Proxy / Serveur HTTP.

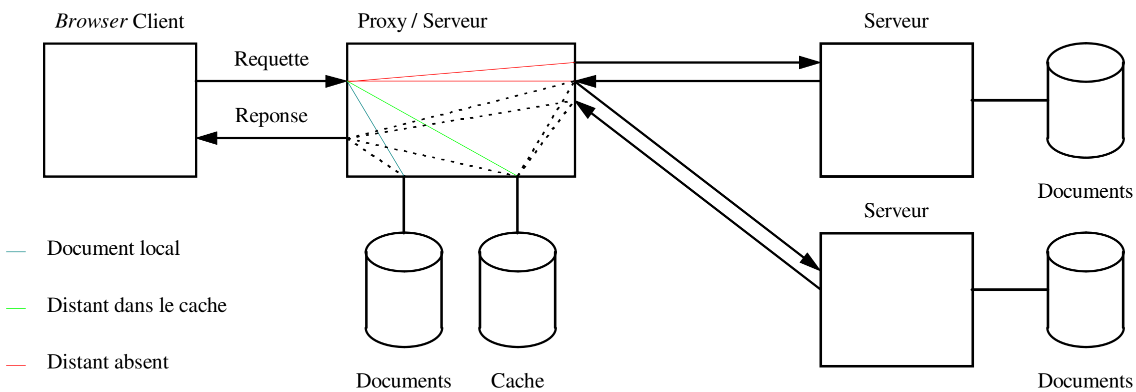
D.E.A Systèmes Informatiques.

Cadinot Philippe, Dorta Neilze.

Mai 1996.

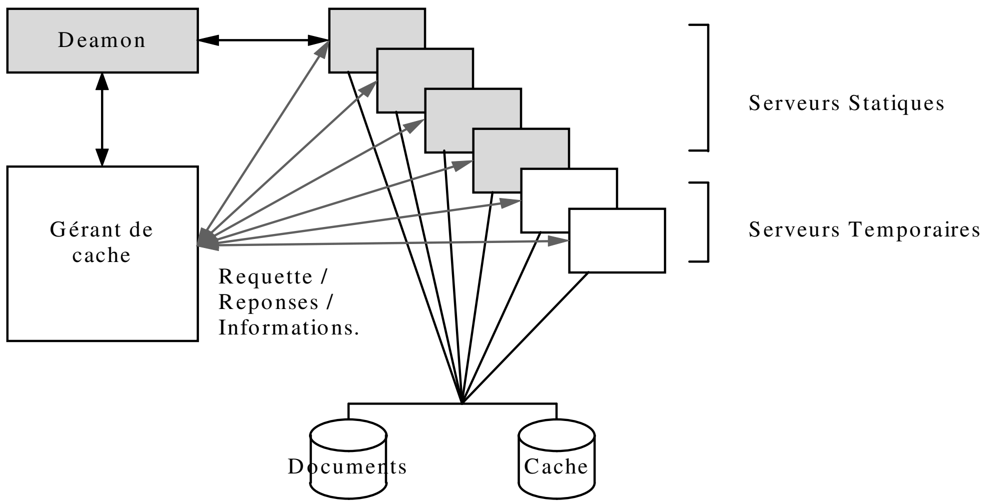
A. Introduction

Le fonctionnement du Wold Wild Web repose sur le protocole HTTP. Ce protocole, constitué de requêtes textuelles, est véhiculé via un mode connecté. Il permet à des *browsers* de visualiser des documents fournis par des serveurs. Les *proxy* sont des caches qui permettent de soulager le réseau et les serveurs en conservant tous les documents qui passent par leur intermédiaire. Ainsi, des requêtes multiples, concernant un même document à travers un même *proxy,* ne nécessitent qu'un seul accès au serveur.



B. Organisation

Nous avons choisi de séparer le Proxy / Serveur en trois entités: le deamon *httpd*, les serveurs et le gérant de cache *cached*.



Nous avons choisi cette configuration de serveurs statiques pour privilégier le temps de réponse des requêtes. Le nombre de serveurs varie entre un minimum (serveurs statiques) et un maximum. Ces serveurs supplémentaires sont temporaires. Ils sont créés lors d'une surcharge des autres serveurs et doivent se terminer au bout d'un certain temps d'inactivité.

Le gérant de cache a pour fonction de contrôler le fonctionnement ainsi que la cohérence du cache. Il gère la correspondance nom fichier / URL.

Le deamon crée initialement le gérant de cache, ainsi que les processus serveurs statiques. Par la suite son rôle est de surveiller les processus serveurs afin d'informer le gérant de leur éventuelle terminaison. C'est lui aussi qui sur ordre du gérant de cache est capable de créer de nouveaux serveurs temporaires.

B.1 Le deamon *httpd*

Ce processus est le père de tous ceux créés par la suite. Son rôle est tout d'abord d'interpréter les arguments qui lui sont passés en ligne de commande. Ensuite il crée le gérant de cache et l'initialise. De même, il engendre les serveurs statiques. Par la suite il a un triple rôle:

- Maintenir le nombre de serveurs statiques: il en génère un nouveau en cas de décès de l'un d'entre eux.
- Créer de nouveau serveurs temporaires sur demande du gérant de cache.
- Informer les gérant de cache de la terminaison des processus serveurs.

Enfin c'est lui qui assure la terminaison correcte, en interrompant tous les processus serveurs et en prévenant le gérant de cache de sauvegarder l'état du cache. C'est lui qui a le plus petit environnement afin de pouvoir crée le plus rapidement possible de nouveau processus serveurs. Lors de cette création les serveurs fils héritent de la socket d'écoute ainsi que des paramètres obtenus en arguments de la ligne de commande.

Les arguments de la ligne de commande sont:

- port: n° de port réservé à la socket d'écoute (80 par défaut).
- data: nom du répertoire ou trouver les documents du serveur local (./data par défaut).
- cache: nom du répertoire ou placer les fichiers constituant le cache (./cache par défaut).

- min: nombre minimum de serveurs -> serveurs statiques (10 par défaut).
- max: nombre maximum de serveurs -> serveurs statiques + temporaires (32 par défaut).
- size: la taille maximale réserve au cache en octet (100.000.000 par défaut).

B.2 Le gérant de cache *cached*

Le gérant de cache à deux fonctions:

- Principalement, il assure le fonctionnement et la cohérence du cache.
- Demander au deamon httpd de créer de nouveaux serveurs temporaires.

Cette deuxième fonction, annexe, est tenue par le gérant de cache car lui seul connaît le nb de requêtes en cours et est susceptible de demander la création d'autres serveurs.

L'implémentation choisie est un serveur RPC. Cela permet de s'abstraire des problèmes réseau et d'étendre rapidement le nb de fonction si nécessaire. Sont fonctionnement repose sur une table de hachage dont la clef d'entrée est l'URL. Cette table de hachage permet lors de la modification de d'une entrée de ne pas avoir à sauvegarder l'état complet de la base mais uniquement ce qui a changé. Les requêtes sont placées dans l'ordre des accès. Les entrées entre elle ne sont pas ordonnées. La sauvegarde de toute la base consiste à créer un fichier par entrée de la table contenant toutes les requêtes liées à cette entrée. Une seconde liste contient toute les requêtes qui doivent êtres évincées. Soit pendant une requête si on s'aperçoit que sa date de péremption est dépassée, soit parcequ'on lui a réservé une entrée mais qu'il n'y a plus de place suffisante dans le cache, soit pendant une phase d'évincement.

Les fonctions du gérant de cache sont les suivantes:

- reset(cache_path,min_serveurs,max_serveurs,size): Appelé par le deamon httpd pour initialiser le gérant et lui passer les paramètres nécessaires a sont fonctionnement.
- naissance(id): Appelé par un serveur pour indique qu'il vient d'être créé.
- deces(id): Appelé par le deamon httpd pour indiquer au gérant de cache la terminaison d'un serveur.
- cache_hit(host,url,access,nocache,id) -> {name,hit/miss/none}: Appelé par un serveur pour demander au gérant si le document est présent, pour l'évincer si nocache et/ou pour avoir une nouvelle entrée si nécessaire. La valeur de retour *none* indique un conflit d'accès le serveur demande un document qui est déjà en cours de récupération par un autre. On lui dit d'aller le chercher tout de même sans le placer dans le cache. *Access (date), host(demandeur)* sont là pour permettre au gérant d'enregistrer la requête dans le *logfile*.
- cache_taille(size,date_modification,id) -> O/N: Appelé par un serveur pour demander au gérant si la place nécessaire est disponible. La date de dernière modification permet de calculer la date de péremption du document: date accès + (date accès - date modif) / 2.
- cache_done(id): Appelé par un serveur pour indiquer au gérant la fin de la requête.

Lorsqu'il n'y a plus de place disponible le gérant de cache commence par éliminer (efface le fichier du cache) toutes les entrées possibles (celle qui ne sont pas encore en cours d'utilisation) placées dans le liste des évincés. Si l'espace disponible est insuffisant (seuil haut), il parcour les entrées de la table de hachage pour placer la plus ancienne dans la liste des évincés jusqu'à avoir une espace potentiellement < au seuil bas. Alors il vide à nouveau toutes les entrée possible de la liste des évincés.

B.3 Les serveurs

Les serveurs restent indéfiniment dans une boucle d'attente d'une requête et de son traitement. Une fois une connexion acceptée par un serveur, il commence par lire la totalité de la requête. Il utilise la librairie "regex" afin de vérifier que celle-ci est correcte. Alors il doit identifier le destinataire de la requête:

- Lui même si le numéro de port et l'adresse destinataire sont bien les siens, ou s'il n'y a pas de champ d'adresse.
- La machine cliente elle même si figure dans la requête l'adresse de rebouclage (127.0.0.1). Alors il répercute l'appel comme distant en remplaçant l'adresse de rebouclage par l'adresse du client (obtenu lors de la connexion).
- Une adresse distante quelconque dans tous les autres cas.

Le traitement d'une requête locale consiste à ouvrir le document pour savoir s'il existe, puis à le retransmettre au client.

Le traitement d'une requête distante demande plus de travail. Il faut tout d'abord s'assurer de la présence du "pragma: no-cache" qui indique que l'on souhaite explicitement passer a travers le cache pour aller chercher la version du serveur. Le serveur demande alors au gérant de cache si le document dont il lui passe l'URL existe et éventuellement l'invalide (pragma no-cache). Dans tous les cas le serveur répond par un nom de fichier: du document s'il est présent ou d'un nouveau fichier pour y placer le nouveau document. Si le document est présent le travail du serveur est identique a une requête locale. Si le document est absent, il faut établir une connexion avec le serveur distant. Si celui-ci répond positivement il commence par transmettre une entête contenant la taille du fichier. Le serveur peut alors demander au gérant de cache si l'espace nécessaire est suffisant. Si c'est le cas, il transfert le document en faisant une copie dans le fichier, sinon le document est simplement transféré au client. Une fois le transfert terminé il l'indique au gérant de cache. Au cours de ces requêtes le serveur passe au client toutes les autres informations nécessaires au gérant pour générer une entrée dans le logfile (adresse demandeur, date requête, date de la dernière modification).

C. Etat d'avancement, améliorations et évolutions

La mise en oeuvre actuel n'effectue pas la sauvegarde de l'état du cache (table de hachage), nécessaire au moment de relancer la machine. De même, la création automatique des serveurs temporaires, selon le besoin, n'a pas été implementé. Toutefois, les structures de données nécessaires à ces fonctions sont présentes.

La configuration implementé présente quelques problèmes qui peuvent être améliorés ou qui exigent des modifications profondes.

Un de ces problèmes est le goulot d'étranglement que représente le gérant du cache qui peut traiter qu'une seule requête RPC à la fois. Pour résoudre cet inconvénient, un changement assez important est nécessaire et ne garanti pas pour autant une meilleure performance. En plus de cela, le gérant du cache n'est pas capable de répondre à une requête et d'exécuter l'évincement en parallèle. Pour que cela soit possible, un soin spécial avec la cohérence du cache est impératif. Ces modifications doivent être suivie d'une étude plus approfondie même si elles sont particulièrement intéressantes dans le cas d'un serveur distribué.

Autre amélioration possible est de permettre un accès concurrent (sémaphore) sur la table de hachage du cache. Chaque serveur peut alors travailler sur une entrée différente. Cela suggère un processus à part, réveillé par le serveur, qui gérerait l'évincement.

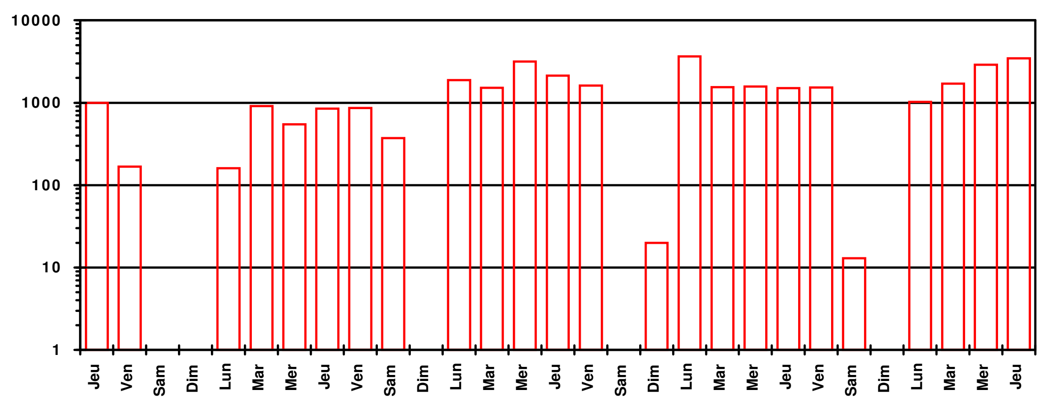
Assez important aussi est le problème de la taille du cache, particulièrement l'accès concurrent à sa variable. Dans le cadre d'un serveur distribué, la taille est naturellement répartie entre les différents serveurs.

D. Statistiques

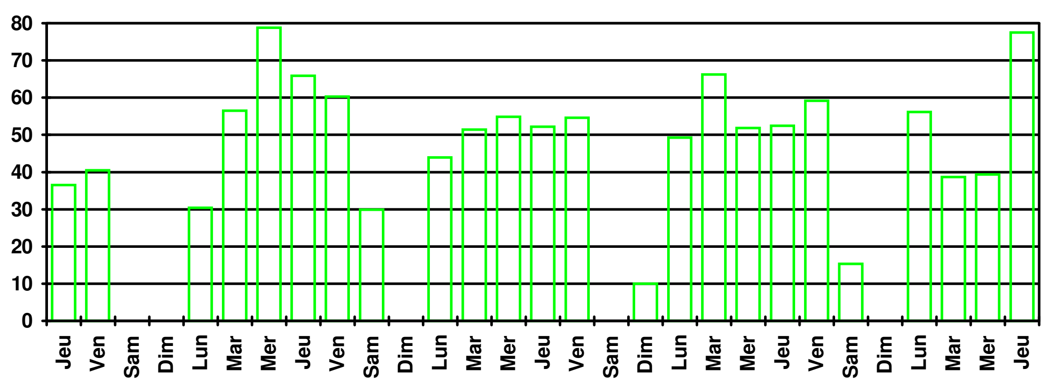
Nous avons conduit trois types de statistiques. La première donne le nombre de requête et le taux de *hit* pour différentes unités de temps. Ensuite, nous avons étudié ces deux mêmes critères en fonction de la taille des fichiers. Enfin nous abordons le problème de la taille du cache.

D.1 NB de requêtes / Taux de Hit par Mois / Semaine / heure

Cette première série de mesures a pour objet de dimensionner les serveurs. Il manque cependant la durée moyenne des requêtes afin de connaître le nombre de serveurs statique nécessaire.

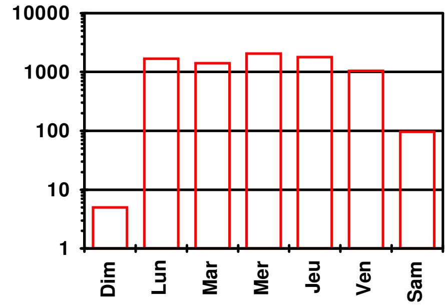


NB de requêtes par jours, sur un mois.

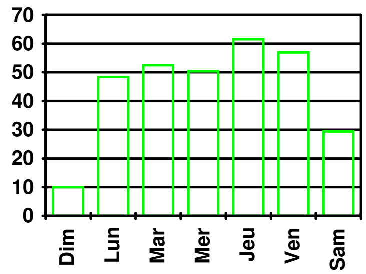


Taux de *Hit* (en %) par jours, sur un mois.

De ces deux premiers histogrammes il apparaît que la presque totalité des requêtes ont lieu les jours ouvrables. La demande semble croître régulièrement mais il manque des informations sur les mois précédents pour avoir si la tendance est réelle. Il n'y a pas non plus de corrélation entre le nombre de requête et le taux de *hit*.

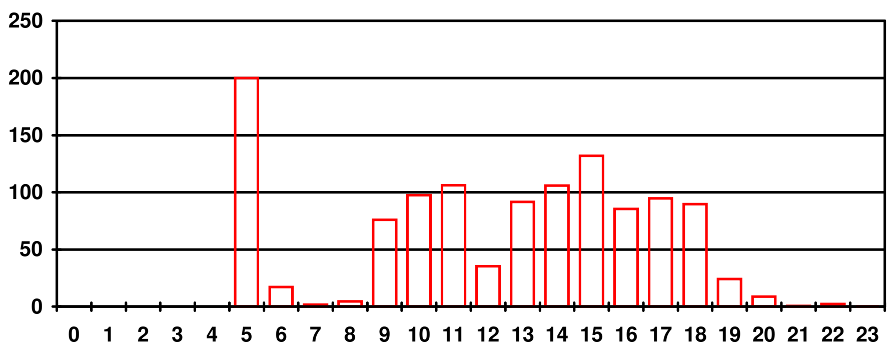


Moyenne sur un mois du NB de requêtes par jours de la semaine.

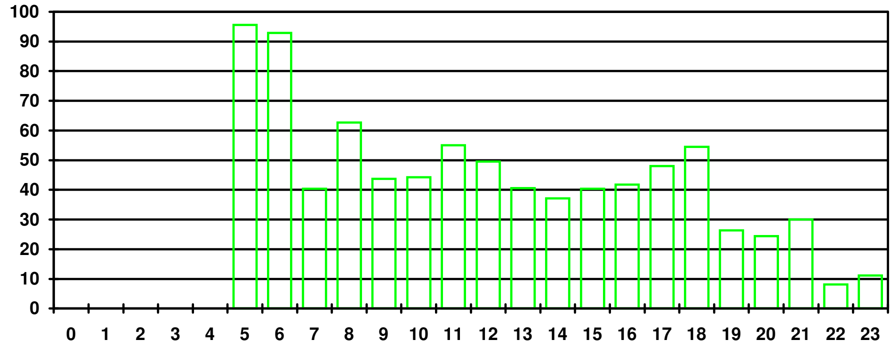


Moyenne sur un mois du taux de *Hit* (en %) par jours de la semaine.

Sur ces deux nouveaux histogrammes il apparaît clairement que toutes les requêtes ont lieu la semaine, avec un taux de *hit* en général supérieur à 50%. Par contre les requêtes faites les samedis et dimanches concernent des documents qui ne sont pas généralement accédés. Il n'apparaît toujours pas de correspondance entre le nombre de requête et le taux de *hit*.



Moyenne sur un mois du NB de requêtes par heure, sur 24 heures



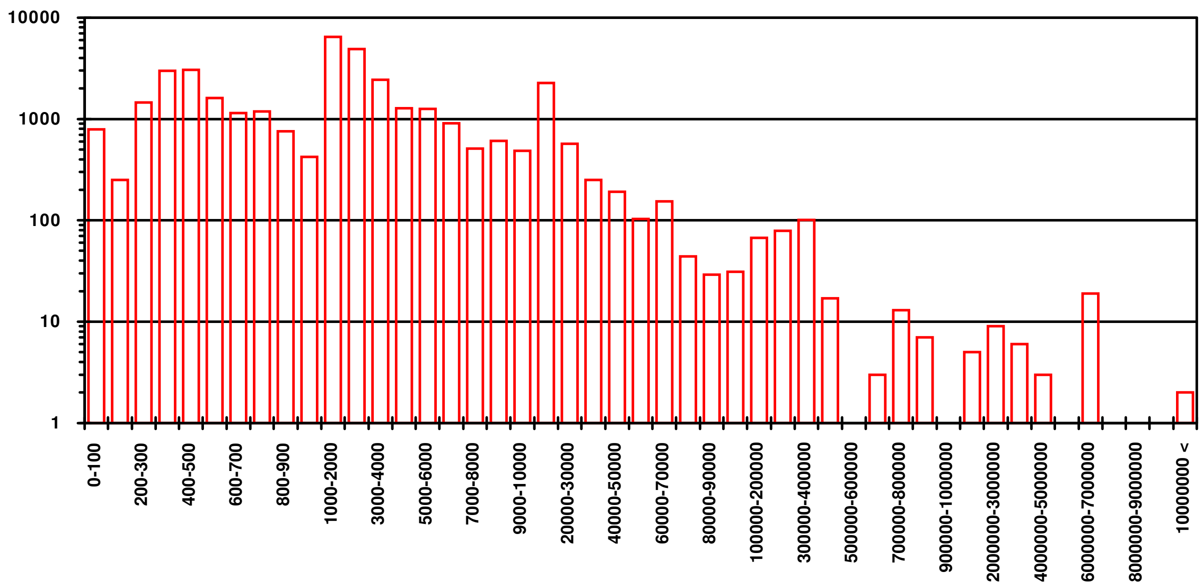
Moyenne sur un mois du taux de *Hit* (en %) par heure, sur 24 heures.

Ces deux histogrammes confirment que la plupart des requêtes sont faites pendant les heures de travail. Qu'il n'y a pas de lien entre le nombre de requêtes et le taux de hit. Excepté pour les requêtes faites hors des heures de bureau qui concerne de nouveau document plus rarement présent dans le cache. Mais surtout pour les requêtes faites entre 5 et 6 heures qui sont de demande de vérification de lien et qui fausses les statistiques.

Globalement nous avons une moyenne de 50% de taux de *hit* et moins de 150 requêtes/Heure (week-end compris).

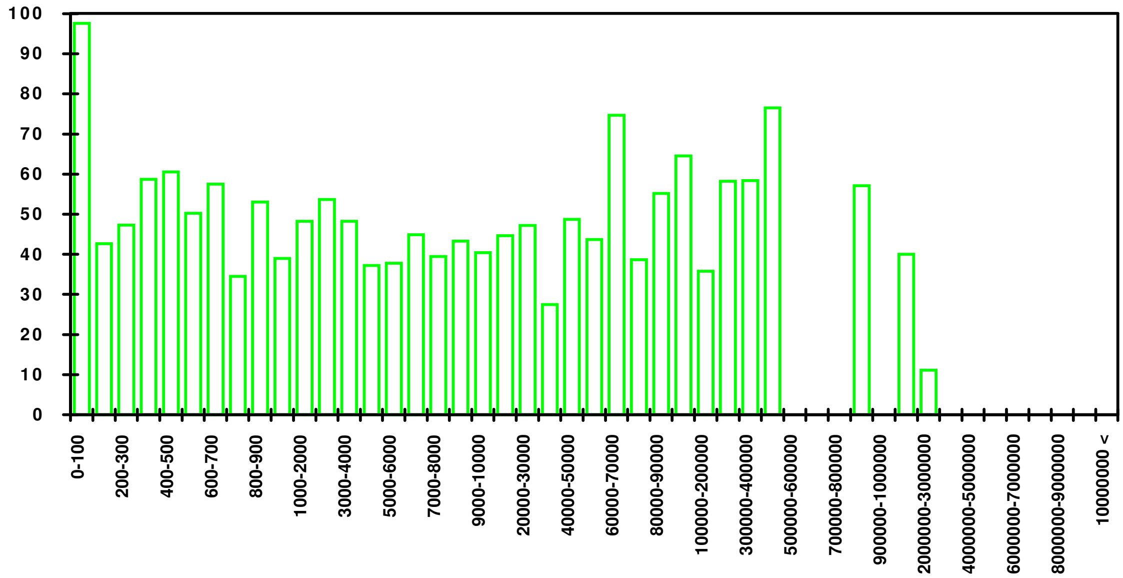
D.2 NB de requête et Taux de Hit par taille de fichier

Cette deuxième série de mesures entre nombre de requête, taux de hit, et taille de fichier.



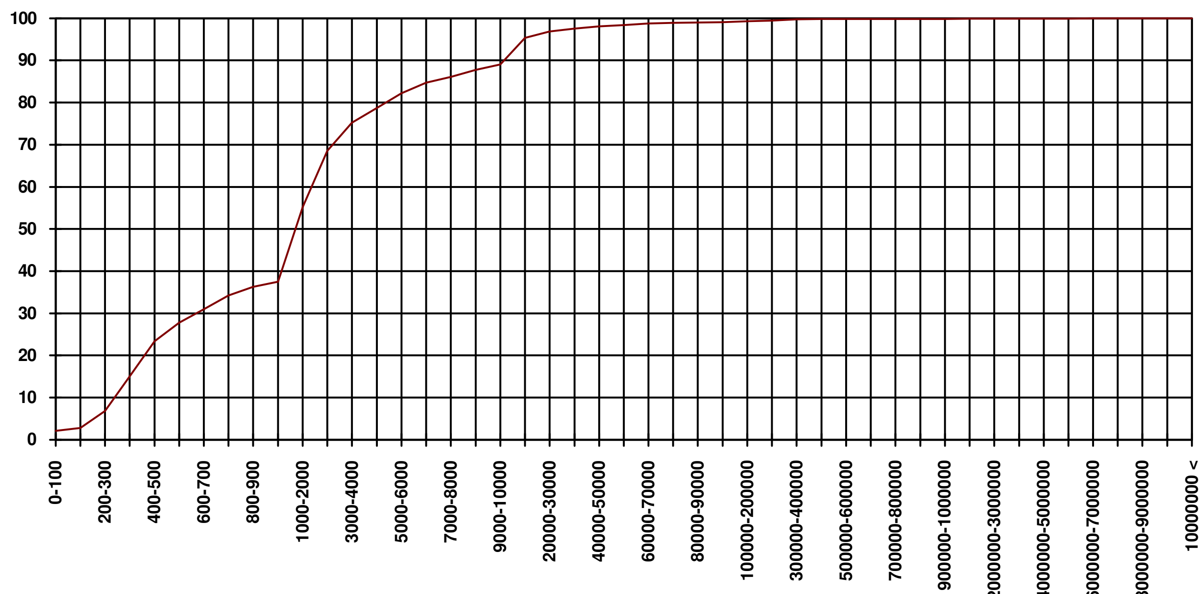
NB de requête par taille de fichier (en octets).

Le nombre de requêtes dépend fortement de la taille des fichiers. La plupart des requêtes concernent de petits documents: textes < quelques k octets, ou des images < qlq 10k octets, les requêtes les plus grosses concernent des fichiers d'archives transférés par FTP et utilisés une seule fois.



Taux de *HIT* (en %) de requête par taille de fichier (en octets).

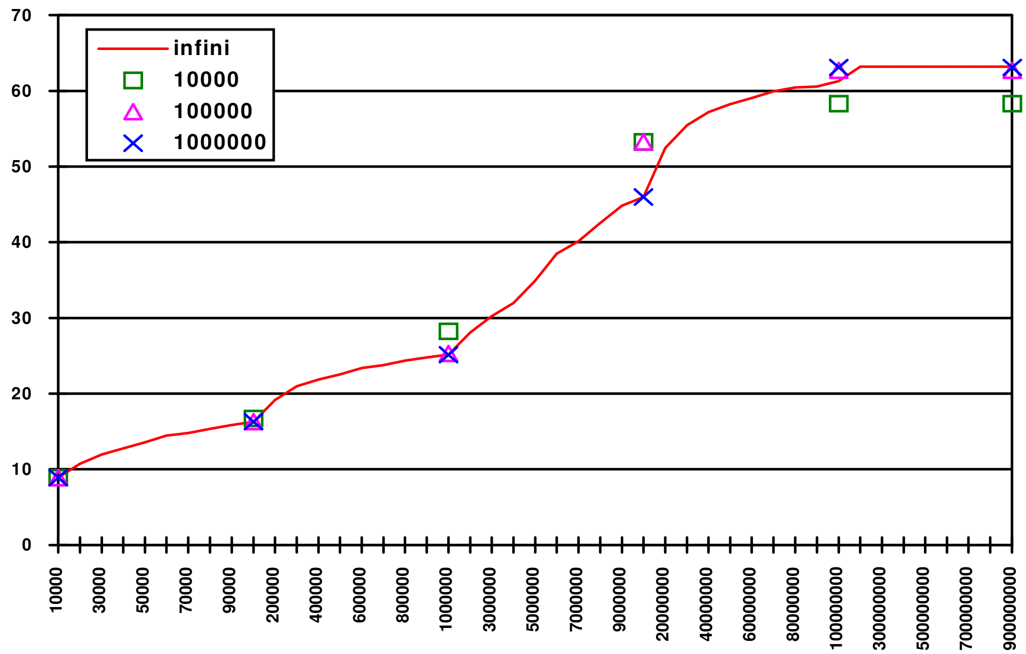
Le taux de hit apparaît clairement en relation avec la taille des fichiers. Le taux en d'environ 50% pour la plupart des documents d'une taille inférieure à 60ko. Les exceptions concernent tout d'abord les requêtes de vérification de lien (taille < 100) et qlq documents > 80ko.



% cumulé du nombre de requêtes par taille de fichier (en octets).

Sur cette courbe il apparaît clairement que 90% des requêtes concernent de fichier dont la taille est inférieure à 10ko et 99,88% pour une taille < 100ko !

D.3 Taux de Hit en fonction de la taille du cache



Taux de *HIT* (en %) en fonction de la taille du cache (en octet).

La courbe en rouge décrie l'évolution du taux de *hit* en fonction de la taille du cache. Elle a été obtenue en rejouant non seulement les requêtes du mois de février mais aussi en réexécutant auparavant les requêtes qui étaient déjà présente dans le cache (*hit* dès la première requête). Le maximum atteint correspond à toutes les requêtes présentes dans le cache. Les autres points sont obtenus en limitant la taille des fichiers que l'on autorise dans le cache. L'amélioration est assez significative puisque l'on obtient un taux de hit équivalent pour une taille de cache deux fois plus petite.

Toutes ces mesures sont cependant insuffisantes. Elles ont été faites sur une durée trop courte et sans avoir assez de connaissance sur les habitudes des utilisateurs. Plus précisément, il manque la durée moyenne des requêtes pour permettre de dimensionner le nombre de serveurs statiques nécessaires. Il manque les mois suivants et précédents pour appréhender correctement les habitudes des utilisateurs et l'évolution des demandes. Je pense qu'il en nécessaire d'établir des mesures supplémentaires. En rejouant sur une plus longue durée des requêtes avec toute la précision possible: date exacte de début de requête, durée exacte de la requête, fonction de coût réseau/disque/système, traitement simultané des requêtes qui sont réellement en parallèle... Les conclusions à en tirer sont un dimensionnement précis des serveurs, une évaluation des choix: vaut-il mieux éliminer les grosses requête au risque de devoir les refaire par la suite et ainsi surcharger le réseau. Le coût d'établissement des requêtes est il significatif...

E. Conclusion

Pendant toute la durée du projet, des choix ont été faits entre la performance, la simplicité et l'extensibilité. Dans ce sens, une configuration de base a été retenue qui implemente les points cités avant. Toutefois, on remarque quelques points négatifs ainsi que des extensions possibles, déjà abordés, dans le cadre de ce projet.

Ce projet nous a, entre autres, fait prendre conscience de la complexité du problème du cache, en ce qui concerne la cohérence et la politique d'évincement. En plus, quelque soit la politique adopté, la manière de gérer le problème de façon plus optimale est fonction de plusieurs facteurs, ce qui nous confronte à de nombreuses possibilités.

Une étude plus approfondie sur le comportement des requêtes et sur les performances du gérant du cache est donc indispensable pour avancer dans le problème et proposer des solutions intéressantes.