On-line Measurement of Web Proxy Cache Efficiency

Simon Patarin, Mesaac Makpangou

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

On-line Measurement of Web Proxy Cache Efficiency

Simon Patarin and Mesaac Makpangou

N° 4782

Mars 2003

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4782--FR+ENG

On-line Measurement of Web Proxy Cache Efficiency

Simon Patarin* and Mesaac Makpangou†

Thème 1 — Réseaux et systèmes
Projet REGAL

Rapport de recherche n° 4782 — Mars 2003 — 29 pages

Abstract: This report presents how Pandora, our flexible monitoring platform, can be used to continuously measure the efficiency of a system of cooperating proxy caches. It circumvents many of the drawbacks of existing tools: Pandora integrates all stages involved in the evaluation process, it operates in real-time, it does not depend on specific cache software, and it can be adapted to any specific system configuration. We detail how this can be achieved using the flexibility offered by Pandora. We also present two experiments that illustrate the utilisation of these techniques: the first one evaluates the proxy cache deployed at INRIA Rocquencourt, the second one measures the efficiency of cooperating caches in an artificial environment. Finally, we describe how we plan to integrate these measurements inside an auto-adaptative Web proxy cache.

Key-words: network monitoring, Web proxy cache, measurement, efficiency

* Simon.Patarin@inria.fr
† Mesaac.Makpangou@inria.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 — Télécopie : +33 1 39 63 53 30

Mesure en continu de l'efficacité des caches Web

Résumé : Ce rapport présente comment Pandora, notre plateforme de surveillance flexible, peut être utilisée pour mesurer l'efficacité d'un système de caches Web coopérants. Cette méthode countourne grand nombre d'inconvénients présents dans les outils existants: Pandora travaille en continu, ne dépend pas de logciel de cache spécifique et peut être adapté à toute configuration système. Nous détaillons comment ceci peut être atteint grâce à la flexibilité offerte par Pandora. Nous présentons également deux applications qui illustrent l'utilisation de ces techniques: la première évalue le cache déployé à l'INRIA Rocquencourt, la seconde mesure l'efficacité de caches coopérants dans un environnement artificiel. Enfin, nous décrivons comment nous projetons d'intégrer ces mesures au sein d'un cache Web auto-adaptatif.

Mots-clés : surveillance réseau, cache Web, mesure, efficacité

1 Introduction

Web proxy caches are deployed almost everywhere, at organisation boundaries as well as at ISPs. Proxy caches improve latency when retrieving Web documents while reducing the overall network traffic on the Internet. Proxy cache administrators spend some effort configuring the caching software and determining their suitable placements in order to obtain the best achievable quality of service, with respect to user demands. However, despite this effort, the efficiency of caches may decline over time. Indeed, the efficiency of a given caching infrastructure depends heavily on the actual use of the caches: number of users, type of requested documents, load of machines running caches. Since these characteristics are subject to change, it is unlikely that any particular configuration and deployment will remain effective during the whole cache life cycle. Therefore, to maximise the benefits offered by Web proxy caches, administrators should reconfigure the existing infrastructure as the traffic characteristics evolve; this implies frequent *measurements* of cache *efficiency*.

This raises two distinct issues. First, how often should one perform measurements? With respect to the frequency of evaluations, it is important to be able to react to sudden modifications (flash crowds, network or servers failures), as well as slower ones with strong trends (growing user community, increasing use of multimedia documents). The appropriate frequency depends mainly on the impact of the cache's efficiency on the actual quality of service. Hence the best choice of frequency is particular to each system. Each administrator should be given a way to adapt the measurement frequency of his caches, taking into account the objectives in terms of overall service improvement and the effect of the degradation of performance. Second, how to cope with the inherent subjectivity of the notion of efficiency? The suitable efficiency metric depends on the goals defined by the cache administrators; the same level of performance may lead to different experiences by different people in different environments. It must be customisable. Examples of basic metrics that can be used include the consistency of the returned documents, bandwidth savings and latency savings.

Regardless of which metric is considered, the measurement of cache efficiency requires three distinct steps. First, one needs traces recording the behaviour of the system of caches. Two different methods can be used to obtain

these traces. The obvious one is to use log files that all caching software generates. However, this technique is overly simple because those logs often lack information and, even worse, sometimes contain inaccurate information [4]. To circumvent these drawbacks, one needs to instrument the software; however, this is not always possible: source code may not be available, or it may require a large amount of work. Another approach to build such traces is to use passive traffic monitoring. Several tools performing HTTP traffic extraction from raw network packets have been proposed in the past. These include BLT [5] and Windmill [11], which can collect information pertinent to the measurement of all necessary metrics. However, none of them consider the biases introduced by caches. Hence, the traffic characteristics they would capture in presence of Web proxy caches are corrupted, leading to inaccurate measures of cache efficiency. The second stage consists of analysing the traces to extract suitable basic metrics. This is commonly achieved by cache log analyser software such as Calamaris [1] or Squeezer [8]. They produce statistical summaries of the given log file in plain text or HTML format. Other packages like WebLog [14] provide software components (Python classes for WebLog) to manipulate logs and perform their own analysis. The final step combines the basic metrics previously extracted to produce the actual efficiency measurement. Trace-driven simulation tools [21, 7, 28] are widely used.[1] However, these tools are usually dedicated to the evaluation of a limited number of metrics. This makes it difficult to extend them in order to take into account metrics that were not thought of beforehand. Furthermore, it is difficult to perform such simulations in real-time, which does not fit our design goals.

What stems from this quick review of existing tools related to cache evaluation is that there does not exist a single tool that integrates all of these three stages. Consequently, it is often necessary to have glue software inserted between two consecutive steps. This also requires each piece of software to be executed one after the other, complicating their use on a continuous (and automated) basis. Finally, the diversity of tools used makes it difficult to have them deployed easily on a wide range of platforms and environments.

In this report we present how Pandora [15], our flexible monitoring platform, can be used to build a well-integrated tool for on-line efficiency measurements of proxy caches, thanks to its component-based architecture and its

[1] Such software also usually integrates also base metric extraction step.

intrinsic flexibility. It allows the collection unbiased metrics for an arbitrary complex system of caches. Pandora uses passive traffic capture and on-line HTTP trace extraction to monitor the behaviour of the caching system. To correct the modifications introduced by proxies on the traffic, it performs active probes. Then, this trace is analysed on the fly to compute the various base metrics it uses to evaluate the efficiency of the system.

The rest of the report is organised as follows: first, Section 2 gives a rapid overview of the architecture of Pandora. Section 3 shows how Pandora can monitor a system of proxy caches. Section 4 presents the metrics used to characterise caching systems. Next, Section 5 describes two experiments, respectively in both real and artificial environments, that make use of these measurements. Section 6 presents some related work concerning the various techniques and tools developed to monitor and evaluate proxy caches. Finally, Section 7 concludes this report and presents our future work with Pandora.

2 Pandora

Pandora [15] is a general purpose monitoring platform. It offers a high level of flexibility while still achieving good performance. In this section we present briefly the architecture of Pandora and its main characteristics.

Each monitoring task executed by Pandora is split into basic and self-contained building blocks called *components*. These components are chained inside *stacks* to constitute high level tasks. Stack execution consists of components exchanging messages (data structures called "packets") from the beginning of the stack, to the end.

Pandora provides a framework dealing with (among others) packet demultiplexing, timers, threads and communication. This allows programmers to concentrate on the precise functionalities they want to implement and promotes code reuse. During stack execution, components are created as necessary and the resources they use are collected after some — user-specified — time of inactivity.

Pandora may be configured in two different (and complementary) ways. First, at run time, Pandora reads static configuration files either from disk or from the network. Second, if told so, Pandora opens a control socket to

which commands can be sent.[2] These commands allow queries of the actual configuration of the platform and to perform arbitrary modifications on it. These modifications impact also the stacks being executed. Configuration itself includes stack definitions and component library localisation. A stack definition specifies the exact chaining of components while the localisation of a component tells Pandora which library to load in order to create the component.

A single Pandora process is able to execute several stacks concurrently. Furthermore, an unique logical stack may be split into several substacks. These substacks may be run either within distinct threads inside the same process or within different processes, possibly on distinct hosts. Indeed, Pandora provides communication components that allow such stack connections.

Concerning performance, experiments show that the overhead related to component chaining is limited to 75 ns per component and per packet, on a 1 GHz Pentium III processor. Pandora also achieves complete HTTP extraction at a sustained 300 Mb/s rate, when reading a packet dump from a file.

3 Monitoring Proxy Caches

Our measurements are based on metrics extracted from HTTP traffic traces. These metrics include latency and round-trip time improvements, bandwidth savings, documents consistency and number of bytes used for cooperation. Before explaining precisely (in Section 4) how these metrics are used to compute a measure of cache efficiency, we present in this section how Pandora captures them. First, Section 3.1 considers the case of isolated proxy caches, then Section 3.2 shows the extra steps necessary to handle the cooperation between caches.

[2] Pandora provides an API with C++, C and Guile [23] bindings to ease the construction of clients.

3.1 Isolated Caches

To evaluate caching system efficiency, one needs to observe traffic both before and after each cache in the system. The comparison of the traffic observed around the caches lets us determine the biases they introduce in the traffic.

Therefore, one needs a stack located before the cache to collect a trace of the traffic between the clients and the cache (called the *client trace*) and another one located after the cache to collect the traffic between the caches and the Web servers (*server trace*). These two traces are sent to a third analysis stack that processes them. This last analysis step consists of detecting biases, correcting them (querying from the originating HTTP servers for any information missing from the traces) and then producing the final unbiased trace as explained below. The exact number of instances of Pandora needed to execute these different stacks depends on the actual network configuration. Whenever this is achievable, running all of these on a single machine seeing all the traffic (e.g. by being connected directly to a border router) is the easiest solution. However, it is always possible to run each of them on distinct machines, or even to use several instances of the same stack (capturing different parts of the traffic).

HTTP extraction has been described previously [15], and we focus on the analysis stage that matches HTTP transactions stemming from both traces. To perform this, we first merge the transactions seen between clients and the proxy cache and those emitted by the cache itself into a single flow. At this point, a classifying component examines each request to determine whether it has been made from a client to a cache or from a cache to a server. We call them respectively *client* and *server* transactions. The precise information the classifier looks at is the format of the URL (requests made to a cache should start with `http://server/`, whereas those directed to servers should start with `/` and use the `Host` header), the `Via` and `X-Forwarded-For` headers. The component looks also at the destination port of the request (3128 or 8080 are assumed to be proxy ports and 80 server ports) to confirm the previous checks. Additionally, it maintains a set of known proxy cache IP addresses (as determined by the above checks) so that future requests to or from this cache are immediately identified. Since the number of proxies in the system is likely to be very small, this reduces drastically the time spent in this stage.

Transactions are then demultiplexed according to their URL. If we find two transactions for the same URL, one *client* and one *server*, it means that the document was not present in the cache (since a request has been sent to the Internet) and we just need to generate a record merging the information collected from both the client and the server trace. In other words, it means rewriting the client transaction by replacing the cache address with the Web server address and adjusting the other parameters to reflect what would have been the request without the cache. Respectively, if we hold a *client* transaction and no matching *server* transaction is found (after a specified timeout) then we assume that the cache made a hit.[3] In the case of a cache hit, we still lack the real characteristics of the request. In order to circumvent this, Pandora sends standard requests to original servers for those documents. If such additional requests are sent, they are extracted eventually by the HTTP monitors and follow the standard path up to the matching component. This component, thanks to a private HTTP header, recognises generated transactions as such, and uses them to produce records like those in the cache-miss case. There may be cases where several clients request the same URL from the same cache at approximately the same time. In such cases, we make use of an heuristic based on request timestamps to match the different transactions.

If there are several isolated proxy caches, two methods may be used. Either the stacks described above are deployed around every cache and the output of each analysis stack is merged into an single "unifying" stack. This merge is done by using the Pandora's general purpose network reader and writer components. Otherwise, if vantage points exist that can capture the traffic for all caches, then one can use the very same stacks since the matching component will use of IP addresses of the caches to distinguish the requests made to each one of them. Both solutions are equivalent (the same trace will be produced) and the choice depends on the actual configuration of the targeted network.

3.2 Cooperating Caches

When several proxy caches are used within a single organisation, they are often configured to cooperate with each other. This means that when a cache

[3] We need to use a timeout since packet losses and clock skews might alter the original request sequencing.

misses a document it will first try to fetch it from one of the other caches before sending a request to the server on the Internet. Caches are organised hierarchically: all caches at the same level are called *siblings* and may be connected to a higher level *parent* cache. Top level caches are connected directly to the Internet. Cooperation involves an inter-cache protocol to let caches know which documents are available from their peers. We consider here the Internet Cache Protocol [26], which is one of the most widely used. ICP is a simple transactional protocol over UDP.

We need to parse ICP messages, extracted from UDP packets, and match together the corresponding queries and responses. In order to do so, we first demultiplex UDP packets according to their "connection" (same pair of source and destination IP addresses, in indistinct order). Next, ICP messages are extracted and then demultiplexed according to their unique identifier. Then, all messages related to the same query (and no others) are passed to a single "matching" component, whose work is to build a record describing this event. The metrics we are interested in are: the latency of the request, the number of bytes transferred, and of course the status of the request (peer's hit or miss).

We have seen in the case of isolated proxy caches that a single client request may generate either one (if it is a hit) or two HTTP transactions (if it is a miss). Here, the total number of transactions is still at least one, but may be at most $n+1$, where n is the number of levels in the hierarchy. Indeed, a request is forwarded for each level in the hierarchy if no cache holds the document. This counts for $n-1$ "internal" requests. Additionally we must take into account the request made from the client, plus the last request which may be directed to the original web server or to a peer. All these transactions (for the same URL) go through the classifying component seen previously. Now, there is another category (in addition to the *client* and the *server* ones) corresponding to requests made between peer caches, that we call *peer* transactions.

Then after demultiplexing according to the URL, the matching component tries to build chains made of a *client* transaction followed by any number of *peer* transactions and a last *server* transaction. These chains are made according to IP addresses matching and the natural timestamp heuristic (assuming that a cache cannot forward a request before it has received it). If we build a complete chain, it means that the requests made a miss, otherwise the length of the chain will tell whether the hit was a plain or a peer hit.

Finally, ICP and cache transaction records are merged into a single component that extracts the suitable metrics from both types of events. This produces an unified format event containing the values of the measured metrics. Such events are forwarded to the evaluation stack described in the next section.

4 Efficiency Measurement

This section discusses the method we use to compute the representation of the efficiency achieved by a system of caches based on the traces produced by the previous stage. First we describe the metrics we have used then we show how these may be consolidated.

4.1 Metrics

As soon as the HTTP traffic trace is collected, it is used to perform the measurement of the efficiency of the system of proxy caches. To this end we define a set of metrics that maps their input (cache or ICP transactions) into unitless decimal numbers. Negative values indicate harmful configurations (i.e., those that actually *degrade* efficiency), positive values denote improved quality of service and 0 corresponds to a situation where caches were not present (which should be the expected minimum of the function in real conditions).

Round-Trip Time Improvement Round-trip time is the time needed for a request to be completed, from the *first* byte of the request until the *last* byte of the response. To compute the metric, we consider the difference between the round-trip time for the *server* transaction and the *client* transaction. In the case of a miss, we expect this to be lightly negative, whereas it should be largely positive in case of hit. Then we compute the ratio of this time with the round-trip time of the *server* transaction. Hence the value computed is equal to the percentage of improvement (or degradation) compared to the situation without the proxy.

Latency Improvement Latency is the time spent between the *last* byte of the request and the *first* byte of the response. This corresponds to the time a user will have to wait before seeing anything in her Web browser (assuming that the browser starts displaying data as soon as it receives it). It captures both the network latency and the latency of both servers: the Web server and the proxy cache. This metric is computed like the previous one by considering the percentage of improvement compared to the no-proxy case.

Bandwidth savings This is simply the number of bytes transferred for each transaction, including the headers of the request and the response. The same percentage is computed as for the above metrics, by making the ratio between the number of bytes saved (or wasted) by the proxy and the total number of bytes for the transaction without the proxy. One must note that proxy caches usually add a few headers to the requests they emit. Also, they may transform, in some circumstances, plain GET requests into "if-modified-since" ones.[4]

Consistency This metric aims at capturing the freshness of the documents returned by the cache. Indeed when a cache returns a document it holds, there is no warranty that it has not been updated on the original server. Usually Web servers include in their response a `Last-Modified` header giving the most recent modification time of the document. When determining if a document is stale, three timestamps are to be considered: t_p, the last modified timestamp of the document returned by the proxy, t_o the last modified timestamp of the original document and t_r the timestamp at which the request was made. If $t_p = t_o$, the document is consistent. If $t_p < t_o < t_r$ the document returned by the cache is stale. If $t_p < t_r < t_o$ we cannot conclude anything. The last case may happen because we cannot ask the server precisely at the time the client request was done: there is always a small delay (necessary to decide whether the lack of transaction between the proxy and the server is really a hit). By reducing the delay, we reduce the probability of this case, but we

[4]These requests are emitted by caches that want to check if the document they hold is still valid. They add a header looking like: `If-Modified-Since: DATE` where `DATE` is the timestamp of the last modification of the document. Upon reception of such requests, the server sends a response with a `304 Not Modified` status code if the document has not been modified or sends a standard `200 OK` response along with the new version of the document.

cannot eliminate it completely.[5] Concerning this metric, we compute the ratio of stale documents with the total number of hits.

Cooperation Efficiency When proxies are cooperating, they necessarily exchange control information in order to know each other's cache content. To capture the efficiency of this cooperation, we compute the ratio of the number of document bytes fetched between peers with the number of control bytes.

4.2 Consolidation

In order to smooth measurement variations, instantaneous evaluations (performed each time a packet is received) are consolidated. There exist two consolidation components in Pandora: with or without a notion of history. Both components use a customisable consolidation function: it is specified as a symbol loaded from a dynamic library. Common functions are provided by Pandora (including mean, sum, maximum, minimum and last value) but one can easily use another one by simply programming it in C. The component without history is used to synthesize many individual measurements into one, representative for a (usually short) specified period of time. The other component holds past measurements in a fixed-size buffer and evaluates its function over the whole buffer. This permits having a parameterisable window width, allowing measurements covering large time scales to be updated more frequently.

The values produced by the consolidation components can be displayed as they are computed (e.g., for plotting) or can be used by a mechanism of alarms which are triggered each time these values, either absolutely or relatively, exceed specified thresholds. Any other exploitation of the results may be considered: it only requires the implementation of the necessary components; such more interesting applications are under development and are described in Section 7.

[5] Currently, the delay used by Pandora is less than two minutes.

5 Experiments

In this section, we show examples of Pandora's use as presented above. The first experiment was conducted at INRIA Rocquencourt during the month of July 1999 and evaluates the behaviour of its isolated proxy cache. The second one is a synthetic test run on a LAN to demonstrate the capabilities of our tool concerning cooperating caches.

5.1 INRIA Proxy Cache Evaluation

distinct users	353
distinct servers	9618
total documents	535643

Table 1: Characteristics of the INRIA cache trace.

INRIA Rocquencourt is connected to the Internet through a single border router which sees all outgoing and incoming traffic. For Web access, users have the choice to use a proxy cache or not.

In this analysis we are only interested in the traffic going through the cache and we focus on a one week long trace, which corresponds to a usual working week at INRIA. The characteristics of this trace are presented in Table 1.

Pandora was configured as described in Section 3.1. Figure 1 shows the results of the efficiency measurements based on this trace. For reasons explained below, we have chosen not to use the latency metric for this evaluation. Furthermore, we have chosen a time-aggregation factor of one hour together with a 12 hour large window. Such parameters allow us to consider a sufficiently large number of events, such that the measurements remain significant, and they still outline trends in cache efficiency evolution occuring within a single day of operation.

Efficiency measurements of the INRIA proxy cache are not encouraging. Indeed, at the time the experiment was run, the proxy cache was still not perfectly configured and only a small fraction of outgoing requests (less than one third) were seen by the cache. Furthermore, INRIA's good network connectivity makes the configuration of a Web proxy cache rather delicate. Further

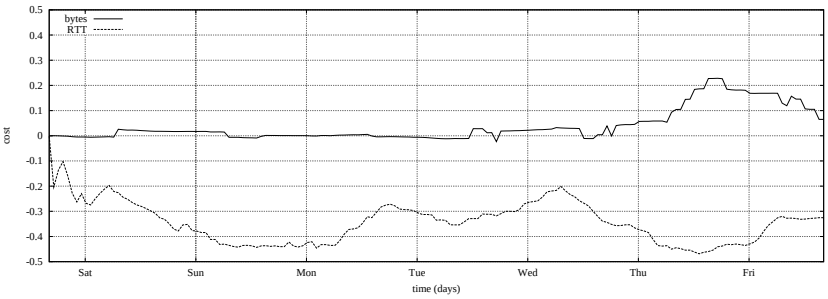


Figure 1: Evaluation of the INRIA proxy cache from July 07 to July 13 1999

investigation explains why negative values are observed in both metrics. Concerning bandwidth savings, as we noticed in Section 4.1, the proxy systematically adds two HTTP headers (`Via` and `X-Forwarded-For`) to the original request which counts for about 60 bytes (which becomes not negligible for small documents). Furthermore, the cache always finishes downloading documents whose retrieval has been interrupted by the client, which is likely to happen with large documents. The low hit rate does not allow compensation for these losses. Concerning round-trip time, some delay is accountable to the cache's processing of requests, but the main overhead is related to name resolution: with a standard request, as made by a Web browser without proxy, when the first packet is seen by Pandora name resolution has already occurred and simple HTTP monitoring cannot tell how long ago the user typed the new URL. In the case of requests made through a cache, the resolution of the cache's name is immediate and it is up to the cache to make the actual resolution of the server's name, and this delay is fully included in the observed round-trip time. This is why we have not presented the evaluation of the latency metric, since its results are not significant at all. Pandora is now able to monitor DNS traffic and, from there, determine this additional latency, but unfortunately this was not implemented at the time the trace was collected.

Since the time we performed these measurements, the INRIA Web caching infrastructure has been greatly reworked. We are currently waiting for the necessary administrative authorisation to make a new series of tests, that we hope will exhibit better performance.

5.2 Cooperating Squids with Web Polygraph Evaluation

In the second experiment, we would have liked to evaluate the quality of service provided by a complete system of cooperating proxy caches. Unfortunately, we did not have the opportunity to perform this test in real conditions. Thus, we have used the Web Polygraph [24] benchmark suite by The Measurement Factory to simulate HTTP clients and servers. This platform has been in use for 3 years by many Web proxy cache vendors (including Lucent Technologies, IBM, Compaq, Dell and Microsoft) to compare the performance of their products.

Controlling both the clients and the servers makes Polygraph able to simulate a real environment with great accuracy. Indeed, this allows:

- unlimited URL space without any disk storage constraints (documents are created by the servers "on the fly");
- representative content type distribution: static and dynamic documents, embedded images, etc.;
- fine grained control over document life cycles (creation and expiration time);
- server and client "think" time: documents are not returned immediately by the servers and clients do not request unrelated documents without delay, while embedded image requests are performed immediately after the main document has been fetched.

Technically speaking, the Web Polygraph benchmark is split into two processes that share the same configuration file: one implementing the clients and one implementing the servers. Each of these processes is able to simulate many different instances of either clients or servers. Each of these instances is assigned an unique IP address. In order to keep the number of different machines small, we make heavy use of IP aliasing that allows several addresses to be bound to the same network interface.

By default, explicit "`Expires`" headers are systematically included in the server responses, letting the caches have an optimal consistency policy. This was done because some cache vendors felt that consistency was not right a metric to expose to their users. We do not share this point of view and, as it will be explained below, we have run some tests with a modified version of Polygraph web server that do not add this header. Besides, Web Polygraph

was not designed to benchmark cooperating caches. We had to trick it to split clients into two halves running on two distinct machines. Each of those contacts a different proxy cache, but shares the accessible URL space (making it possible to have cache peer hits).

We used also IP DummyNet [18] to introduce an artificial latency between the proxies and the servers. As The Measurement Factory did for their own benchmarks, we used a 40 ms additional delay in both directions (resulting in a 80 ms total latency) and introduced an average packet loss of 0.5%.

We chose a switched architecture. This means that any communication between any pair of machines goes through a single router. This allows us to run a single DummyNet (on the router) and it reduces drastically the amount of ARP traffic. Furthermore, this configuration is similar to the one now used at INRIA: proxy caches are located on a different network to the clients, so the requests to the caches flow through the router, and then requests to the servers flow through the same router again to reach the external network. Finally, this allows us to concentrate the vantage points, needed to capture packets, on a lightly loaded machine (compared to the others involved).

For this experiment, client machines are 500 MHz Pentium III, running GNU/Linux. The two proxy caches are Squid [25] version 2.4 each running a 1 GHz Pentium III GNU/Linux box. Servers are located on two machines with the same kind of hardware as the proxies. The router is a FreeBSD 4.4 on a 400 MHz Pentium II. The machine running Pandora is also a 1 GHz Pentium III GNU/Linux that sees all the traffic flowing through the router. Finally, the network interconnecting all these machines is a switched 100baseTX Ethernet.

Only one instance of Pandora was needed to perform the complete analysis, running both the cache and the ICP stacks described in Section 3.2.

A typical Polygraph benchmark starts with a cache filling stage. This step is done in order to perform further measurements with a cache in its usual (filled) operating state. Then, two successive measurement phases take place. Each of these steps is separated by idle periods. Our reference experiment has been set up with the characteristics described in Table 2. Other tests are variations of this one, and the modifications to the reference one are stated in Table 3.

Figure 2 shows the evaluation of our three base metrics (round-trip time, latency, and bandwidth) for our reference experiment. Furthermore, we have

Number of Clients	2×63
Number of Servers	2×258
Request Rate	50 req/s
Cache Size	1 GB
Servers Latency	80 ms
Servers Think Time	2.5 s
Proxy Cooperation	true

Table 2: Reference Web Polygraph experiment.

Name	Characteristic	
size $\times 2$	size	2 GB
think time /2	server think time	1 s
no coop.	proxy cooperation	false
clients /2	request rate	20 req/s

Table 3: Other tests, compared to the reference.

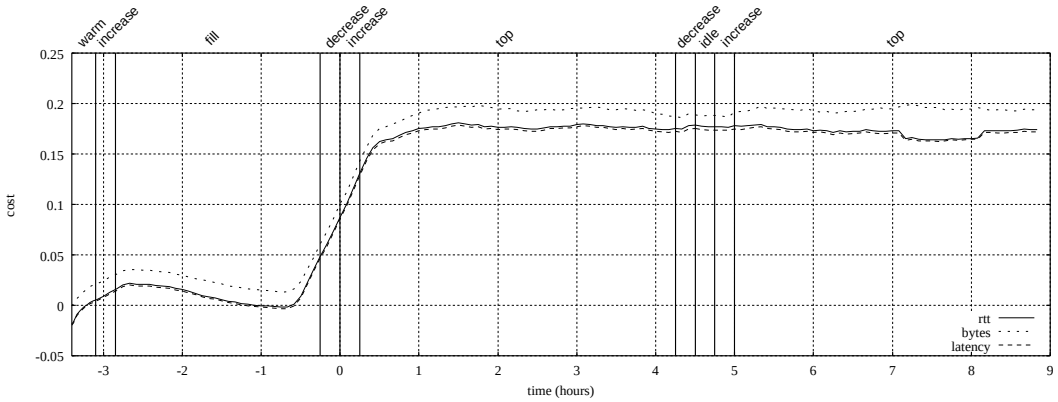


Figure 2: Evaluation of the three base metrics for the reference Polygraph experiment with explicit benchmark phases. Phases of the Web Polygraph benchmark are shown above the frame. The *increase* and *decrease* stages corresponds to 15 minutes phases where the request request falls from 50 req/s down to 5 req/s and rises from 5 req/s up to 50 req/s. The two measurement phases are those called *top*.

indicated in the figure the separations between the various phases. In this figure the origin for the time axis corresponds to the end of the cache filling phase, and thus the beginning of the interesting part of the measurements. Hence, for the following tests, we show only the evaluation performed after this time. Furthermore, we used for all measurements a one hour-large window over 5 minutes samples.

The general layout of this graph is rather intuitive: the higher the hit rate (which corresponds to both so-called `top` phases), the better the efficiency. The fact that improvement is seen before the actual end of the cache filling phase is due to the window on which measurements are averaged: in our case, each computed value takes into account the measures made 30 minutes before and 30 minutes after it.[6] One might also wonder why round-trip time and latency efficiency are substantially inferior to bandwidth efficiency. This may be easily explained by analysing the way these metrics are computed, using a very simple model. In this model, we consider that:

- the time t_t needed to perform a complete HTTP transaction can be split into two parts: the time t_s spent by the proxy to retrieve the document (either from the server or from its cache) and the time t_p accounting for the treatment of the request by the proxy and the transmission of the document between the proxy and the client: $t_t = t_p + t_s$;
- when a miss occurs, the time t^0 needed to retrieve a document without any proxy cache is equal to the time spent by the cache to retrieve the same document : $t^0 = t_s^{miss}$;
- the time needed by the cache to process a request is the same whether this was a miss or a hit: $t_p^{hit} = t_p^{miss} = t_p$;
- in case of a cache hit, $t_s^{hit} \ll t_s^{miss}$ and t_s^{hit} may be neglected.

In the following c_r denotes the round-trip time cost, N the total number of documents considered, N^{hits} the total number of hits among them, HR the observed hit rate, and $\bar{x}$ the arithmetic mean of x.

[6] This implies naturally that each value can only be displayed 30 minutes after the timestamp it shows.

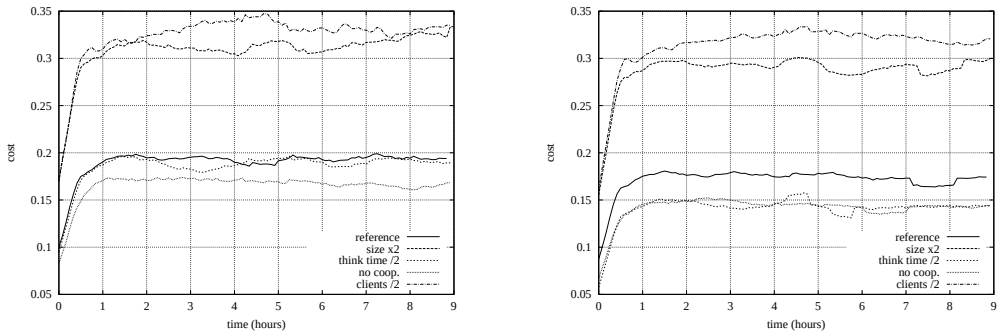
$$
\begin{aligned}
c_r &= \frac{1}{N} \sum_{documents} \left(\frac{t^0 - t_t}{t^0} \right) \\
&= \frac{1}{N} \left[\sum_{misses} \left(\frac{t^0 - t_s^{miss} - t_p^{miss}}{t^0} \right) + \sum_{hits} \left(\frac{t^0 - t_s^{hit} - t_p^{hit}}{t^0} \right) \right] \\
&= \frac{1}{N} \left[\sum_{misses} \left(\frac{-t_p^{miss}}{t^0} \right) + \sum_{hits} \left(\frac{t^0 - t_p^{hit}}{t^0} \right) \right] \\
&= \frac{1}{N} \left[\sum_{misses} \left(\frac{-t_p}{t^0} \right) + N^{hits} + \sum_{hits} \left(\frac{-t_p}{t^0} \right) \right] \\
&= \frac{N^{hits}}{N} + \frac{1}{N} \sum_{documents} \left(-\frac{t_p}{t^0} \right) \\
&= HR - \left(\frac{\bar{t_p}}{t^0} \right)
\end{aligned}
$$

If we suppose that the byte hit rate is roughly equal to the hit rate (which has been verified afterwards for our tests), we can see that round-trip time cost should be lower than bandwidth cost. Indeed, the difference between plain hit rate and round-trip time cost is the ratio of the average cache processing time of requests over the average time needed to download documents from their original location. This tends towards 0 as cache overhead diminishes.

In this figure, we can also see that round-trip time and latency costs are almost identical and this holds for all the experiments we have made. This is due to the fact that, with our benchmark settings, average latency is almost negligible compared to the time it takes to transfer a document over the network. Thus, in the following discussion we will only focus on the round-trip time cost as the same conclusions apply for latency.

For all different configurations, we show on Figure 3(a) the evaluation of the bandwidth metric and on Figure 3(b) the evaluation of the round-trip time metric. This allows us to analyse the influence of the different configuration variations that we have tested.[7]

[7] Of course, one must keep in mind that the conclusions we draw from these experiments only apply to the specific benchmark we have run and that few generalisations could be done.



(a) Bandwidth

(b) Round-trip time

Figure 3: Evaluation of round-trip time and bandwidth metrics.

The two best performing configurations are those with more disk space for one and with fewer clients for the other. If more space improves hit rate, it seems that there is a problem with the configuration of the cache. Indeed, assuming that the replacement policy does not make trivially wrong choices, this means that the number of stored documents is too small to catch up with the time period between two accesses to the same document: they must be discarded beforehand in order to store those freshly downloaded. This could also explain why the configuration with a smaller number of clients performs well. Indeed, if there are fewer clients (hence fewer requests made), the time needed to fetch the same number of documents increases, which allows us to keep a document longer in the cache. Furthermore, in the latter case, the cache overhead is smaller since there are fewer simultaneous connections.[8]

Next we can see that, in our configuration, cooperation was really useful. Naturally, this improves hit rate, but we can observe that cache request handling is not significantly impacted by this. Looking at the round-trip time metric indicates even that possible time improvements do not compensate the hit losses.

[8]Results that could not be presented here, due to space limitations, confirm this obvious fact.

Last, the configuration where server think time had been reduced to 1 second exhibits equivalent byte cost compared to our reference configuration, but worse round-trip time cost. This result is quite straight forward: cache overhead is almost the same in both cases, meaning that the improvement – which we measure – is higher when it takes more time to download documents.

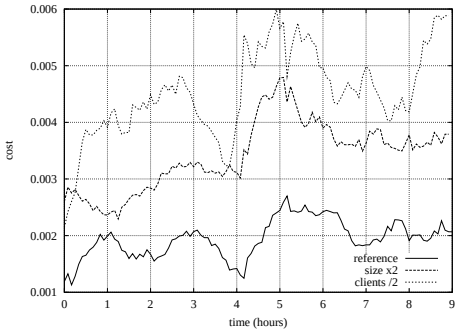


Figure 4: Consistency metric

Figure 4 presents the evaluation of our consistency metric for the reference configuration and the two configurations that exhibit higher hit rates. The first thing to see is that the percentage of stale hits remains low for all configurations (less than 0.6%). However, this figure shows significantly higher stale hit ratios for the last two ones (more disk space and fewer clients) than the reference. We explain this phenomenon by the fact that in these configurations document time-to-live in the cache is longer, which naturally increases the probability of inconsistency. This transforms a little bit the vision one could have of the efficiency of these cache settings. Deciding which one is the most suitable for end-users remains the privilege of the system administrator.

Finally, we show on Figure 5 the results of our measurements of cooperation efficiency. It appears that sibling hits are always much bigger in size than the traffic generated by the ICP protocol. Then we can see that the longer the time-to-live of the documents in the caches, the more efficient the cooperation, which is not surprising. The configuration with server think time reduced to 1 second is more interesting. The figure shows that in this case cooperation is less efficient. As the average time needed to fetch documents

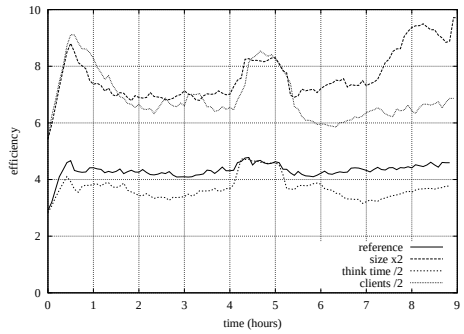


Figure 5: Cooperation efficiency metric

from servers decreases, cooperation is less used by the proxies because there are fewer benefits.

What we would like to retain from these tests is that Pandora's evaluation lets us analyse precisely the different configurations we tried. It also appears that examining the different metrics in parallel is much more instructive than combining them with some sort of compromise function. The accuracy it shows leads us to think that it could be used efficiently to evaluate real systems. Its measurements are detailed enough to show small but significant variations that could have been overlooked otherwise. Of course, with such real systems, one could not test several configurations and just pick up the best. Yet, Pandora provides a very convenient and reactive way to inform the administrator of the cache that some parts of the configuration are to be modified. Furthermore, one can observe very quickly how the corrections made impact the behaviour of the cache.

Furthermore, these tests allowed us to validate Pandora in conditions that we had not met in a real environment. In 12 hours, a single instance of Pandora analysed about 4 million transactions, representing more than 40 GB of traffic. For comparison, this number of requests represents about one week of traffic at INRIA (which counts about 1000 distinct Web users), or more than twice the number of page views over the same period of time (12 hours) for the whole OSDN network (which includes sites like Slashdot, SourceForge, Freshmeat and NewsForge). Furthermore, the machine on which Pandora was running was at least 80% idle (about 5% in user mode and 15% in kernel mode —

mainly spent in interrupt, handling network frames arrival). During the tests, the memory footprint of Pandora was around 40 MB.

These experiments also helped us to discover problems that had remained uncovered beforehand. For example, in our configuration, we noticed that a single stack was not sufficient to reemit requests in cases of cache hits. Indeed, given the number of such requests — between 5 and 10 per second — you have at most 200ms to establish the connection with the server (after this phase, all open file descriptors are handled by a single `select(2)` and documents are read as they are received). Since there exists a 80ms latency between the clients and the servers, and that it takes a least one round-trip to establish a connection, very few spare time remains. As the server becomes loaded, this becomes totally unmanageable and requests are queued, up to the moment (which arises quickly) when requests are emitted several dozens of minutes after they are received. At this time, the transactions had been flushed a long time ago by the component that were interested in it and the request is perfectly useless. We found that distributing the requests in 8 distinct threads solved the problem. Doing this with Pandora was only the matter of creating a new stack, composed by 3 components, all of which existed already.

6 Related Work

Many other studies have been made and many tools exist concerning the measurement of the performance of proxy caches. In this section we focus mainly on the metrics that have been used in the past to perform such evaluation, together with the techniques needed to extract them.

We have selected some of these pieces of work that we consider representative of what has been done in this field. We present them in a concise form in Table 4. We have also included Pandora at the bottom so that it can be easily compared to the others. In this table we have shown for each of these studies the goals that motivated the measurements, the software technique used to extract the metrics and the exact metrics that were used.

Two metrics are very widely used: hit rate (HR) and byte hit rate (BHR). Indeed, these have become over time *de facto* standards to compare various cache configurations. This comes from the fact that while still being good

Reference	Goal	Technique	HR	BHR	RTT	Other
Cao [2]	policy	simulation	√	√	√	hops
Rizzo [19]	policy	simulation	√	√		
Maltzahn [12]	implementation	simulation			√	CPU, memory, I/O
Wolman [27]	evaluation	simulation	√	√	√	
Feldmann [6]	evaluation	simulation	√	√	√	
Menaud [13]	evaluation	simulation	√	√	√	cooperation
Krishnan [9]	evaluation	simulation	√			cooperation
Mahanti [10]	evaluation	simulation	√	√		
Pierre [16]	configuration	simulation	√	√	√	cooperation, consistency
Calamaris [1]	characterisation	log analysis	√	√		
Sparks [22]	evaluation	log analysis	√	√		money
Rousskov [20]	characterisation	instrumentation	√	√	√	CPU, I/O
Pandora	evaluation	trace analysis	√	√	√	cooperation, consistency, latency

Table 4: Comparison of previous cache efficiency measurement tools and metrics.
HR: hit rate, BHR: byte hit rate or bandwidth savings, RTT: round-trip time or response time improvement

indicators of cache behaviour, they are very cheap to obtain: these discrete quantities are the most visible effect of the cache on its environment and they can be directly extracted from log files. Round-trip time improvement (RTT) is also commonly used, but it does not appear as-is in log files. Without simulation, one can only make statistical assumptions for such a metric (comparing mean response time for hits and for misses). These assumptions could be incorrect if requests making cache hits are substantially different from the others. This will be the case if the byte hit rate is different from the hit rate (meaning that hit documents are bigger in size) or if these hits come from closer servers than the average.

Besides, we see that few measurements account for alternative metrics: in particular, consistency is almost never taken into consideration. Although this is achievable in a simulation environment, very few traces contain the necessary information to study this issue. Saperlipopette [16] is an exception: originally, Saperlipopette used proxy log files collected once a day. Immediately thereafter a crawler asked original servers the last modification date of document hits. Since this phase took place up to 24 hours after the actual hit, a non-negligible part of the documents had been modified in the meantime, preventing any conclusions from being drawn (see discussion in Section 4.1). The project leading to Pandora has been started precisely to circumvent this drawback and Saperlipopette is now able to use traces produced by Pandora.

Little work has been done to evaluate cooperation between proxy caches, and no specific metrics emerge to characterise the performance of such systems. Standard metrics are used instead with little insight into the overhead that cooperation introduces compared to its benefits.

Simulation holds a preeminent share of the software used to obtain cache performance metrics. A survey of Web cache evaluation techniques [3] by Brian Davison, which covers most of the studies published about single cache performance, emphasises this fact. Simulators are indeed very valuable because they can capture metrics that are not present in log files. However, these tools are not designed to evaluate caching systems on a continuous basis: they look much more like batch processing systems.

7 Conclusion and Future Work

We showed how Pandora can be used to measure the efficiency of a system of cooperating Web proxy caches. In particular, we showed how it could extract pertinent metrics by monitoring raw HTTP traffic in the presence of caches and evaluate in real-time a specific configuration. We illustrated these issues with two examples that assess Pandora's usability in a real environment.

Compared to existing software, Pandora proposes a well integrated support for the collection, analysis and evaluation stages of efficiency measurement. Moreover, it easy to take into consideration new metrics for this evaluation thanks to the flexibility of Pandora. It is usually the matter of a few lines of code to add a new component that will extract the information and format it for Pandora. Moreover, the fact that Pandora operates on-line allows one to evaluate running systems with no perceptible perturbations for their users.

We are now working on a flexible cache project, which will be able to use monitoring information provided by Pandora to reconfigure itself as the conditions of utilisation change. We have already built a prototype, based on a flexible cache named C/NN. This cache runs on top of the YNVM [17]: a dynamic compiler with C semantics. Among many other features outside the scope of this report, the underlying platform makes heavy use of dynamic library loading. For example, this can be used to replace a function implementation at run time, or — which is our case — to pass callbacks between two distinct programs. Pandora has been adapted to run on the YNWM and C/NN utilises Pandora's API to control its execution. Our early prototype automates the construction of a "site exclusion list". This list is used by the proxy to avoid caching documents coming from "close" sites, thus preventing wasting disk space for documents to which the cache would offer little benefit. In the prototype, C/NN and Pandora are intimately coupled,[9] so that the cache can configure the monitoring performed by Pandora, while being able to receive its notifications. More precisely, Pandora evaluates a latency based efficiency metric of the sites seen in the HTTP requests. When this metric falls below a low-water mark, C/NN is requested to exclude this site from caching. Later on, if the metric rises above a high-water mark, the site is removed from the exclusion list. These marks are dynamically configurable and C/NN sets them

[9] This piece of software received the name of C/SPAN: C/NN in Symbiosis with PANdora.

according to its internal state (disk utilisation, load). Our goal is to extend this to permit a complete dynamic reconfiguration of the cache, including its policies (replacement, consistency).

In the longer term, we are interested in building a system where applications will be able to contract guaranteed "deals" in spite of the dynamics of the Internet. These "deals" can be seen as constraints on the quality of the service that an application requires for its users. In this context, Pandora will help in evaluating in real-time the achieved quality of service and will request system reconfiguration when "deals" cannot be guaranteed any longer.

Pandora is publicly available under the GPL license. See `http://www-sor.inria.fr/projects/relais/pandora/` for release information.

References

[1] Cord Beermann. Calamaris. software. `http://Calamaris.Cord.de/`.

[2] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1st USENIX Symposium on Internet Technologies and Systems*, Monterey, California, December 1997. `http://www.usenix.org/publications/library/proceedings/usits97/full_pap%ers/cao/cao.pdf`.

[3] Brian D. Davison. A survey of proxy cache evaluation techniques. In *Proceedings of the 4th International Web Caching Workshop*, April 1999. `http://www.ircache.net/Cache/Workshop99/Papers/davison2-final.ps.gz`.

[4] Brian D. Davison. Web traffic logs: An imperfect resource for evaluation. In *Proceedings of th INET'99 Conference*, June 1999. `http://www.isoc.org/inet99/proceedings/4n/4n_1.htm`.

[5] Anja Feldmann. BLT: Bi-Layer Tracing of HTTP and TCP/IP. In *9th International World Wide Web Conference*, Amsterdam, The Netherlands, May 2000. `http://www.www9.org/w9cdrom/367/367.html`.

[6] Anja Feldmann, Ramon Caceres, Fred Douglis, Gideon Glass, and Michael Rabinovich. Performance of Web proxy caching in heterogeneous bandwidth environments. In *Proceedings of the INFOCOM '99 conference*, March 1999. `http://www.research.att.com/~anja/feldmann/papers/infocom99_proxim.ps.gz`.

[7] 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/cisi/crisp/crisp-wisp.ps.gz`.

[8] Maciej Koziński. Squeezer. software. `http://www.geocities.com/maciej_kozinski/w3cache/squeezer.html`.

[9] P. Krishnan and Binay Sugla. Utility of co-operating Web proxy caches. *Computer Networks and ISDN Systems*, 30(1–7):195–203, 1998.

[10] Anirban Mahanti, Derek Eager, and Carey Williamson. Temporal locality and its impact on web proxy cache performance. *Performance Evaluation Journal: Special Issue on Internet Performance Modelling*, 42(2–3):187–203, September 2000. `http://www.cs.usask.ca/grads/anm474/papers/perf00.ps`.

[11] G. Robert Malan and Farnam Jahanian. An extensible probe architecture for network protocol performance measurement. In *Proceedings of ACM SIGCOMM '98*, Vancouver, British Columbia, September 1998. `http://www.eecs.umich.edu/~rmalan/publications/mjSigcomm98.ps.gz`.

[12] Carlos Maltzahn, Kathy Richardson, and Dirk Grunwald. Performance issues of enterprise level Web proxies. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, June 1997. `http://www.cs.colorado.edu/~carlosm/sigmetrics.ps.gz`.

[13] Jean-Marc Menaud, Valerie Issarny, and Michel Banatre. Improving the effectiveness of web caching. In *Advances in Distributed Systems*, pages 375–401, 1999. `http://www-rocq.inria.fr/solidor/doc/ps00/Caching.ps.gz`.

[14] Mark Nottingham. Weblog. software. `http://www.mnot.net/scripting/python/WebLog/`.

[15] Simon Patarin and Mesaac Makpangou. Pandora : A flexible network monitoring platform. In *Proceedings of the USENIX 2000 Annual Technical Conference*, San Diego, June 2000. `ftp://ftp.inria.fr/INRIA/Projects/SOR/papers/2000/PFNMP_usenix2000/pand%ora-usenix.ps.gz`.

[16] Guillaume Pierre and Mesaac Makpangou. Saperlipopette!: a distributed Web caching systems evaluation tool. In *Proceedings of the 1998 Middleware conference*, pages 389–405, September 1998. `http://www-sor.inria.fr/publi/SDWCSET_middleware98.html`.

[17] Ian Piumarta. YNVM: dynamic compilation in support of software evolution. In *Proceedings of the Engineering Complex Object-Oriented Systems for Evolution Workshop*, Tampa Bay, Florida, October 2001.

[18] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1), January 1997. `http://info.iet.unipi.it/~luigi/dummynet.ps.gz`.

[19] Luigi Rizzo and Lorenzo Vicisano. Replacement policies for a proxy cache. *IEEE/ACM Transactions on Networking*, 8(2):158–170, 2000. `http://www.iet.unipi.it/~luigi/lrv98.ps.gz`.

[20] Alex Rousskov and Valery Soloviev. A performance study of the squid proxy on HTTP/1.0. *World Wide Web*, 2(1-2):47–67, 1999. `http://www.cs.ndsu.nodak.edu/~rousskov/research/cache/squid/profiling/p%apers/wwwj99.ps.gz`.

[21] Junho Shim, Peter Scheuermann, and Radek Vingralek. Proxy cache design: Algorithms, implementation and performance. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):549–562, 1999. `http://www.ece.nwu.edu/~shimjh/publication/tkde98.ps`.

[22] Michael Sparks, George Neisser, and Richard Hanby. An initial statiscal analysis of the performance of the UK national JANET cache. In *Proceedings of the 4th International Web Caching Workshop*, San Diego, California, March 1999. `http://workshop99.ircache.net/Papers/sparks-final.ps.gz`.

[23] Maciej Stachowiak. Guile. software. `http://www.gnu.org/software/guile/`.

[24] The Measurement Factory. Web polygraph. software. `http://www.web-polygraph.org/`.

[25] Duane Wessels. The Squid Internet object cache. National Laboratory for Applied Network Research/UCSD, software, 1997. `http://www.squid-cache.org/`.

[26] Duane Wessels and K. Claffy. Internet Cache Protocol (ICP), version 2. National Laboratory for Applied Network Research/UCSD, Request for Comments 2186, September 1997. `ftp://ftp.isi.edu/in-notes/rfc2186.txt`.

[27] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Anna R. Karlin, and Henry M. Levy. On the scale and performance of cooperative web proxy caching. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 16–31, December 1999. `http://www.cs.washington.edu/homes/nitin/Papers/sosp99.ps`.

[28] Roland P. Wooster and Marc Abrams. Proxy caching that estimate page load delays. In *Proceedings of the 6rd International WWW Conference*, April 1997. `http://www.scope.gmd.de/info/www6/technical/paper250/paper250.html`.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399