Conception d'un système de cache Web adapté aux spécificités des utilisateurs

Guillaume Pierre

INRIA projet SOR
B.P. 105 - 78153 Le Chesnay Cedex - France
e-mail : Guillaume.Pierre@inria.fr
Web : http://www-sor.inria.fr/

RÉSUMÉ — Une infrastructure de caches Web peut améliorer la performance des accès des utilisateurs, mais sa conception n'est pas chose facile. Il faut choisir des politiques internes qui fonctionnent bien avec le trafic à gérer, et dimensionner le système. Pour aider à de telles décisions, nous proposons Saperlipopette!, un outil destiné à l'évaluation des qualités de service procurées par différentes configurations. Nous étudions deux jeux de traces Web afin de déterminer leurs configurations optimales et nous montrons que ces trafics différents requièrent des protocoles internes et des dimensionnements différents.
MOT-CLÉS : *Cache Web, configuration, évaluation de performances.*

ABSTRACT — Designing a caching infrastructure to improve the Web performance is a difficult task. It is hard to determine which internal protocols are needed, and how to dimension them. To guide the decision of systems administrators, we propose Saperlipopette!, a tool that can be used to evaluate, *a priori*, the quality of the service offered by each potential configuration. We study two sets of Web traces in order to determine their optimal configurations. We show that these traces require different protocols and different dimensionning.
KEY WORDS: *Web cache, configuration, performance evaluation.*

1. Introduction

La performance d'accès au Web se dégrade régulièrement car le nombre d'utilisateurs augmente plus vite que la capacité de l'infrastructure réseau. Pour améliorer la performance des accès Web à ses utilisateurs, un organisme peut décider d'utiliser un ou plusieurs proxies caches. Or, le choix d'une bonne configuration de cache n'est pas chose simple.

Un cache est composé d'un ensemble de fonctionnalités, qui peuvent chacune être mises en œuvre selon différentes stratégies. En particulier, il existe plusieurs politiques de remplacement (LRU, FIFO, SIZE, etc.) et plusieurs politiques de maintien de cohérence (TTL, Alex, invalidation, etc.). Or, toutes ces stratégies reposent sur des heuristiques. Il faut donc choisir celle qui fonctionne le mieux dans les conditions d'exploitation. De plus, le cache doit être dimensionné (taille du cache, valeur des paramètres de durée de vie, etc.). La question qui se pose alors est de déterminer quel ensemble de politiques est le plus efficace, et quelle est le dimensionnement optimal du cache.

De nombreuses études ont été menées pour déterminer quel est le meilleur choix de politiques. Malheureusement, leurs résultats sont contradictoires [Arlitt 96, Kim 98, Williams 96]. Suivant la nature du trafic Web étudié et les métriques utilisées, l'un ou l'autre des algorithmes tire son épingle du jeu. Cela nous conduit à penser qu'il n'existe pas une configuration unique qui soit optimale en toutes circonstances. Il faut donc, dans chaque cas particulier, étudier la nature du trafic pour concevoir le système de cache et déterminer la configuration qui convient le mieux.

Nous avons développé Saperlipopette!, un outil qui permet à chaque organisme de déterminer sa configuration optimale [**?**]. Il permet à un administrateur de capturer le trafic Web généré par ses utilisateurs, puis de le rejouer dans un simulateur qui indiquera les performances de différentes configurations envisageables de caches.

Dans cet article, nous détaillons l'utilisation de Saperlipopette! pour la conception de systèmes de caches adaptés aux spécificités des utilisateurs. Nous étudions deux jeux de traces différents, respectivement recueillis à l'INRIA et au Digital's Western Research Lab. Nous montrons que leurs configurations idéales diffèrent du point de vue quantitatif (dimensionnement) et qualitatif (choix des algorithmes).

L'article est organisé comme suit : la section 2 présente les algorithmes de cache existants et les diverses évaluations de leurs performances ; la section 3 discute de nos métriques ; la section 4 décrit Saperlipopette! ; la section 5 décrit la recherche d'une configuration optimale pour les deux jeux de traces, et la section 6 conclut.

2. État de l'art

2.1. Les fonctionnalités constitutives d'un cache

Un système de cache peut être vu comme un ensemble de fonctionnalités indépendantes les unes des autres [Baggio 97] :

- Un proxy Web : il reçoit des requêtes en provenance des clients (ou de proxies fils) et émet des requêtes vers les serveurs (ou des proxies parents).
- Un filtre qui décide pour chaque message HTTP, s'il concerne ou non le cache. Par exemple, c'est lui qui décidera de ne pas mettre en cache les documents générés dynamiquement.
- Un module de stockage, qui assure le stockage sur disque des documents en cache.
- Un module de remplacement qui a pour charge de déterminer quel(s) document(s) il convient d'expulser du cache quand il n'y a pas suffisamment d'espace libre pour stocker un nouveau document. Cette fonctionnalité peut être réalisée par un grand nombre d'algorithmes [Williams 96], parmi lesquels nous retiendrons FIFO (le document le plus ancien dans le cache est expulsé), LRU (le document qui n'a pas été accédé depuis le plus longtemps est expulsé), et SIZE (le plus gros document en cache est expulsé).
- Un module de maintien de cohérence qui est chargé de faire en sorte que les documents délivrés aux clients soient à jour par rapport à la version originale sur le serveur Web. Les algorithmes de cohérence forte, tels l'invalidation, sont généralement trop coûteux à mettre en œuvre pour un système à grande échelle comme le Web. Nous étudierons ici les algorithmes TTL et Alex [Gwertzman 96]. TTL affecte à chaque document une durée de vie fixée à l'avance, et l'expulse à expiration de la durée de vie. Alex est une variante de TTL, qui affecte des durées de vie différentes suivant les documents : la durée de vie est proportionnelle à l'âge du document. Ainsi, un document modifié une heure auparavant héritera d'une durée de vie brève, tandis qu'un document qui n'a pas été modifié depuis un mois héritera d'une durée de vie plus importante.

- Éventuellement, un module de coopération qui permet à un cache de tirer parti de la présence d'autres caches proches. Parmi les protocoles de coopération existants, nous citerons ICP [ICP 94], CRISP [Gadde 97b] et Relais [Makpangou 97].

2.2. Les études de performance de caches

Un grand nombre de travaux a été réalisé ces dernières années sur l'évaluation et l'optimisation des performances des caches Web. On peut les classifier en trois catégories: les analyses statistiques sur des traces, les évaluations d'algorithmes élémentaires et les recommandations générales de configuration.

Le premier groupe a pour objectif de caractériser l'activité des utilisateurs et d'en déduire des recommandations sur la conception des caches. Par exemple, l'équipe de Bestavros a montré que les accès Web recèlent une importante localité, mais que la plupart des requêtes redondantes concernent de petits documents [Bestavros 95]. Ce résultat suggère l'utilisation de caches en mémoire [Markatos 96]. Il a également été montré que l'utilisation de caches partagés peut significativement augmenter les performances. Une autre étude intéressante modélise des algorithmes de caches "parfaits" (ils prennent toujours les bonnes décisions, grâce à leur connaissance des requêtes futures) [Kroeger 97]. De tels algorithmes ne peuvent pas diminuer la latence de plus de 26% sans utiliser de techniques de préchargement, ou 56% avec de telles techniques. Ces résultats sont intéressants, mais ils ne donnent que des indications sur l'efficacité des algorithmes; ils ne peuvent pas prévoir la performance offerte par une configuration donnée.

La deuxième catégorie contient les évaluations des bénéfices offerts par des politiques de cache particulières. Plusieurs d'entre elles ont évalué les coûts et bénéfices de différentes politiques de remplacement [Arlitt 96, Cao 97, Kim 98, Williams 96], de protocoles de maintien de cohérence [Gwertzman 96, Worrell 94], de protocoles de coopération [Gadde 97a] ou d'algorithmes de préchargement [**?**]. Ces travaux reposent souvent sur des outils de simulations basés sur des traces. Cependant, ils étudient les algorithmes au sein de modélisations grossières de leur environnement. De telles études procurent des comparaisons intéressantes entre algorithmes, mais ne tiennent pas compte des effets de bord du reste de la configuration du cache sur le comportement du système. De plus, elles ne sont pas conçues pour prédire la performance résultante dans le cas d'un profil d'utilisation donné.

Un certain nombre d'administrateurs système expérimentés ont publié des informations sur des expérimentations à grandes échelle, telles des caches nationaux [Neal 96, Smith 96] ou des infrastructures de caches coopératifs distribués [Melve 97, Wessels 96]. Malheureusement, ces comptes-rendus ne sont pas d'un grand secours quand il s'agit de construire un système de cache pour une population d'utilisateurs donnée. Seuls quelques travaux ont été réalisés sur la personnalisation des configurations. Ils adressent surtout le problème du choix de la taille d'un cache [Duska 97]: *"Les petits groupes d'utilisateurs ont besoin de 1-Go de cache par tranche de 35 000 requêtes/jour, et les plus grands groupes ont besoin de 1-Go par tranche de 70 000 à 100 000 requêtes/jour"*. Saperlipopette! propose de personnaliser de telles recommandations pour un trafic Web donné. De plus, il étend son champ d'action à tous les aspects de la configuration: algorithmes internes de cache, dimensionnement et architecture de caches distribués.

3. Critères de choix d'une configuration

Nous voulons évaluer les gains et les coûts résultants de l'utilisation d'une configuration particulière de cache. En particulier, nous nous intéressons à la qualité de service perçue par les utilisateurs. Un système de cache Web a trois effets majeurs sur son environnement: il réduit la latence des accès, la charge des réseaux longue distance et la charge des serveurs Web. En contrepartie, il introduit des problèmes de cohérence: les documents délivrés par le cache ne sont pas forcément à jour. Les métriques que nous utilisons ici caractérisent la qualité de service perçue par les utilisateurs; ils concernent donc uniquement le gain de performance et l'incohérence des documents délivrés.

La performance du Web (avec ou sans cache) peut s'exprimer soit par la latence moyenne des requêtes, soit par le temps de retrait moyen. La latence mesure la durée entre le début de la requête et la réception du premier octet de la réponse ; le temps de retrait mesure la durée totale de la requête. Le gain de performance peut se définir par le ratio de la latence (ou du temps de retrait) lors de l'utilisation du système de cache, par rapport à une configuration sans cache.

La métrique de cohérence des documents délivrés par le cache sera définie comme le pourcentage de documents périmés délivrés par le système. Pour déterminer si un document est périmé ou non, nous comparons son champ "`Last-modified`" avec celui du document présent sur le serveur à l'instant de la requête. Si les deux estampilles coïncident (parce que le document a été récupéré depuis le serveur, ou parce que le document sur le serveur n'a pas été modifié depuis un retrait préalable), le document est déclaré à jour. Sinon, il est considéré comme périmé.

L'appréciation des importances relatives des différentes métriques varie avec les individus et les conditions dans lesquelles ils sont placés. Par exemple, quelqu'un qui est relié à Internet via une connexion coûteuse et de faible débit s'intéressera en priorité à la réduction de la consommation réseau ; par contre, quelqu'un disposant d'une liaison généreusement dimensionnée, et pour qui les temps de retrait ne posent pas de problème majeur, s'intéressera davantage à la cohérence des documents. Le critère que nous utiliserons dans cet article consiste à optimiser en priorité la durée de retrait des documents, tout en conservant une cohérence raisonnable.

4. Saperlipopette!

Saperlipopette! se présente en deux parties. La première partie consiste à capturer le trafic Web ; la deuxième partie est un système de simulation de caches construit sur un ordonnanceur à événements discrets.

4.1. La collecte d'informations

La collecte d'informations consiste à capturer toutes les caractéristiques du trafic Web nécessaires à la simulation. Pour cela, nous utilisons les traces d'un (ou plusieurs) proxy par lequel transite le trafic Web. Les informations ainsi récupérées étant insuffisantes pour calculer les métriques qui nous intéressent, nous collectons d'autres données grâce à un outil qui contacte directement les serveurs Web. De plus, nous évaluons la qualité de service du réseau.

4.1.1. Collecte d'informations sur la cohérence des documents

Afin d'évaluer la cohérence des documents pour un système de cache donné, Saperlipopette! doit comparer, pour chaque requête, l'estampille du document délivré au client et l'estampille du document qui *aurait dû* être délivré. Ces estampilles sont constituées de la valeur du champ `Last-modified` des documents. Nous appelons la première *l'estampille délivrée* et la seconde *l'estampille idéale*.

Pour générer les deux estampilles, Saperlipopette! a besoin d'une seule donnée en entrée : l'estampille idéale de chaque requête. Lorsqu'il simule un défaut de cache, il considère que l'estampille délivrée est égale à l'estampille idéale. Il met donc en cache le document avec son estampille et journalise un enregistrement où l'estampille délivrée est égale à l'estampille idéale. Lorsque Saperlipopette! simule un *Hit*, il considère l'estampille attachée au document dans le cache comme l'estampille délivrée. Il journalise donc un enregistrement dans lequel l'estampille délivrée et l'estampille idéale sont potentiellement différentes.

Nous capturons le trafic en exploitant les traces d'un (ou plusieurs) proxy Web. Dans le cas où le proxy ne fait pas office de cache, il est assez simple d'obtenir les estampilles idéales, puisque les estampilles idéales sont égales aux estampilles délivrées. C'est cette technique qui a été utilisée chez DEC pour obtenir leurs traces (voir § 4.1.3). Par contre, si l'on extrait le trafic des traces d'un environnement contenant des

caches, les choses deviennent plus délicates. Les traces des caches ne peuvent indiquer que l'estampille des documents qui ont été délivrés lors d'une exécution d'une configuration particulière du système, et non pas l'estampille du document qui *aurait dû* être délivré s'il n'y avait pas eu de cache. Pour répondre aux deux cas, nous avons séparé l'observation du trafic de la collecte d'informations concernant la cohérence des documents.

Chaque nuit, un robot rejoue les requêtes de la journée. Il envoie des requêtes HTTP `HEAD` aux serveurs, et observe le champ `Last-modified` de la réponse. Si nous appelons T_{req} la date à laquelle la requête originale a été effectuée, et $T_{lastmod}$ la valeur du champ `Last-modified` de la réponse du serveur, on peut distinguer deux cas :

- Si $T_{lastmod} \leqslant T_{req}$, alors nous savons que le champ `Last-modified` du document n'a pas changé depuis la requête originale. Nous pouvons donc utiliser $T_{lastmod}$ comme l'estampille idéale de cette requête.
- Si $T_{req} < T_{lastmod}$, alors le document a été modifié entre la requête originale et la requête du robot. Nous n'avons donc pas d'information sur l'estampille idéale de la requête. Pour tenir compte de cette incertitude, Saperlipopette! calcule la métrique de cohérence dans le meilleur et le pire cas, c'est-à-dire en supposant que toutes les requêtes incertaines ont retourné des documents à jour, ou en supposant qu'elles ont toutes retourné des documents périmés. Cela donne un intervalle encadrant le résultat correct.

4.1.2. Collecte d'informations sur la performance du réseau

Par mesure de simplification, nous avons décidé que la simulation supposerait l'absence de pannes de réseau, de partitions, et de déconnexions. La simulation fait également abstraction des délais dûs aux accès disques ou à l'usage du processeur. Pour construire les métriques de performance, Saperlipopette! a uniquement besoin de données sur les temps de transfert réseau. Pour déterminer la performance perçue par les utilisateurs, le simulateur utilise la taille des documents ainsi que les caractéristiques des connexions avec les serveurs.

Pour obtenir la taille des documents délivrés par le cache, on est placé devant le même problème que pour les informations sur la cohérence des documents : les traces des caches ne peuvent pas indiquer la taille des documents qui auraient dûs être délivrés. Pour obtenir la taille idéale des documents, nous avons utilisé la même solution : le robot capture les tailles des documents en sus des estampilles.

Pour déterminer les caractéristiques des connexions réseau, nous avons utilisé une approche très simple : pour chaque serveur Web inclus dans nos traces, nous estimons la latence et la bande passante moyennes observées pour des requêtes HTTP. Pour ce faire, nous avons développé un autre outil dédié à la mesure des performance réseau. Périodiquement, pendant la journée, il envoie une requête HTTP `GET` à chacun des serveurs présents dans les traces. Nous avons décidé de récupérer systématiquement la racine du serveur afin de garantir que tous les documents ainsi récupérés aient approximativement la même taille (quelques kilo-octets), et soient représentatives des requêtes Web habituelles. Ensuite, pour chaque serveur observé, nous utilisons la valeur moyenne des performances mesurées. Nous ne reproduisons pas pour l'instant la dispersion des temps de réponse.

4.1.3. Une méthode alternative de collecte d'informations

Les chercheurs de Digital utilisent une méthode plus simple, mais plus limitée que la notre pour collecter les données [Kroeger 96]. Elle consiste à instrumenter un proxy Web par lequel transite l'ensemble du trafic. Il journalise toutes les informations nécessaires sur les requêtes qu'il gère : estampilles des documents, tailles, temps de retrait, etc. Comme nous le soulignons au paragraphe 4.1.1, cette technique ne peut fonctionner que si le proxy n'a pas de fonction de cache. En fait, pour assurer l'exactitude des résultats, il ne doit se trouver aucun cache dans le système :

- S'il y a un cache en aval du proxy instrumenté, alors les estampilles journalisées par le proxy seront potentiellement fausses : elle correspondront aux documents délivrés par le cache en aval.
- S'il y a un cache en amont du proxy instrumenté, alors celui-ci ne verra pas passer la totalité du trafic généré par les utilisateurs. Il ne devient alors possible que de réaliser une étude sur la partie du système en aval du cache, celui-ci étant considéré comme une constante de l'environnement.

Cette technique de collecte d'information a été utilisée pour obtenir le jeu de traces DEC que nous étudions dans la suite.

4.2. La simulation de caches

La simulation des caches a été conçue de façon très modulaire et paramétrisable [Baggio 97]. Un cache Web est décomposé en un ensemble de modules indépendants qui implémentent chacun une fonctionnalité élémentaire : contrôle de cohérence, remplacement, coopération inter-caches, etc. Pour ajouter une nouvelle politique, il suffit de dériver la classe de base correspondante, et d'implémenter les spécificités de l'algorithme.

Pour mettre en œuvre une configuration particulière, il faut instancier : (i) un serveur simulé qui répondra à toutes les requêtes ; (ii) un ou plusieurs cache(s) avec leur politiques internes, leur dimensionnement et leur mode de coopération ; (iii) et un ensemble de clients, chacun étant pourvu d'un scénario à exécuter. Il faut également indiquer les performances des connexions réseau entre tous les éléments.

4.3. L'ordonnanceur

Bien que nous considérions un système distribué, composé de clients, de caches et de serveurs, notre simulation est mono-processus. Elle repose sur un ordonnanceur d'événements qui déclenche toutes les actions du système : envois de requêtes, arrivées de réponses, expiration des durées de vie des documents, etc.

L'ordonnanceur gère une queue d'événements à déclencher. Un événement est un objet d'une classe C++ qui dérive de la classe de base `EventHandler` : il doit implémenter une méthode `go()` qui définit les actions à effectuer. Chaque événement à exécuter doit être enregistré auprès de l'ordonnanceur avec la date à laquelle il doit être déclenché. L'ordonnanceur est chargé de trier les événements, et de les déclencher en temps voulu. Il gère également le temps simulé : lorsqu'aucun événement ne s'exécute, il peut sans danger avancer le temps simulé, et déclencher l'événement suivant immédiatement.

Pour des raisons de simplicité, l'ordonnanceur n'autorise pas l'exécution concurrente de plusieurs événements. Cela impose une contrainte importante : chaque événement doit être instantané en termes de temps simulé. Dans notre modèle, une opération instantanée (par exemple une requête dans le cache local) est implémentée comme un événement. Par contre, les opérations qui ont une durée non nulle (par exemple une requête sur un serveur) doivent être divisées en deux événements : le premier événement initie l'opération, puis calcule sa date de terminaison, crée un deuxième événement, l'enregistre auprès de l'ordonnanceur, et rend la main. À l'heure dite, l'ordonnanceur déclenche le deuxième événement qui peut ainsi terminer l'opération.

Par exemple, la simulation d'une requête dans un cache est réalisée par deux événements. Le premier réalise le début de la requête : il cherche l'URL demandée dans le cache. S'il la trouve, il retourne immédiatement le document demandé. Sinon, il initie une requête vers le serveur Web. Pour cela, il calcule la date à laquelle la requête arrivera au serveur, il crée un événement `ServerRequest`, et l'enregistre auprès de l'ordonnanceur (figure 1). Ultérieurement, l'ordonnanceur déclenchera le `ServerRequest`, permettant la livraison de la requête au serveur (figure 2).

```
void BeginOfRequest::go() {
  if (cache.lookup(request.url())) {
    /* Succès */
    request.client().deliver_document();
  }
  else {
    /* Défaut de cache */
    Timestamp     t = scheduler.now() + request.duration();
    EventHandler *e = new ServerRequest(request);
    scheduler.register(e,t);
  }
}
```

Figure 1. *Pseudo-code pour l'événement* `BeginOfRequest`

```
void ServerRequest::go() {
  request.get_server()->receive_request(request);
}
```

Figure 2. *Pseudo-code pour l'événement* `ServerRequest`

5. Systèmes de cache personnalisés

Dans cette section, nous montrons comment on peut déterminer une configuration de caches adaptée à ses besoins. Nous étudions pour cela deux jeux de traces, correspondant aux trafics Web de l"INRIA et de DEC WRL.

L'objet de cette étude est de montrer l'utilisation de Saperlipopette! pour l'évaluation de performance des différentes configurations. Pour ne pas alourdir l'étude, nous nous limiterons à un petit nombre de critères de configuration. Nous jouerons donc uniquement sur la taille du cache, sa politique de remplacement (LRU, FIFO, ou SIZE) et à sa politique de cohérence (Alex et TTL).

Pour rechercher une configuration optimale, la démarche naturelle consiste à démarrer avec une configuration quelconque, puis à faire varier l'un des paramètres de configuration pour déterminer sa valeur optimale. On fixe ensuite ce paramètre à sa valeur optimale, puis on fait varier un autre paramètre, et ainsi de suite. Cependant, dans le cas de cette étude, cette méthode conduirait à suivre deux cheminements divergents (pour les deux jeux de traces étudiés). Cela rendrait inutilement complexe la comparaison entre les deux traces. Pour chaque paramètre que nous faisons varier, nous repartirons donc sur des configurations identiques arbitraires pour les deux traces.

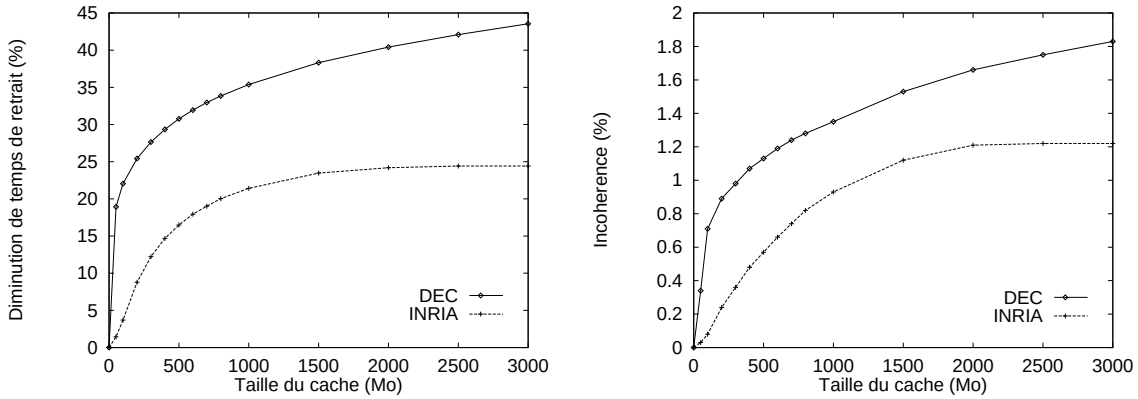
5.1. Les jeux de traces

Notre analyse porte sur deux jeux de traces : l'un a été constitué à l'INRIA, et l'autre est le jeu de traces diffusé par DEC [Kroeger 96].

L'INRIA est composé de cinq unités de recherche. Les traces que nous avons collectées concernent le trafic Web des deux plus grandes unités de recherche (Rocquencourt et Sophia), entre le 13 septembre 1997 et le 9 février 1998. Bien que ces traces aient été recueillies sur deux sites distants de plusieurs centaines de kilomètres, nous ferons abstraction de la distribution des clients : toutes les requêtes seront traitées de la même façon, et dirigées vers un cache unique.

Tableau 1. *Caractéristiques des jeux de traces*

	INRIA	*DEC*
Durée couverte	150 jours	25 jours
Nb de requêtes	1 617 916	24 658 773
Nb de requêtes uniques	1 014 390	3 934 419
Taux de Hit max. théorique	37,3%	84,0%
Nb de machines clientes	1 089	17 354
Taille moyenne des documents	12,1 ko	11,5 ko
Temps de retrait moyen (sans cache)	9,3 s	3,6 s



(a) Réduction du temps de retrait

(b) Incohérence

Figure 3. *Variation des performances et de la cohérence avec la taille*

Les traces DEC ont été collectées du 29 août 1996 au 22 septembre 1996 au Digital's Western Research Lab. Elles sont en libre diffusion et servent régulièrement de "traces de référence" pour les études de performance. Les données disponibles ne permettent pas le calcul de la métrique de latence ; nous ne calculerons donc que la métrique de temps de retrait.

Le tableau 1 montre quelques caractéristiques des jeux de traces. On y voit que les traces DEC correspondent à un trafic très supérieur aux traces INRIA : elles contiennent beaucoup plus de requêtes concentrées sur une durée plus courte. Les propriétés de localité sont également très différentes : les traces INRIA exhibent un taux de Hit maximum théorique de $(1617916 - 1014390)/1617916 = 37,3\%$, alors que les traces DEC sont à 84,0%. Cette différence peut sans doute s'expliquer par les différences de volume des traces : 17354 clients ont sans doute plus de centres d'intérêt communs que 1089.

On constate également que Digital possède une meilleure connectivité que l'INRIA. Cela est sans doute dû au fait que Digital est localisé géographiquement plus près des documents : la majorité des documents consultés sont localisés sur le continent américain.

Tableau 2. *Réduction de temps de retrait*

	LRU	*FIFO*	*SIZE*
DEC (500 Mo)	30,76%	29,02%	21,73%
INRIA (500 Mo)	16,49%	14,61%	21,73%

Tableau 3. *Incohérence*

	LRU	*FIFO*	*SIZE*
DEC (500 Mo)	1,13%	0,53%	0,92%
INRIA (500 Mo)	0,57%	0,42%	1,09%

5.2. Influence de la taille du cache

La figure 3 montre les variations de temps de retrait et de cohérence des documents avec la taille du cache. Les paramètres fixés sont une politique de remplacement LRU et un algorithme de cohérence Alex, avec un coefficient de durée de vie fixé arbitrairement à 30% (que nous noterons Alex(0,3)).

La première constatation est que la performance en temps de retrait augmente avec la taille du cache, ainsi que l'incohérence des documents délivrés. Ces résultats s'expliquent aisément : lorsque la taille du cache augmente, le nombre de documents présents en cache augmente aussi. Pour chaque requête arrivant sur le cache, la probabilité que le document demandé soit présent en cache augmente. D'un autre côté, une plus grande taille du cache permet aux documents de séjourner plus longtemps dans le cache. La probabilité que le document original soit modifié avant l'éjection de la copie du cache augmente.

Il est également intéressant de comparer les formes des courbes de performance et d'incohérence. Dans le cas de l'INRIA, on voit aisément que, pour de petites tailles de cache (entre 0 et 750 Mo environ), la performance augmente sensiblement plus vite que l'incohérence. Cela est sans doute dû à un effet de bord de la politique de remplacement : dans un cache de petite taille, les documents ont peu de chance d'être conservés jusqu'à l'expiration de leur durée de vie ; ils sont éjectés avant cette échéance par l'algorithme de remplacement, afin de faire de la place pour d'autres documents. En un sens, l'algorithme de remplacement se substitue donc à l'algorithme de maintien de cohérence en éjectant de façon plus agressive les documents du cache.

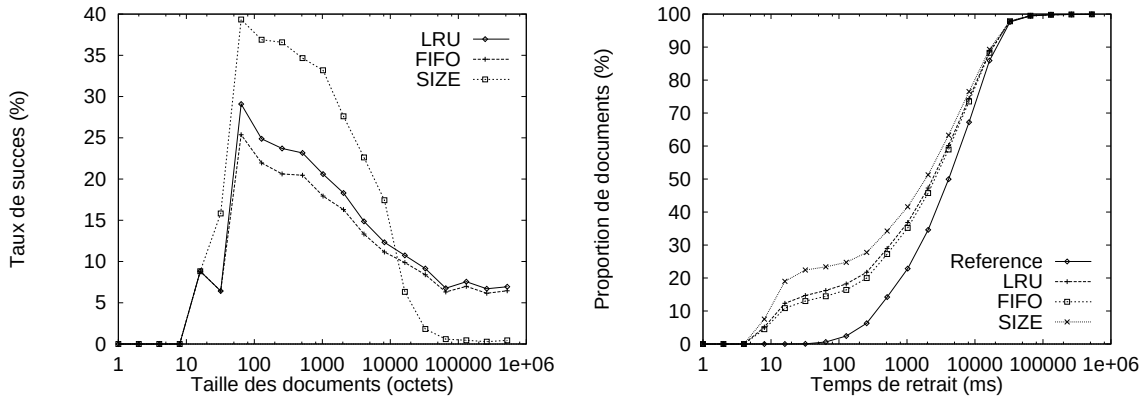
De même, on peut observer que la performance se stabilise aux environs d'une taille de cache de 1,5 Go, alors que l'incohérence ne se stabilise que vers 2 Go. À partir de 1,5 Go, augmenter la taille du cache n'augmente donc quasiment pas la performance, mais accroît sensiblement l'incohérence : la plupart des Hits supplémentaires dans le cache correspondent à des documents périmés.

Les graphiques ne montrent pas la stabilisation des performances et cohérences pour les traces DEC : celle-ci apparaît pour des tailles de cache que nous n'avons pas pu simuler. Cependant, on peut y retrouver les mêmes caractéristiques que pour les traces INRIA : pour de petites tailles de cache, la performance croît plus vite que l'incohérence ; pour de grandes tailles de cache, l'incohérence croît plus vite que la performance.

5.3. L'influence de la politique de remplacement

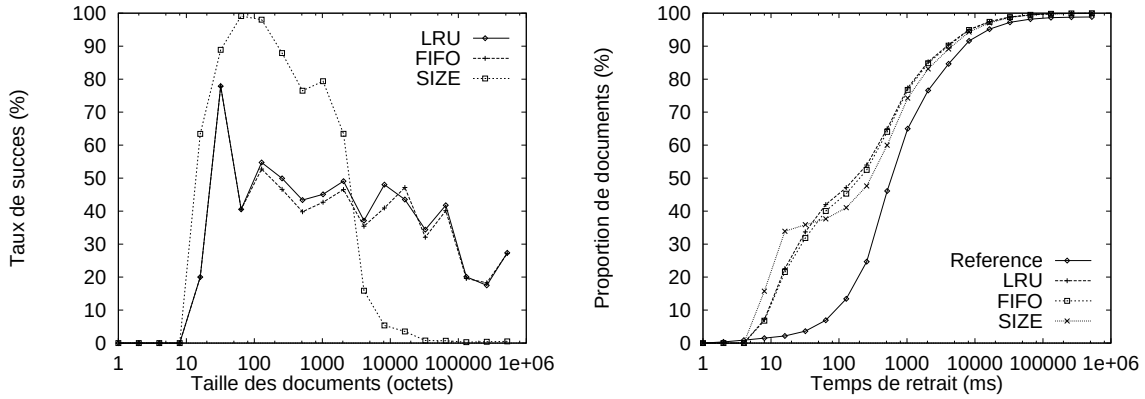
Pour étudier l'impact des algorithmes de remplacement, nous avons utilisé des caches de 500 Mo, avec un protocole de maintien de cohérence Alex(0,3). Nous avons étudié les algorithmes de remplacement LRU, FIFO et SIZE.

Les tableaux 2 et 3 montrent la performance et la cohérence des différents algorithmes. La surprise majeure vient du fait que les deux jeux de traces se comportent de façon très différente. Les traces DEC atteignent une performance maximale pour un remplacement LRU. Cependant, on peut lui préférer le



(a) Distribution des Hits (INRIA)

(b) Distribution des temps de retrait (INRIA)



(c) Distribution des Hits (DEC)

(d) Distribution des temps de retrait (DEC)

Figure 4. *Détail des performances des algorithmes de remplacement*

remplacement FIFO : la performance sera légèrement moins bonne que pour LRU, mais il y aura deux fois moins d'incohérence. La politique SIZE, par contre, perd sur les deux tableaux. Sa performance est nettement moins bonne que les deux autres, et l'incohérence est moyenne.

Les traces INRIA se comportent mieux, au contraire, avec le remplacement SIZE : son utilisation permet de gagner 5% sur LRU, et 7% sur FIFO. On remarque cependant que l'incohérence générée est elle aussi maximale.

Les variations de cohérence entre les différentes politiques sont relativement faciles à expliquer. Les politiques LRU et SIZE permettent à certains documents de séjourner dans le cache jusqu'à expiration de leur durée de vie (les petits documents pour SIZE, les documents populaires pour LRU). Par contre, FIFO traite tous les documents à la même enseigne, et ne leur permet pas de rester longtemps en cache. Il est donc normal qu'il crée beaucoup moins d'incohérence que SIZE et LRU. Dans ce cas, comme dans le cas des caches de petite taille, l'algorithme de remplacement se substitue à l'algorithme de maintien de cohérence en imposant des contraintes plus fortes que lui sur la durée de séjour des documents dans le cache.

Pour permettre de mieux comprendre les différences de performance, la figure 4 présente les détails de la répartition des Hits par tailles de documents (figures 4(a) et 4(c)) et la répartition cumulative des temps de retrait (figures 4(b) et 4(d)).

Les distributions de Hits ont des aspects similaires pour les deux jeux de traces. Les remplacements FIFO et LRU sont proches ; les courbes montent très vite pour atteindre leur maximum vers une taille de document de 100 octets. Il s'agit de documents de petite taille et référencés à de nombreuses reprises dans un même site (généralement des icônes). Ensuite, le taux de Hits redescend en pente régulière. On constate que la courbe LRU est presque toujours au-dessus de la courbe FIFO, ce qui explique les meilleures performances globales de LRU. La courbe SIZE, par contre, monte beaucoup plus vite et beaucoup plus haut. Elle se maintient un moment, puis retombe brusquement : la politique SIZE favorise la présence de petits documents dans le cache et ne permet donc pas de bonnes performances sur les gros documents.

Les distributions de Hits des deux jeux de traces sont semblables, mais pas identiques. Tout d'abord, la valeur des taux de Hits n'est pas la même : pour l'INRIA elle culmine vers 40%, alors que chez DEC elle atteint presque 100% en deux points. D'autre part, la courbe SIZE redescend beaucoup plus tôt dans les traces DEC que dans les traces INRIA (environ 3 ko pour DEC contre 20 ko pour l'INRIA). Cela s'explique par le fait que la politique SIZE favorise les petits documents, et que les requêtes DEC sont plus nombreuses. Il se trouve donc davantage de très petits documents dans le cache, et la frontière entre les "petits" et les "gros" document passe de 20 ko à 3 ko.

Les figures 4(b) et 4(d) montrent, pour un temps de retrait donné, le pourcentage des requêtes qui ont pu être traitées dans ce délai. La courbe "référence" montre la distribution en l'absence de caches. Elle a une forme assez classique : à peu près aucun document n'est récupéré en moins de 100 ms, puis la majorité des documents arrive entre 100 ms et 50 s.

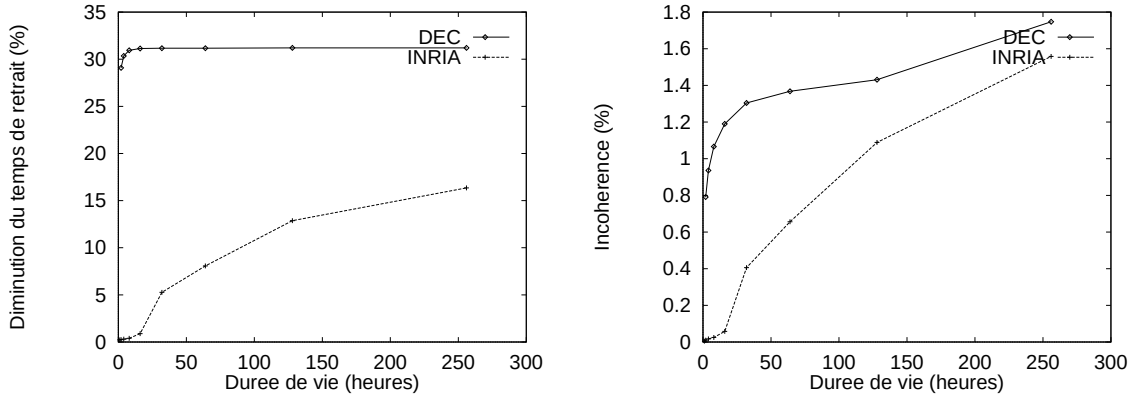
On constate que les courbes FIFO et LRU se placent nettement au-dessus de la courbe de référence. Le gain en temps de retrait se fait surtout sentir entre 10 ms et 10 s. La courbe SIZE se distingue une fois de plus : elle croît très vite (ce qui correspond aux petits documents récupérés dans le cache), puis stoppe brusquement sa progression. On constate que, dans le cas de l'INRIA, la courbe SIZE reste tout de même au-dessus des courbes LRU et FIFO. Il n'est donc pas étonnant que le temps de retrait moyen soit meilleur pour SIZE que pour les autres algorithmes. Par contre, pour DEC, la courbe commence de manière similaire, puis passe en dessous des autres. Cela montre que les petits documents sont obtenus beaucoup plus rapidement, mais que ça se fait au détriment des plus gros documents. La pondération entre ces deux effets n'est pas la même que pour les traces INRIA, ce qui explique pourquoi SIZE a de mauvaises performances sur les traces DEC.

5.4. L'influence de la politique de cohérence

Nous étudions ici les protocoles de maintien de cohérence TTL et Alex. TTL associe à chaque document une durée de vie fixe. À l'expiration de la durée de vie, le document est expulsé du cache. Il existe des variantes de TTL, qui à l'expiration de la durée de vie, préfèrent rafraîchir le document, et le conserver en cache. Pour des raisons techniques, nous ne les simulons pas ici.

Alex associe une durée de vie proportionnelle à l'âge du document. Ainsi, si l'on utilise un coefficient de 50%, un document modifié pour la dernière fois une heure auparavant aura une durée de vie d'une demi-heure, et un document modifié un mois auparavant aura une durée de vie de 15 jours. Cette politique est justifiée par des études statistiques qui démontrent que, si un document a été modifié récemment, il a de fortes chance de l'être à nouveau dans un futur proche. Par contre, un document qui n'a pas été modifié depuis longtemps a peu de chances de changer très bientôt.

Ces deux politiques ont été testées avec un algorithme de remplacement LRU, et un cache de 500 Mo.



(a) Réduction du temps de retrait

(b) Incohérence

Figure 5. *Variation des performances et de la cohérence avec la durée de vie dans TTL*

5.4.1. Politique TTL

La figure 5 montre l'évolution de la cohérence et de la performance avec la durée de vie. On constate que l'incohérence varie fortement avec la durée de vie. Dans le cas INRIA, elle passe de 0 à 1,5% lorsque la durée de vie varie de 0 à 256 heures (10 jours 1/2). Dans le cas DEC, elle augmente également rapidement, mais démarre à une valeur nettement plus importante. Il est probable que cette différence est dûe au mode de récupération des informations de cohérence pour les traces INRIA (§ 4.1.1) : il supprime *de facto* les informations sur les documents qui changent très souvent. Les résultats sur les traces INRIA ne deviennent fiables qu'à partir d'une durée de vie de 24 heures.

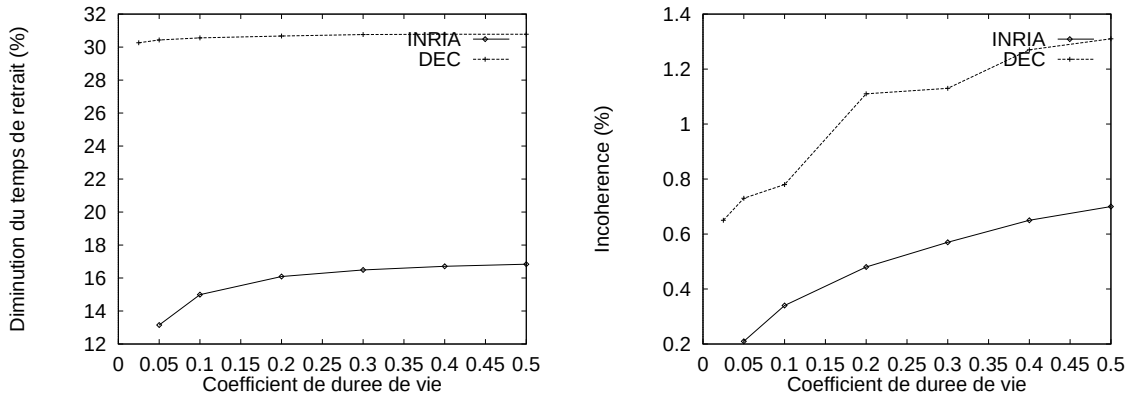
La performance varie également avec la durée de vie. Dans le cas INRIA, elle part presque de zéro lorsque les documents restent très peu de temps dans le cache, jusqu'à un plafond aux alentours de 18%. En effet, une politique TTL trop restrictive ne permet pas aux documents de rester dans le cache suffisamment longtemps pour être redemandés. Les traces DEC montrent le même phénomène, mais limité au tout début de la courbe (entre 0 et 12 heures). Ensuite, la courbe plafonne. Il est probable que dans ce cas, la majorité des documents est expulsée du cache par l'algorithme de remplacement avant l'expiration de la durée de vie. Par contre, les quelques documents qui restent jusqu'à expiration font sensiblement augmenter l'incohérence.

Dans le cas DEC, il est simple de choisir la durée de vie optimale : il s'agit de celle qui maximise la performance, tout en conservant l'incohérence à un niveau relativement faible. Une durée de vie de 12 heures est raisonnable. Par contre, dans le cas INRIA, le choix est beaucoup plus difficile.

5.4.2. Politique Alex

La figure 6 montre l'évolution des performances et cohérences en fonction du coefficient donné à l'algorithme Alex. Il est clair que, plus le coefficient est faible, plus les documents retirés du cache sont cohérents. En effet, un faible coefficient correspond à des durées de vie courtes. La probabilité que le document original soit modifié pendant la durée de séjour du document dans le cache est donc faible.

Cependant, le fait d'expulser très tôt les documents du cache n'est pas sans influence sur la performance. En particulier pour les traces INRIA, le gain de performance passe de 13% à 16,5% entre les coefficients 0,05 et 0,3. Le choix d'un coefficient est donc le résultat d'un compromis entre performance et cohérence. Pour l'INRIA, un compromis raisonnable pourrait être 0,2 ou 0,3 : la performance résultante est proche de



(a) Réduction du temps de retrait

(b) Incohérence

Figure 6. *Variation des performances et de la cohérence avec le coefficient de durée de vie dans Alex*

son maximum, tandis que la cohérence est encore assez bonne.

Les traces DEC sont légèrement différentes. L'incohérence moyenne des documents augmente globalement avec le coefficient de durée de vie. Par contre, la courbe n'est pas aussi régulière que pour les traces INRIA. La performance du cache DEC reste quasiment identique quelque soit le coefficient de durée de vie : le cache étant soumis à un trafic intense, la grande majorité des documents est expulsée du cache avant l'expiration de sa durée de vie. Pour les traces DEC, un bon compromis performance/cohérence pourrait être d'utiliser un coefficient de 0,05 ou 0,1.

5.4.3. Comparaison des politiques

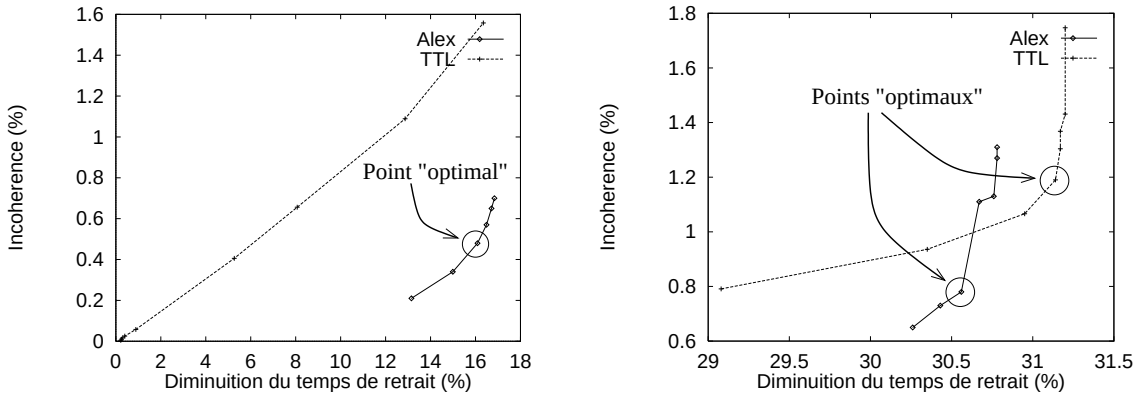
Il n'est pas évident de choisir une politique de maintien de cohérence, ni la bonne configuration de celle-ci. Pour aider à ce choix, la figure 7 reprend les données des figures 5 et 6 en montrant les compromis performance/cohérence atteignables par les différentes configurations de protocoles.

La figure 7(a) montre les compromis atteignables dans le cas des traces INRIA. On voit que la courbe "Alex" est très nettement en dessous et à droite de la courbe "TTL". D'une façon générale, quelque soient leurs configurations, Alex offre donc une meilleure performance et une meilleure cohérence que TTL. Le choix de Alex est donc nettement meilleur que celui de TTL. Pour ce qui est de choisir un compromis performance/cohérence acceptable, le point noté "optimal" sur la figure est un bon candidat : il offre une performance proche du maximum, avec une cohérence raisonnable. Ce point correspond à une politique Alex(0,2). Nous retrouvons donc le point que nous avions sélectionné au paragraphe 5.4.2.

Pour les traces DEC, la comparaison entre TTL et Alex est moins nette : les deux courbes sont sécantes. Le choix d'une politique dépendra des préférences des utilisateurs. S'ils préfèrent de bonnes performances, avec une cohérence moyenne, on choisira le point "idéal" de la courbe Alex ; s'ils préfèrent perdre un peu en performance pour gagner en cohérence, on choisira le point "idéal" de la courbe TTL.

6. Conclusion

Nous avons présenté Saperlipopette!, un outil d'évaluation de caches Web. Cet outil peut être utilisé pour aider un organisme à choisir sa configuration de caches : grâce à Saperlipopette!, le choix d'une infrastructure



(a) Traces INRIA

(b) Traces DEC

Figure 7. *Compromis performance/cohérence atteignables*

de cache peut se faire en connaissance de cause en ce qui concerne les coûts impliqués, ainsi que des bénéfices attendus.

Nous avons utilisé Saperlipopette! pour étudier les différents choix possibles de politiques de cache, ainsi que leur configuration. Les résultats montrent que, pour chaque fonctionnalité, aucune politique n'est optimale dans tous les cas, et que les dimensionnements à mettre en œuvre doivent eux aussi être personnalisés.

L'étude des traces DEC nous a montré un exemple de cache très chargé. Pour atteindre ses performances optimales (en particulier si l'on privilégie la cohérence), il a besoin d'un espace disque de 3 Go, d'un remplacement FIFO et d'un maintien de cohérence Alex(0,1).

Les traces INRIA sont un exemple de cache beaucoup moins chargé. Il atteint ses performances optimales pour une taille de 1,5 Go, un algorithme de remplacement SIZE, et un maintien de cohérence Alex(0,2).

Ces résultats plaident pour l'utilisation d'un outil systématique de recherche de la meilleure configuration, tel Saperlipopette!. Cependant, il n'existe pas aujourd'hui de système de cache dans lequel on puisse mettre en œuvre les choix issus d'une telle étude : ce cache devrait permettre de choisir les politiques internes mises en œuvres au même titre que les divers dimensionnements. À notre connaissance, tous les caches Web existants implémentent le remplacement LRU et la cohérence Alex (ou l'une de ses variantes). Seul le dimensionnement de ces algorithmes est aujourd'hui accessible aux administrateurs. Pour combler ce manque, nous envisageons de développer un système de cache flexible, dans lequel tous les choix de protocoles peuvent être remis en cause pendant l'exécution.

7. BIBLIOGRAPHIE

[Arlitt 96] Martin F. Arlitt et Carey L. Williamson. Trace-driven simulation of document caching strategies for Internet Web servers, septembre 1996. `http://www.cs.usask.ca/faculty/carey/papers/webcaching.ps`.

[Baggio 97] Aline Baggio et Guillaume Pierre. Oléron: supporting information sharing in large-scale mobile environments. Dans *Proceedings of the ERSADS '97 seminar*, Zinal, Switzerland, mars 1997. `http://www-sor.inria.fr/publi/OSISLME_ersads97.html`.

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

[Cao 97] Pei Cao et Sandy Irani. Cost-aware WWW proxy caching algorithms. Dans *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, décembre 1997. `http://www.cs.wisc.edu/~cao/papers/gd-size.ps.Z`.

[Duska 97] Bradley M. Duska, David Marwood, et Michael J. Freeley. The measured access characteristics of World-Wide-Web client proxy caches. Dans *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, décembre 1997. `http://www.cs.ubc.ca/spider/marwood/Projects/SPA/wwwap.ps.gz`.

[Gadde 97a] Syam Gadde, Jeff Chase, et Michael Rabinovich. Directory structures for scalable Internet caches. Rapport Technique CS-1997-18, Duke university, novembre 1997. `ftp://ftp.cs.duke.edu/dist/techreport/1997/1997-18.ps.gz`.

[Gadde 97b] Syam Gadde, Michael Rabinovich, et Jeff Chase. Reduce, reuse, recycle: An approach to building large Internet caches. Dans *Proceedings of the HotOS '97 Workshop*, mai 1997. `http://www.cs.duke.edu/ari/cisi/crisp-recycle.ps`.

[Gwertzman 96] James Gwertzman et Margo Seltzer. World-Wide Web cache consistency. Dans *Proceedings of the 1996 Usenix Technical Conference*, San Diego, CA, janvier 1996. `http://www.eecs.harvard.edu/~vino/web/usenix.196/caching.ps`.

[ICP 94] Internet cache protocol specification 1.4, 1994. `http://excalibur.usc.edu/icpdoc/icp.html`.

[Kim 98] Ilhwan Kim, Heon Y. Yeom, et Joonwon Lee. Analysis of buffer management policies for WWW proxy. Dans *Proceedings of the International Conference on Information Networking (ICOIN-12)*, Tokyo, Japan, janvier 1998. `http://arirang.snu.ac.kr/~yeom/paper/icoin12b.ps`.

[Kroeger 96] Thomas M. Kroeger, Jeff Mogul, et Carlos Maltzahn. Digital's Web proxy traces. `ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.v1.2.html`, 1996.

[Kroeger 97] Thomas M. Kroeger, Darrell D. E. Long, et Jeffrey C. Mogul. Exploring the bounds of Web latency reduction from caching and prefetching. Dans *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems*, Monterey, CA, décembre 1997. `http://csl.cse.ucsc.edu/~tmk/ideal.ps`.

[Makpangou 97] Mesaac Makpangou et Eric Bérenguier. Relais : un protocole de maintien de cohérence de caches Web coopérants. Dans *Proceedings of the NoTeRe '97 symposium*, Pau, France, novembre 1997. `http://www-sor.inria.fr/publi/RPMCCWC_notere97.html`.

[Markatos 96] Evangelos P. Markatos. Main memory caching of Web documents. Dans *Proceedings of the 5th International WWW Conference*, Paris, France, mai 1996. `http://www5conf.inria.fr/fich_html/papers/P1/Overview.html`.

[Melve 97] Ingrid Melve, Lars Slettjord, Henny Bekker, et Ton Verschuren. Building a Web caching system - architectural considerations. Dans *Proceedings of the 8th Joint European Networking Conference*, Edinburgh, Scotland, mai 1997. `http://www.terena.nl/conf/jenc8/papers/121.ps`.

[Neal 96] Donald Neal. The Harvest object cache in New Zealand. Dans *Proceedings of the 5th International WWW Conference*, Paris, France, mai 1996. `http://www5conf.inria.fr/fich_html/papers/P46/Overview.html`.

[Smith 96] Neil G. Smith. The UK national Web cache - the state of the art. Dans *Proceedings of the 5th International WWW Conference*, Paris, France, mai 1996. `http://www5conf.inria.fr/fich_html/papers/P45/Overview.html`.

[Wessels 96] Duane Wessels. Evolution of the NLANR cache hierarchy: Global configuration challenges. `http://www.nlanr.net/Papers/Cache96/`, novembre 1996.

[Williams 96] Stephen Williams, Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, et Edward A. Fox. Removal policies in network caches for World-Wide Web documents. Dans *Procedings of the ACM SIGCOMM '96 Conference*, Stanford University, CA, août 1996. `http://ei.cs.vt.edu/~succeed/96sigcomm/`.

[Worrell 94] Kurt Jeffery Worrell. Invalidation in large scale network objects caches. Master's thesis, Faculty of the graduate school of the University of Colorado, 1994. `ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/WorrellThesis.ps.Z`.

Biographie

Guillaume Pierre est étudiant en thèse à l'université d'Evry-val-d'Essonne et au projet SOR de l'INRIA Rocquencourt. Il s'intéresse aux systèmes flexibles et à l'optimisation des performances de l'Internet. Sa thèse porte sur l'étude de performance et l'optimisation des infrastructures distribuées de caches Web.