Saperlipopette!: a Distributed Web Caching Systems Evaluation Tool

Guillaume Pierre and Mesaac Makpangou
INRIA SOR group
78153 Le Chesnay Cedex - France
Tel: +33 1 39 63 57 55 — Fax: +33 1 39 63 53 72
Guillaume.Pierre@inria.fr — Mesaac.Makpangou@inria.fr
http://www-sor.inria.fr/

Abstract

Designing a distributed cache infrastructure to improve the Web performance for the users of a large-scale organization is a difficult task. To guide the decisions of system administrators, we propose Saperlipopette!, a tool that can be used to evaluate, *a-priori*, the quality of the service offered by each potential configuration of the distributed cache infrastructure. Saperlipopette! is based on trace-driven simulations. Our methodology is two-fold. First, we monitor the targeted organizations' Web related activity. Second, we replay the organization's access pattern while simulating the distributed Web support infrastructure.

This paper presents the information gathering as well as the design of the tool. We show that beyond a certain cache's size, the performance stays constant whereas the consistency continues to decrease. We also evaluated a number of distributed configurations, among which peer-to-peer Relais cooperation proved to be the best one.

Keywords

Web caches, configuration, performance evaluation, Relais.

1 INTRODUCTION

The performance of the World-Wide Web is getting worse and worse because the number of users grows faster than the network infrastructure capacity. To improve the Web performance for their members, many organizations cache documents that are likely to be requested in the near future. For a small-scale organization, that is an organization of a few dozen members located in the same local area network, a single shared proxy cache is often enough to make the performance acceptable. For large-scale organizations (i.e. thousands of members, possibly located at geographically separated sites), the use of a single proxy cache by all members will certainly not achieve the best possible system performance, at least for the members located far away from the cache.

1.1 Background on Distributed Caching System for the Web

There are mainly three kinds of Web caches: client caches are integrated in the browsers for improving a user's perceived performance; server caches are placed in front of Web servers for speeding up their response time; and proxy caches are placed at an intermediate location and shared by a group of related users (e.g. members of the same intranet or extranet, users located in the same region or country). Our work targets groups of proxy caches.

Proxy caches can run in isolation from each other or cooperate with one another. Four main cooperation strategies have been defined. The first strategy is hierarchical cooperation (Chankhunthod *et al.* 1996): it consists of building a tree-structure where nodes are caches. When a cache misses a document, it delegates the resolution of the request to its parent. Eventually, the requested document is returned to the cache that initiated the request; it then returns this document to the requesting client. On its way back, the response can be cached at each node it traverses.

The second strategy is the inter-cache protocol as implemented by Harvest and Squid (Chankhunthod *et al.* 1996, ICP 1994). When a cache misses a document, it broadcasts lookup messages to its siblings. If none of its siblings can serve this request, the cache retains the control of how to obtain the document.

The two remaining cooperation protocols are represented by CRISP (Gadde *et al.* 1997a) and Relais (Makpangou & Bérenguier 1997). Both systems allow a group of cooperative caches to share a common directory that lists documents cached by the entire group. The main difference between these systems is the representation of this directory. CRISP favors a centralized management whereas Relais replicates this directory at each cache location. For both systems, when a cache misses a document, it looks at the common directory. If a partner owns the requested document in its cache, the request is forwarded to it. If not, the request is sent to the original server (possibly through a parent cache).

The above cooperation strategies don't require the same investment and don't impose the same constraints. For instance, a parent cache will probably need more storage and more CPU than its children; it would be better placed nearer to the backbone than its children. On the other hand, cooperative caches need good network connections between each other and with the source servers.

1.2 Configuration Issues

The benefits of using one particular distributed cache depends on the users' access patterns, the cooperation strategy used by the group, the placement of caches and the sizing of the group of caches. We call any particular set of choices a *distributed cache configuration* (or simply a *configuration*). We also include in the distributed cache configuration the selection of a particular set of internal cache policies: replacement strategy, cache consistency management, deciding which documents to cache, and storage management.

What makes configuring a distributed Web caching system difficult is the lack of information about the performance of candidate configurations. With the current state of the art, a system administrator must choose and setup one particular configuration, evaluate it *a-posteriori*, and probably refine that configuration a number of times before reaching an acceptable result. This approach has a number of drawbacks: (i) it is necessary to invest in the hardware and the software with no guarantee of the resulting performance; (ii) it is impossible to evaluate a large number of configurations, so administrators may miss the optimum solution; (iii) it is difficult to compare the performance of different configurations because the users' access patterns and the network quality of service change as well as the Web cache configuration; (iv) during the configuration's refinement process, users can become discouraged and decide to bypass the caches.

1.3 Saperlipopette!

We present Saperlipopette!, a tool for evaluating *a-priori* the performance of distributed Web cache configurations. It allows a system administrator to capture the characteristics of its users' access patterns as well as his network quality of service. Then it allows the captured traffic to be replayed in differently-configured systems, and to analyze the resulting performance. Given the quality of service and cost information produced by Saperlipopette!, administrators can objectively decide on the appropriate configuration to use.

The rest of the paper is organized as follows. Section 2 presents some related work. Section 3 describes the methodology that we use, and Section 4 presents the design of the the simulation tool that constitutes the heart of Saperlipopette!. Section 5 discusses two case studies that we achieved with Saperlipopette!. Section 6 outlines some perspectives for this work and Section 7 draws some conclusions.

2 RELATED WORK

A lot of work has been done during the last few years on evaluating and improving the performance of Web caches. They can be classified into three groups: statistical analysis on collected traces, evaluations of elementary protocols, and general configuration recommendations.

Studies of the first group try to characterize users' access patterns, and draw some general cache design recommendations. For example, Bestavros et al. showed that Web traces exhibit a high locality, but that most redundant requests concern small documents (Bestavros *et al.* 1995). This result suggests the use of main memory caching (Markatos 1996). They also showed that LAN-level cache sharing can much improve performance. Another interesting study models "perfect" caching algorithms (i.e. algorithms that always take correct decisions based on knowledge of future requests). Such algorithms cannot improve latency more than 26% without prefetching, or 57% with prefetching (Kroeger *et al.* 1997). The main drawback of such studies is that they only give hints on the efficiency of algorithms; they cannot predict the user-perceived performance for a given configuration.

The second group encompasses studies that evaluate the benefits of particular caching policies. Many of them have evaluated the cost and performance of caches for particular replacement policies (Arlitt & Williamson 1996, Cao & Irani 1997, Kim *et al.* 1998, Williams *et al.* 1996), particular consistency protocols (Gwertzman & Seltzer 1996, Worrell 1994), particular cooperation protocols (Gadde *et al.* 1997b) or prefetching techniques (Padmanabhan & Mogul 1996). These studies often rely on trace-driven simulation tools. However, they run the protocols inside rough models of their environment. Such techniques bring out useful comparisons, but they ignore the side effects of the overall caching configuration on the resulting performance. Moreover, they are not designed for predicting the resulting performance in the particular case of a given user's traffic.

A number of experienced system administrators have published information about large-scale experiments, for national caches (Neal 1996, Smith 1996) and for distributed cooperative caching infrastructures (Melve *et al.* 1997, Wessels 1996). Unfortunately, these reports don't help a lot for building a distributed caching system designed for a particular user population. Only a few studies on personalized configuration that we know of address the sizing of isolated caches (Duska *et al.* 1997): *"Small client populations need 1-GB of cache per 35,000 requests/day and larger populations 1-GB per 70,000 to 100,000 requests/day"*. Saperlipopette! proposes a solution for personalizing such recommendations to a particular user's traffic, and for extending its scope to all aspects of configurations: internal caching protocols, distributed architecture and sizing.

3 THE METHODOLOGY

We are interested in evaluating the gain and the costs of using a distributed cache system (i.e. a group of caches) for accessing Web documents. We are mainly concerned by the users' perceived quality of service. To achieve this, we rely on simulations. This choice allows one to replay the same sequence of requests while simulating different numbers of caches, cache locations, cooperation strategies, and so on.

As it is difficult to forge realistic users' access patterns, we decided to run trace-driven simulations. We capture the actual Web traffic of the target organization (intranet or extranet) and replay the same sequence of requests in the simulation. By varying the system configuration, we can determine its impact on the resulting performance.

3.1 The Metrics

A caching system has three main effects on its environment: reducing user-level latency, reducing the load of the worldwide network, and reducing the load of the Web servers. Conversely, it introduces inconsistency problems: the delivered documents are not always up-to-date. The metrics we want to provide are the ones that characterize the user-perceived quality of the the Web system. They include the performance gain and the delivered documents' inconsistency.

The performance of the Web (with or without caches) can be expressed either by the average request latency or the average document retrieval time. The former measures the average amount of time between the beginning of the request and the instant at which the first byte of the response is received; the latter gives the average total duration of requests. The performance gain due to the use of a cache can be obtained by calculating the ratio of the latency (or retrieve time) when the cache system is used, by the one of a configuration with no cache.

The delivered documents' consistency metric is defined as the percentage of outdated documents retrieved from the system. To determine if a delivered document is outdated, we compare its "`Last-modified`" field with the one of the document in the server at the time of the request. If the two timestamps agree (either because the document was retrieved from the server or because the document on the server was not updated since it was previously downloaded by the cache), the document is considered up-to-date. If not, it is considered out-of-date.

3.2 Information Gathering

The information gathering consists of capturing all the Web traffic characteristics needed for the simulation. For that, we exploit the logs of one or several Web proxies carrying the traffic. As the information contained in the logs is insufficient, we collect additional data with a separate tool which directly contacts the Web servers. We also monitor the network's quality of service.

3.2.1 Collecting Information about Document Consistency

To evaluate the cache consistency metric for a given cache system, one expects Saperlipopette! to compare, for each request, both the timestamp of the copy actually delivered to the client and the timestamp of the document that *should have been* delivered. These timestamps are the values of the `Last-modified` fields of those copies. We call the former timestamp the *delivered timestamp* and the latter the *ideal timestamp*.

To generate such information, Saperlipopette! needs only one input datum: the ideal timestamp of each request. When the simulated system performs a cache miss, it assumes that the timestamp delivered by the server is the "ideal" timestamp. It then caches this document together with the ideal timestamp. Then, it logs a record in which the delivered and the ideal timestamps of the document are identical. When the simulated system performs a cache hit, it considers the timestamp attached to the cached document as the delivered timestamp. Hence, it logs a record containing the cached timestamps and the ideal timestamp.

If the proxy used for the monitoring of the traffic is not acting as a cache, it is straightforward to obtain the ideal timestamps (the delivered timestamp is equal to ideal timestamp). That's what Digital have done for collecting their Web proxy traces (Kroeger *et al.* 1996). However, if we monitor an extranet that already has caches, things get complicated. The proxy logs, when caches are used, can only indicate the timestamp of the documents that were delivered during one execution of a particular system configuration, not the timestamp of the document that *should have been* delivered if there were no cache. To address both cases, we separate the traffic monitoring and the gathering of document consistency related information.

Every night, we run a robot which replays the requests of the day. It sends HTTP `HEAD` requests to the Web servers, and looks at the `Last-modified` field of the response. If we call T_{req} the actual timestamp of the request and $T_{lastmod}$ the timestamp of the `Last-modified` response of the server, we can distinguish two cases:

- If $T_{lastmod} \leqslant T_{req}$, then we know that the `Last-modified` field of the document didn't change since the time of the request. We can then use $T_{lastmod}$ as the "ideal" timestamp.
- If $T_{req} < T_{lastmod}$, then the document has been updated between the time of the request and the time of the robot request. We then have no information on the "ideal" timestamp of this request. To reflect this uncertainty, Saperlipopette! calculates the consistency metric both in the worst and best case (i.e. when all uncertain requests are assumed to return out-of-date documents, or when they are assumed to return up-to-date documents).

3.2.2 Collecting Information about Network Performance

For simplicity reasons, the simulation assumes the absence of communication failures, network partitions, disconnections or crashes. It also ignores the delays due to computation or disk accesses. For building the performance metrics, it only needs data about the elapsed network transfer times. To determine the user-perceived performance, the simulator relies on knowledge of the actual document size as well as the characteristics of connections with document providers (both proxy caches and source servers). Once those are known, Saperlipopette! can determine the performance gain metric.

To obtain the actual sizes of documents, we face the same problem as for the document consistency related information: the cache logs cannot indicate the size of the documents that should have been delivered. To gather the actual sizes of documents, we used the same solution, i.e. to make our robot look at document sizes as well as timestamps.

To determine the connection characteristics, we took a very simplistic approach: for each Web server involved in our traces, we estimate the average latency and bandwidth observed for typical Web requests to that server. We built another small monitoring tool dedicated to network performance measurements: periodically, during daytime, it sends one HTTP `GET` request to each of the Web servers involved in our traces. We decided to request the root document of each server (i.e. its home page). This guarantees that documents retrieved this way are of approximately of the same size (a few kilobytes), and are representative of typical Web requests. In addition, the tool also monitors the internal connections (between clients and caches, and between the caches).

4 SIMULATION TOOL DESIGN

The heart of Saperlipopette! is a trace-driven distributed cache simulation tool built on top of a generic discrete event scheduler.

4.1 The Distributed Cache Simulation

To ease the evaluation task, we designed a highly modular and parameterizable distributed cache simulation which conforms to the design of Oléron (Baggio & Pierre 1997). Roughly, a Web cache is decomposed into several independent modules which implement elementary features: consistency control, replacement, inter-cache cooperation, and so on. To define a new policy, one only needs to inherit from the corresponding abstract class and code the specificities of the algorithm.

To setup one particular configuration, the evaluator instantiates (i) a simulated server for responding to all missed requests; (ii) a number of caches with their internal policies, their dimensioning and their cooperation scheme; (iii) and a number of clients (one per cache), each with the scenario it is to execute. All clients and caches are also given the network connections characteristics as perceived from their location.

4.2 The Scheduler

Although we consider a distributed Web infrastructure (with multiple clients, caches and servers), we run a mono-process simulation. Hence, the simulation relies on an event scheduler to trigger all the actions of the distributed system: these include issuing requests, arrivals of requests or responses, documents' time-to-live expirations, etc.

The scheduler manages an event handler queue. An event handler is an object of any C++ class that inherits from the `EventHandler` base class. It must implement a `go()` method which defines the behavior of the handler. Each event handler to be executed by the simulation must be registered with the scheduler, together with its wake up time. The scheduler is in charge of sorting the event handlers, and triggering them at the right time. It also manages the simulated time: when no event is running, it can safely bring forward the simulated time, and triggers the next event handler immediately.

For simplicity reasons, the scheduler doesn't allow concurrent execution of events. This imposes an important constraint: each event must be instantaneous in terms of the simulated time. In our model, an instantaneous operation (e.g. a local cache lookup) is implemented as a single event. Conversely, an operation which takes some simulated time (e.g. a document request) has to be divided into two events: the first event starts the computation, calculates the duration of the operation, creates a new event handler, registers it with the scheduler at the calculated termination time, and returns control to the scheduler. At the requested time, the scheduler will trigger the second handler, which can then finish the operation.

For example, the simulation of a cache request is done via two event handlers. The `BeginOfRequest` handler implements the beginning of the request: it makes a lookup for the URL in the cache. If it finds it, then it returns the document immediately. If it doesn't find it, it initiates a request to the Web server. For that, it first computes the timestamp at which this request will arrive; it creates a `ServerRequest` handler, and registers it to the scheduler (Figure 1). Later on, the scheduler will trigger the `ServerRequest`, allowing it to deliver the request to the server (Figure 2).

5 APPLICATIONS

We present here two example studies realized with Saperlipopette!: the first one shows the impact of a cache's size on the resulting performance and consistency; the second studies the impact of different distributed caching architectures on the resulting performance.

5.1 Collected Traces Analysis

The example we chose to work on is an extranet, namely the INRIA network. INRIA comprises five research units, located in different parts of France. The system

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

Figure 1 Simple pseudo-code BeginOfRequest handler

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

Figure 2 Simple pseudo-code ServerRequest handler

we studied is the overall caching system, intended for enabling INRIA personnel to browse the Web. We actually collected the two main research units' cache traces (Rocquencourt and Sophia) from the 13th September 1997 to the 9th February 1998. Table 1 shows some of its characteristics. We also measured the network QoS, which we summarize in Table 2.

A quick statistical analysis of the trace shows interesting results. By comparing the number of requests and the number of unique requests, we can see that the theoretical maximum Hit rate of Rocquencourt is $(1021417 - 674212)/1021417 = 34.0\%$, Sophia's one is 34.2%, and the whole group's one is 37.3%.

These figures also show the potential of resource sharing between Rocquencourt and Sophia. If we compare the number of unique requests for Rocquencourt, Sophia, and all together, we see that there are $392262 + 674212 - 1014390 = 52084$ unique documents requested by both Rocquencourt and Sophia. This represents 5.1% of the total number of unique documents.

Table 1 Requests trace characterictics

	Nb reqs	*Nb unique reqs*	*Nb Clients*	*Avg doc size*
Rocq	1021417	674212	726	13.4 kb
Soph	596499	392262	364	11.4 kb
All	1617916	1014390	1089	12.1 kb

Table 2 Network quality of service

	Local connection	*Rocq-Soph*	*INRIA-external servers*
Avg. latency	5 ms	100 ms	7569 ms
Avg. bandwidth	300 kb/s	45 kb/s	1.3 kb/s

5.2 Scope of the Simulation

As previously pointed out, the simulator can be configured in many ways. One can easily change the internal policies (replacement, consistency control, inter-cache co-operation), the dimensioning, the number and placement of the caches. Of course, if we vary all of these parameters at the same time, an exponential number of configurations must be simulated. We decided to limit this number by fixing the following parameters:

- The replacement policy we use in the following studies is LRU.
- The consistency control policy we use is Alex(Gwertzman & Seltzer 1996) (often called TTL): it allocates a Time-To-Live to incoming documents proportional to the document's age. When a document's TTL expires, it is ejected from the cache. In our studies, we use a fixed proportion of 30% for all documents, which is representative of typical cache configurations.

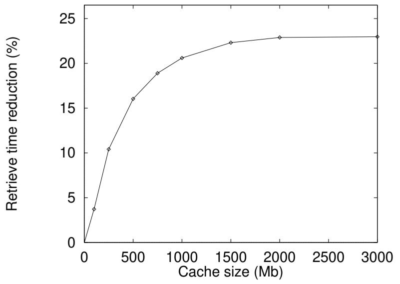
The variable parameters are the cache size, the placement of the caches and the cooperation scheme between them: no cooperation at all, hierarchical relations and Relais-like cooperation. This latter protocol is a simplified version of Relais (Makpangou & Bérenguier 1997): it assumes no propagation delay for the replicated shared directory; and all the consistency control parts of Relais have been ignored.

5.3 Impact of Cache Size

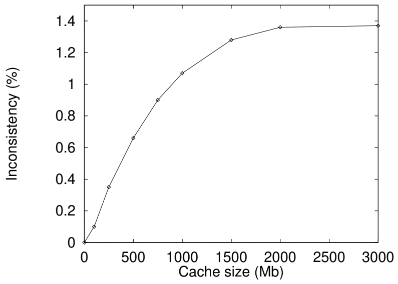
The experiment presented here is simple: whereas INRIA users are actually located in two different places, we assume they are all present in a single location. Every user accesses a single shared cache. We change the cache's size, and watch the variations of documents' inconsistency and retrieve time reduction (Figure 3). The reductions are given by comparing the observed performance with the one of a configuration with no cache at all.

We can see that performance increases when the cache size increases. The performance reaches an asymptote near 1.5 Gb: adding cache storage space doesn't improve significantly the performance. One can't get more than a 22% reduction in retrieve time from this cache.

The inconsistency also increases when cache size increases. A large storage space

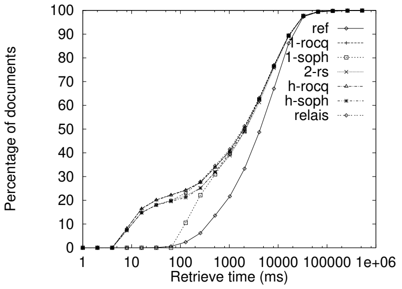


(a) Retrieve time reduction

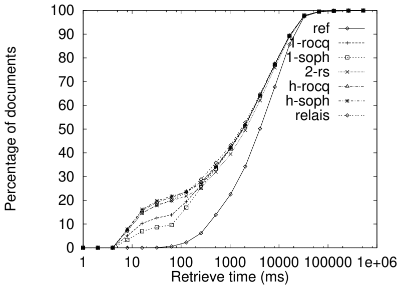


(b) Inconsistency

Figure 3 Performance and consistency of a single Web cache



(a) Rocquencourt



(b) All

Figure 4 Infinite size caches

allows documents to stay longer in the cache, increasing their probability of becoming out-of-date. As for the retrieve time, inconsistency reaches an asymptote; but it does so at a higher cache size (near 2 Gb). After a certain size, adding extra storage space hardly increases performance, but significantly increases inconsistency: most of the extra cache hits actually deliver out-of-date documents.

5.4 Impact of the Distributed Architectures

We show here the impact of several distributed caching architectures on the resulting performance. The architectures we simulated are presented in Table 3. They include a reference configuration with no cache at all, two configurations with one single cache located at Rocquencourt or at Sophia, and three cooperative configurations. The cooperation scheme can be hierarchical relationship (with the parent cache at Rocquencourt or Sophia), or peer-to-peer Relais-like cooperation.

5.4.1 Infinite Cache Size

We study here caches with an infinite storage size. To analyze the results in detail, we show the cumulative retrieve time both as graphs and tables. They show the percentage of documents retrieved in less than a given duration.

Figure 4 and Table 4 show the cumulative retrieve time for infinite size caches,

Table 3 Configurations

Config	*Description*
Ref	No cache at all: the clients access directly the servers.
1-rocq	One single cache, located at Rocquencourt.
1-soph	One single cache, located at Sophia.
2-rs	Two caches at Rocquencourt and Sophia, with no cooperation between them.
h-rocq	Two caches; Rocquencourt's cache is the parent of Sophia's cache.
h-soph	The same as h-rocq, but the parent is at Sophia.
relais	Two caches cooperating with a Relais-like protocol.

Table 4 Cumulative retrieve time (infinite size caches — All users)

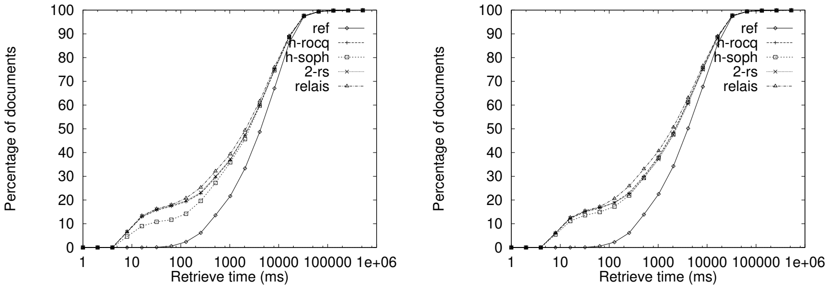
Config	*Ref*	*1-soph*	*1-rocq*	*2-rs*	*h-soph*	*h-rocq*	*relais*
128 ms	2.26%	16.90%	19.45%	21.79%	23.48%	23.63%	23.48%
1024 ms	22.52%	41.71%	41.99%	39.47%	41.97%	42.18%	43.14%
8192 ms	67.76%	77.11%	77.19%	75.92%	77.16%	77.24%	77.59%

from the point of view of Rocquencourt users and all users. We don't show the point of view of Sophia users, as their profile is very similar to the one of Rocquencourt.

We can observe the "no-cache" reference curve, which has a well-known shape: almost no document is retrieved within 100 ms, then most documents are retrieved between 1 second and 100 seconds. The curves for configurations including caches have a different shape: about 20% of the documents are retrieved within 100 ms; the curve remains significantly higher than the reference until 20 seconds retrieve time.

The graphs show that caches widely improve document retrieval time. For example, 22.52% of the documents are retrieved within 1024 ms without any cache, whereas the caching configurations increase this proportion to 43.14%. This corresponds to the user-level experience of Web caches: many more documents are retrieved "instantaneously".

The Figure 4(a) shows the perceived improvement for Rocquencourt users only. We can see three different shapes: the reference one, the one of "good" caching strategies (1-rocq, 2-rs, h-rocq, h-soph, relais), and a single one staying between them (1-soph). This curve corresponds to an obviously sub-optimal configuration: one single cache, placed in Sophia. Of course, the high latency between Rocquencourt and Sophia degrades the documents' retrieve time, even in the case of cache hits. Nevertheless, as Rocquencourt-Sophia bandwidth is quite good, this curve joins the others when latency becomes a negligible part of the total retrieve time. This



(a) Rocquencourt (b) All

Figure 5 1 Go total size caches

effect is identical from Sophia's point of view, with a single cache located at Rocquencourt.

From the overall user group's point of view, we can distinguish three groups of configurations: the reference, the configurations with a single cache, and the distributed configurations. The performance of all distributed configurations are very close, except the 2-rs configuration, which has significantly lower performance (Table 4). This shows that inter-cache cooperation increases performance. The documents downloaded at high cost by one cache can be later obtained at low cost by a partner cache.

If we try to distinguish the cooperative configurations, we see that hierarchical configurations are slightly better than peer-to-peer ones for very small retrieval times whereas peer-to-peer configurations are better for higher retrieval times. But no real difference distinguishes the two cooperation schemes.

5.4.2 Limited Cache Size

We then wondered if the distributed configurations would have very different performance for smaller caches: inter-cache cooperation (either via hierarchical or via peer-to-peer relations) should be much more useful when sharing limited resources. We studied the previous distributed configurations, in the case of a total of 1 gigabyte being distributed among two caches (we ignore the single-cache configurations, as they proved to be sub-optimal). For all configurations, Rocquencourt had 700 Mb of storage space and Sophia had 300 Mb (because Rocquencourt has much more traffic than Sophia), except for the h-soph configuration where Sophia had 700 Mb and Rocquencourt had 300 Mb. The cumulative retrieval times are shown in Figure 5 and in Table 5.

This time, we clearly see that cooperative caches lead to better performance than non-cooperative ones; among cooperative relations, peer-to-peer caches are better than hierarchical ones.

We next decided to retain the peer-to-peer configuration only, and search for the optimal storage size distribution between the two caches. Figure 6 shows the average retrieval time improvement when varying the storage space distribution.

If we observe the shape for Sophia, we can see that the optimal configuration for

Table 5 Cumulative retrieve time (1 Gb cache — All users)

Config	*Ref*	*2-rs*	*h-soph*	*h-rocq*	*relais*
128 ms	2.26%	18.88%	17.21%	18.91%	20.61%
1024 ms	22.52%	37.15%	37.82%	38.33%	40.70%
8192 ms	67.76%	74.89%	75.33%	75.54%	76.53%

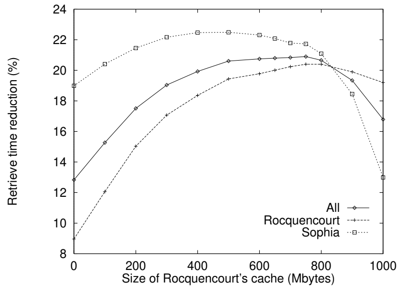


Figure 6 Retrieve time reduction of peer-to-peer cooperating caches

their users is not to take all the storage space. The optimal distribution, from their point of view, is to take 500 Mb, and leave the remaining 500 Mb for Rocquencourt. We explain this as follows: in the case of having all the storage for themselves, they don't benefit from Rocquencourt's users retrievals. In the other hand, if they give some storage space to Rocquencourt, they lose some local hits due to the fact that their local cache is smaller, but benefit from Rocquencourt's retrievals. Of course, if Sophia's cache gets too small, then they will start to have a lot of "neighbor hits", but will see increased retrieval time because the latency between Rocquencourt and Sophia is higher than the local one.

Similarly, from the point of view of Rocquencourt, the optimal placement is 800 Mb at Rocquencourt and 200 Mb at Sophia. The conflicting interests of both sites are balanced when allocating 750 Mb to Rocquencourt and 250 Mb for Sophia: no site gets the best possible performance, but this balance optimizes the overall performance.

This application study shows how we can use Saperlipopette! for evaluating and comparing distributed architectures. We showed that, in the particular case of INRIA, peer-to-peer cooperation leads to better performance than hierarchical cooperation; and any cooperation is better than no cooperation at all. We also showed how Saperlipopette! helps the sizing and placement of caches: if Rocquencourt and Sophia decided to invest in a total 1 Gb of storage space, the optimal placement would be 750 Mb at Rocquencourt and 250 Mb at Sophia.

6 FUTURE WORK

The current prototype presents a number of limitations. They can be classified in two categories: some are due to human factors, others are introduced by simplifications we made (both in the gathering and simulation processes).

The caches' quality of service has an impact on the users' behavior. For example, if a Web caching system suddenly becomes much faster, users will probably start to browse more. Similarly, if users notice that caches always deliver up-to-date documents, they will stop using the "reload" button of their browser. Our evaluation tool doesn't anticipate such possible changes on users' behavior and assumes that the users will conform to their past behavior.

The data collection process ignores a number of requests issued by users of the target system. (i) It takes into account only the HTTP requests; our simulator doesn't include FTP or Gopher protocols. (ii) It selects only the `GET` requests because documents obtained via `POST` or other HTTP methods are generally not cached. It also ignores the requests to CGI scripts for the same reason. (iii) Some requests lack reliable information (e.g. the ideal timestamp).

The wide-area network simulation needs to be improved: it is currently based on simple measurements, and doesn't simulate the network failures or performance fluctuations. Another simplification we made on the simulator is that it only takes into account the network costs, whereas under high load the disk performance has proved to be crucial (Smith 1996). Similarly, a number of protocols have been simplified in the simulator's implementation.

We also plan to develop additional policy modules for the simulator: caching policies (e.g. invalidation consistency control protocols), cooperation (e.g. ICP), prefetching and automatic mirroring. These protocols will allow us to continue the case study of INRIA traffic. We also plan to use Saperlipopette! for studying other target environments.

Saperlipopette! can also be used for fast evaluation of newly designed caching algorithms. Instead of integrating the new algorithm in a real caching system (which can be a long and painful process), it is easy to build a rough version in the simulator. After an evaluation, and possibly a refinement process, one can decide if actually deploying the new algorithm is worth the implementation cost or not.

7 CONCLUSION

We presented Saperlipopette!, a distributed Web cache evaluation tool. This tool can help large-scale organizations to make the decisions that suit their needs: the quality of the service offered to its members as well the costs to the organization. It can help to optimize the use of resources of large organizations. Finally it can help save money by identifying sub-optimal configurations.

We discussed the utilization of Saperlipopette! to evaluate various configurations of a distributed cache shared by the personnel of INRIA-Rocquencourt and INRIA-

Sophia. Though this study is still in its preliminary phase, we reported some initial results. Beyond a certain cache's size, the performance stays constant whereas the consistency continues to decrease. We also evaluated a number of distributed configurations, among which peer-to-peer Relais cooperation proved to be the best one.

Our future plans are to complete this study as well as to use our tool to evaluate potential benefits of different configurations. We also consider using Saperlipopette! to guide the refinement of some existing cache policies.

REFERENCES

Arlitt, M. F. & Williamson, C. L., (1996) Trace-driven simulation of document caching strategies for Internet Web servers. `http://www.cs.usask.ca/faculty/carey/papers/webcaching.ps`.

Baggio, A. & Pierre, G., (1997) Oléron: supporting information sharing in large-scale mobile environments. In *Proceedings of the ERSADS '97 seminar*. Zinal, Switzerland.

Bestavros, A. *et al.*, (1995) Application-level document caching in the Internet. In *Proceedings of the 2nd International Workshop in Distributed and Networked Environments (IEEE SDNE '95)*. Whistler, British Columbia.

Cao, P. & Irani, S., (1997) Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*. Monterey, CA.

Chankhunthod, A. *et al.*, (1996) A hierarchical Internet object cache. In *Proceedings of the 1996 Usenix Technical Conference*. San Diego, CA.

Duska, B. M. *et al.*, (1997) The measured access characteristics of World-Wide-Web client proxy caches. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*. Monterey, CA.

Gadde, S. *et al.*, (1997a) Reduce, reuse, recycle: An approach to building large Internet caches. In *Proceedings of the HotOS '97 Workshop*.

Gadde, S. *et al.*, (1997b) Directory structures for scalable Internet caches. Technical Report CS-1997-18, Duke university.

Gwertzman, J. & Seltzer, M., (1996) World-Wide Web cache consistency. In *Proceedings of the 1996 Usenix Technical Conference*. San Diego, CA.

ICP, (1994) Internet cache protocol specification 1.4. `http://excalibur.usc.edu/icpdoc/icp.html`.

Kim, I. *et al.*, (1998) Analysis of buffer management policies for WWW proxy. In *Proceedings of the International Conference on Information Networking (ICOIN-12)*. Tokyo, Japan.

Kroeger, T. M. *et al.*, (1996) Digital's Web proxy traces. `ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.v1.2.html`.

Kroeger, T. M. *et al.*, (1997) Exploring the bounds of Web latency reduction from caching and prefetching. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems*. Monterey, CA.

Makpangou, M. & Bérenguier, E., (1997) Relais : un protocole de maintien de cohérence de caches Web coopérants. In *Proceedings of the NoTeRe '97 symposium*. Pau, France.

Markatos, E. P., (1996) Main memory caching of Web documents. In *Proceedings of the 5th International WWW Conference*. Paris, France.

Melve, I. *et al.*, (1997) Building a Web caching system - architectural considerations. In *Proceedings of the 8th Joint European Networking Conference*. Edinburgh, Scotland.

Neal, D., (1996) The Harvest object cache in New Zeland. In *Proceedings of the 5th International WWW Conference*. Paris, France.

Padmanabhan, V. N. & Mogul, J. C., (1996) Using predictive prefetching to improve World-Wide Web latency. In *Proceedings of the ACM SIGCOMM '96 Conference*. Stanford University, CA.

Smith, N. G., (1996) The UK national Web cache - the state of the art. In *Proceedings of the 5th International WWW Conference*. Paris, France.

Wessels, D., (1996) Evolution of the NLANR cache hierarchy: Global configuration challenges. `http://www.nlanr.net/Papers/Cache96/`.

Williams, S. *et al.*, (1996) Removal policies in network caches for World-Wide Web documents. In *Procedings of the ACM SIGCOMM '96 Conference*. Stanford University, CA.

Worrell, K. J., (1994) *Invalidation in large scale network objects caches*. Master's thesis, Faculty of the graduate school of the University of Colorado.

8 BIOGRAPHY

Guillaume Pierre is a PhD student at University of Evry val d'Essonne and at INRIA SOR group. His main interests are flexible systems and Internet performance optimization. His thesis focuses on performance studies and optimization for distributed Web caching systems.

Mesaac Makpangou received a Ph.D. from University Pierre et Marie Curie of Paris (France), in february 1989. He then spent a one-year post-doctoral at Cornell, in the ISIS group with Professor Kenneth Birman, where he developped the Isis' longhaul stuff. Since 1990, he holds a "Chargé de Recherche" position at INRIA, within the SOR Group. His current research interest concerns efficient support of large-scale cooperative applications.