroir sont a priori très intéressantes. Elles fourniront aux utilisateurs un moyen de recherche rapide dans l'ensemble des documents du WEB dupliqués localement.

Références

[1] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, and Edward A. Fox. Caching proxies : limitations and potentials. In *Proceedings of the 4th International WWW Conference*, Boston, MA, December 1995. `http://www.w3.org/pub/Conferences/WWW4/Papers/155/`.

[2] Paul Albitz and Cricket Liu. *DNS and BIND in a nutshell*. O'Reilly and Associates, 1992.

[3] Jussara Almeida and Pei Cao. Wisconsin proxy benchmark 1.0. `http://www.cs.wisc.edu/~cao/wpb1.0.html`, 1997.

[4] Matthew Alonso, Raphael Blaze. Dynamic hierarchical caching for large-scale distributed file systems. In *Proc. of the Twelvth Int. Conf. on Distributed Computing Systems*, june 1992.

[5] M. Arlitt and C. Williamson. Web server workload characterization : The search for invariants. In *Proc. of the SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, may 1996. `http://www.cs.usask.ca/projects/discus/VOIR.html`.

[6] Azer Bestavros, Robert L. Carter, Mark E. Crovella, Carlos R. Cunha, Abdelsalam Heddaya, and Sulaiman A. Mirdad. Application-level document caching in the Internet. In *Proceedings of the 2nd International Workshop in Distributed and Networked Environments (IEEE SDNE '95)*, Whistler, British Columbia, June 1995. `http://cs-www.bu.edu/faculty/best/res/papers/sdne95.ps`.

[7] Burton Bloom. Space/time tradeoffs in hash coding with allowable errors. *Communications of ACM*, 13(7) :422–426, july 1970.

[8] Hans-Werner Braun and Kimberly Claffy. Web traffic characterization : an assessment of the impact of caching documents from NCSA's web server. In *Proc. of the Second International World Wide Web Conference*, Oct. 1994.

[9] Richard B. Bunt, Derek L. Eager, Gregory M. Oster, and Carey L. Williamson. Achieving load balance and effective caching in clustered Web servers. In *Proceedings of the 4th International Web Caching Workshop*, April 1999. `http://www.ircache.net/Cache/Workshop99/Papers/bunt-final.ps.gz`.

[10] Pei Cao and Chengjie Liu. Maintaining strong cache consistency in the world wide web. *IEEE Transactions on Computers*, 47(4) :445–457, april 1998.

[11] Vincent Cate. Alex, a global filesystem. In *Proc. of the Usenix File Systems Workshop*, pages 1–11, may 1992.

[12] Anawat Chankhundthod, Peter B. Danzig, Chunk Neerdaels, Schwartz Micheal F., and Kurt J. Worrel. A Hierarchical Internet Object Cache. Technical report, University of South California and Dept. of Computer Science, University of Colorado, Boulder, 1995.

[13] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J Worrell. A Hierarchical Internet Object Cache. In *Proc. 1996 USENIX technical conference*, San Diego, CA, Jan 1996.

[14] C Claveleira. Évaluation des performances des caches http, le 1^{er} bake-off. `http://www.serveurs-nationaux.jussieu.fr/Cacheshow`, 1999.

[15] Asit Dan and Dinkar Sitaram. Multimedia caching strategies for heterogeneous application and server environments. *Multimedia Tools and Applications*, 4 :279–312, 1997.

[16] Peter Danzig, Katia Obraczka, and Anant Kumar. An Analysis of Wide-Area Name Server Traffic : A study of the Domain Name System. In *1992 ACM SIGCOMM Proceedings*, pages 281–292, 1992. `http://www.catarina.usc.edu/danzig/dns.ps.Z`.

[17] Peter B. Danzig, Richard S. Hall, and Micheal F. Schwartz. A case for caching file objects inside internetworks. Technical report, Dept. of Computer Science, University of Colorado, Boulder, 1993.

[18] L. Fan, P. Cao, J. Almeida, and A. Broder. Summary Cache : A scalable Wide-Area Web Cache Sharing Protocol. In *Proceedings of the ACM SIGCOMM'98 conference*, pages 254–265, September 1998. `http://www.cs.wisc.edu/~cao/papers/summarycache.html`.

[19] Syam Gadde, Jeff Chase, and Michael Rabinovich. Directory structures for scalable internet caches. Technical Report Technical Report CS-1997-18, Department of Computer Science, Duke University, November 1997.

[20] Syam Gadde, Jeff Chase, and Michael Rabinovich. Reduce, Reuse, Recycle : An Approach to Building Large Internet Caches. In *The Sixth Workshop on Hot Topics on Operating Systems*, may 1997. `http://www.cs.duke.edu/ari/crisp/`.

[21] Syam Gadde, Jeff Chase, and Michael Rabinovich. A taste of crispy Squid. In *Proceedings of the Workshop on Internet Server Performance (WISP'98)*, June 1998. `http://www.cs.duke.edu/ari/crisp/`.

[22] D. James Gemmel, Harrick M. Vin, Dilip D. Kandlur, P. Venkat Rangan, and Lawrence A. Rowe. Multimedia Storage Servers : A Tutorial. *IEEE Computer*, pages 40–49, may 1995.

[23] Internet cache protocol specification 1.4, 1994. `http://excalibur.usc.edu/icpdoc/icp.html`.

[24] Eric Dean Katz, Michelle Butler, and Robert McGrath. A Scalable HTTP Server : The NCSA Prototype. In *Proc. of the First International Conference on the World-Wide Web*. CERN, Geneva (Switzerland), may 1994. `http://www.cern.ch/PapersWWW94/ekatz.ps`.

[25] Ari Luotonen and Kevin Altis. World-wide web proxies. *Computer Networks and ISDN Systems 27*, 27(2), 1994. `http://www1.cern.ch/PapersWWW94/luotonen.ps`.

[26] Ari Luotonen, Henrik Frystyk Nielsen, and Tim Berners-Lee. Status of the W3C httpd, June 1996. `http://www.w3.org/Daemon/Status.html`.

[27] Scott Michel, Khoi Nguyen, Adam Rosenstein, Lixia Zhang, Sally Floyd, and Van Jacobson. Adaptive Web caching : towards a new global caching architecture. *Computer Networks And ISDN Systems*, 30(22-23) :2169–2177, November 1998. `http://www.elsevier.nl/cas/tree/store/comnet/sub/1998/30/22-23/2052.pdf`.

[28] D. Muntz and P. Honeyman. Multi-level caching in distributed file systems - or - your cache ain't nothin' but trash. In *Proc. of the USENIX Winter Conference*, pages 305–313, january 1992.

[29] Dean Povey and John Harrison. A Distributed Internet Cache. In *Proc. of the 20th Australian Computer Science Conference*, Sydney, Australia, Feb 1997.

[30] P. Rodriguez, K. W. Ross, and E. W. Biersack. Distributing Frequently-Changing Documents in the Web : Multicasting and Hierarchical Caching. In *Selected papers of the 3rd International caching workshop*, Computer Networks and ISDN Systems, pages 2223–2245, 1998.

[31] Pablo Rodriguez, Christian Spanner, and Ernst W. Biersack. Web caching architectures : hierarchical and distributed caching. dont know where published.

[32] Alex Rousskov. Watchdog : Network Computing Review. `http://polygraph.ircache.net/Watchdog/netcomp.html`, 1999.

[33] Alex Rousskov and Duane Wessels. Cache digests. In *Proceedings of the 3rd International WWW Caching Workshop*, June 1998. `http://www-sor.inria.fr/mirrors/wcw98/31/rousskov@nlanr.net.ps`.

[34] Mirjana Spasojevic, Mic Bowman, and Alfred Spector. Using a wide-area file system within the world-wide web. In *Second International World-Wide Web Conference*, Chicago, Illinois, October 1994.

[35] R. Tewari, M. Dahlin, H.M.. Vin, and J.S. Kay. Beyond Hierarchies : Design Considerations for Distributed Caching on the Internet. UTCS Technical report TR98-04, Univ of Texas at Austin, Austin, TX 78712-1188, feb 1998.

[36] Vinod Valloppollil and Keith W. Ross. Cache Array Routing Protocol v1.1. Internet Draft, `http://ds1.internic.net/internet-drafts/draft-vinod-carp-v1-03.txt`, feb 1998.

[37] Duane Wessels. Squid internet object cache. `http://squid.nlanr.net/Squid`, Nov. 1998.

[38] Duane Wessels and K. Claffy. Application of Internet cache protocol (ICP), version 2. RFC 2187, September 1997.

[39] Duane Wessels and K. Claffy. Internet cache protocol (ICP), version 2. RFC 2186, September 1997.

[40] Duane Wessels and K. Claffy. ICP and the Squid Web cache. *IEEE Journal on Selected Areas in Communication*, 16(3) :345–357, April 1998. `http://www.ircache.net/~wessels/Papers/icp-squid.ps.gz`.

[41] Gregory Yerxa. Speedy Performance, Rock-Bottom Price Put Squid Freeware on Top. *Network Computing*, pages 84–91, may 1999.

[42] Lixia Zhang, Sally Floyd, and Van Jacobson. Adaptive Web caching. In *Proceedings of the 1997 NLANR Web Cache Workshop*, June 1997. `http://ircache.nlanr.net/Cache/Workshop97/Papers/Floyd/floyd.ps`.

[43] H. Zhu, T. Yang, Q. Zheng, D. Watson, O.H. Ibarra, and T. Smith. Adaptative load sharing for clustered digital library servers. Technical report, CS, UCSB, 1998.