Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Networks 49 (2005) 743–765
Computer Networks
www.elsevier.com/locate/comnet

Distributed redirection for the World-Wide Web

Aline Baggio *, Maarten van Steen

Vriji Universiteit, Computer Science, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

Received 6 May 2004; received in revised form 28 October 2004; accepted 14 January 2005
Available online 26 March 2005

Responsible Editor: C.B. Westphall

Abstract

Replication in the World-Wide Web covers a wide range of techniques. Often, the redirection of a client browser towards a given replica of a Web page is performed after the client's request has reached the Web server storing the requested page. As an alternative, we propose to perform the redirection as close to the client as possible in a fully distributed and transparent manner. Distributed redirection ensures that we find a replica wherever it is stored and that the closest possible replica is always found first. By exploiting locality, we can keep latency low.

Keywords: Web-client request redirection; Web site replication; Content delivery networks; CN request-routing

1. Introduction

Replication in the World-Wide Web encompasses a wide range of techniques from proxy caches to mirrors and content distribution networks (CDNs). One goal of these replication mechanisms is to allow clients to use the replicas that best suit their needs in terms of network distance, consistency or security. Nevertheless, the use of location-dependent URLs in today's World-Wide Web does not facilitate transparent access to replicated Web pages. Instead, it is often necessary to explicitly pass a replica's URL to the client from where another attempt to access the document can be made.

Redirection in the case of proxy caches occurs in an implicit way: each HTTP request is routed through the cache or the hierarchy of caches and, in the best case, the replica of the requested Web page is retrieved directly from the cache storage space. In the case of mirrors or CDNs, the client browser has to be explicitly redirected to a host that is normally not on the route followed by the request. Redirection is, in most cases, achieved only after a client's request has reached the *home* server of a document, that is to say, the host

* Corresponding author. Tel.: +31204447707.
E-mail address: baggio@cs.vu.nl (A. Baggio).

1389-1286/$ - see front matter
doi:10.1016/j.comnet.2005.01.019

named in the document's URL. The decision where to redirect a client to is therefore centralized.

One important disadvantage of centralized redirection is the induced latency. Another is that the home server may become overloaded. Ideally, a client request should not be forced to go all the way to the home site in order to be redirected to a close-by replica. On the contrary, the redirection should take place as soon and as close to the client as possible. We have devised a distributed redirection scheme in which the redirection decision can be taken locally at the client machine or, in the worst case, before the HTTP request leaves the client's network. In this paper, we present our design and show how it can be transparently integrated with the current Web.

The paper is organized as follows. Section 2 gives a brief overview of the existing redirection methods for the World-Wide Web. It outlines the advantages and disadvantages of each method. Section 3 presents the principles of our distributed redirection scheme. Section 4 details the design of the redirection server. Section 5 describes aspects concerning client and redirection server interaction. Section 6 presents simulation results of the distributed redirection mechanisms. Section 7 discusses some of our design choices. And finally, Section 8 concludes and gives some future work directions.

2. Alternatives for redirecting clients in the Web

Redirection in today's World-Wide Web is achieved in three different ways: application-level redirection, DNS-based redirection and transport-level redirection [5].

HyperText Transfer Protocol (HTTP) features can be used to achieve application-level redirection. Whenever a client browser requests a Web page, it contacts the Web server named in the URL of the page. Instead of directly sending back the contents, the Web server can decide to redirect the client browser to another server. This redirection takes the form of another URL naming the server where a replica of the requested page can be found [5,8]. The browser then issues a new HTTP request to fetch the Web page at the replica site. HTTP is widely used for communication with browsers, servers and proxy caches [8]. As a consequence, HTTP-based redirection has the merit of being simple and easy to deploy.

The Transmission Control Protocol (TCP) can also be used to redirect clients [5,12,20] and belongs to the group of transport-level redirection mechanisms. In TCP, communicating parties are identified by an end point: the network address of the machine on which the party resides and the port number it uses. The data exchanged between two communicating parties are sent in portions called segments. The origin end point can be falsified when producing TCP segments. Using this feature, a third party such as a Web server hosting a replica of the requested Web page can let the requesting client believe that the segment originates from the original Web server. The client browser keeps sending its requests to the original Web server. The requests are intercepted by the original Web server's gateway, which forwards the requests to the Web server holding the replica. This Web server can respond directly to the client (with falsified origin end points) or indirectly through the original Web server's gateway. The former approach for redirection at the transport-layer level is known as TCP handoff, the later as TCP splicing.

Finally, the Domain Name System (DNS) can be used for redirection purposes [5,10,18]. DNS-based redirection exploits the fact that a browser needs to resolve the domain name contained in a URL to a network address. Unless the name-to-address mapping is already cached at the client's DNS, the client's DNS request eventually reaches the DNS server responsible for the Web server's domain (i.e., the authoritative DNS server). As a reply, the authoritative server can decide to send any appropriate network address and not only the address of the Web server designated in the URL. In particular, the DNS server can respond with the address of a Web server holding a replica of the Web page. The returned address is cached at the client's DNS. The subsequent DNS requests for this domain are therefore resolved to the replica's network address until the address is flushed from the DNS cache. A similar approach, used

for example in the Akamai Content Distribution Network [7], redirects a client in two steps. The client's DNS request reaches first the DNS server responsible for the Web server's domain, as in the previous case. It is asked to contact a close-by DNS server which chooses the replica to which the client is redirected. Until the DNS cache get flushed, the client keeps contacting the close-by DNS server. This approach is known as two-tier DNS-redirection.

Each of these three methods for Web redirection has its own characteristics that make it not entirely satisfactory. Most importantly, the three methods require that the actual redirection is done by a server close to the Web server hosting the requested page. Either it is the Web server itself, the front end to the Web server (in the case of a Web cluster) or the Web server's DNS server. As a consequence, a large number of requests travel up to the server side before the redirection takes place. DNS-based and two-tier DNS redirection tackle this problem by using DNS caches. While each first request for a particular domain has to travel to that domain's DNS server before being redirected, subsequent requests can in the best case be treated locally using the client's DNS or a close-by one in the case of two-tier DNS redirection. In fact, DNS-based redirection mechanisms rely heavily on temporal locality in accessing documents. Whenever DNS cache entries get flushed due to staleness or replacement in the cache, a request has to travel all the way to the Web server side again. CDNs like Akamai make use of two-tier DNS-redirection and are therefore subject to these limitations.

The worst case with respect to not benefiting from locality is HTTP-based redirection, where each single request has to be redirected independently of the others. Latency is thus an important disadvantage of HTTP-based redirection [5]. However, since subsequent redirections are independent from each other, HTTP-based redirection provides a fine granularity that TCP- and DNS-based redirection cannot offer. For both DNS-based and two-tier DNS redirection, the granularity of the redirection is the DNS domain name. This makes the use of different replica repositories for different (sub)directories in a given domain impossible. For example, one may like to replicate the 'Bioinformatics' and the 'Computer Systems' pages from the Vrije Universiteit Web server http://www.cs.vu.nl separately. However, a given Web server, accessible through a given domain name such as www.cs.vu.nl has to be replicated as a whole or make use of virtual domains at the Web server level. This coarse granularity also makes it difficult to use different replication policies for different documents of the same domain as advised in [16].

Another disadvantage of HTTP-based redirection is that it does not provide support for redirection transparency. Clients are aware of the fact that they are redirected since the replica's address is passed to the client. This allows a client to cache and reuse references towards replicas, which may conflict with the redirection policy of the Web server. On the other hand, TCP handoff or TCP splicing are fully transparent but not scalable. The traffic generated by the segment forwarding in TCP handoff makes the method more suited for long-lived sessions such as FTP [5,17] or for use in local-area networks such as with clusters of Web servers. In that respect, DNS-based redirection is more scalable, as messages need not be forwarded and travel in the best case to local DNS servers. DNS-based redirection also achieves a reasonable transparency provided the users are referring to documents using domain names and not IP addresses. In the latter case, the DNS name resolution is by-passed and so is the redirection.

3. Principles of distributed redirection

Considering the disadvantages of HTTP-, DNS- and TCP-based redirection, we would like to devise a redirection method offering a fine granularity in redirection without loss of scalability or transparency. We consider scalability by locality important. First, a request to look for a replica of a Web page has to avoid traveling a long distance. Second, the selected replica should remain the nearest possible to the client browser. This is what we refer to as *network locality*. In addition, we would like our redirection mechanism to be as independent as possible of temporal locality,

as used for example in DNS-based redirection mechanisms. It should also work well for pages that are frequently updated and as such cannot simply be replicated everywhere.

To explain, consider a replicated Web page, referred to as http://www.globule.org/index.html that is available at four Web servers: Amsterdam (the "home" location), Naples, San Francisco and Sydney (see Fig. 1). Assume a client browser located in San Diego issues an HTTP request for the Globule page. With the current redirection mechanisms, the request travels, in principle, to the home server in Amsterdam and only there is it redirected to a close-by replica. We propose to improve locality for client HTTP requests by using a collection of redirection servers installed close to the clients. In our example, the browser's HTTP request is processed first by its local redirection server in San Diego.

For preserving locality when looking up replicas, a redirection server knows only about pages that are available in its own area. Since the Globule Web page is not locally available, the redirection server in San Diego has to issue a lookup request to find a replica. To keep the communication costs relatively low and preserve locality, a redirection server always tries first to find a requested Web page in its vicinity and gradually expands the search area if necessary. The gradual search expansion is achieved by organizing the collection of servers as a hierarchy and by forwarding the lookup requests along this hierarchy. The organization of the redirection servers is done on a per-page basis: each page or group of pages has its own separate hierarchical organization of servers that assist in redirecting HTTP requests. To further enforce locality, only leaf servers store addresses of replicas. The information on which leaf server holds which replica is distributed to the relevant intermediate servers so that any replica can be found when issuing a request at any point of the hierarchy. Fig. 2 shows the hierarchy for the page http://www.globule.org/index.html.

The replica lookup request issued by the redirection server in San Diego is further treated as follows. It reaches the page's redirection server for the USA. The intermediate USA server does not have an address for the Web page. However, as shown in Fig. 3, it holds a pointer to a child redirection server, here in San Francisco, which is known to have information about a replica of the page. The USA server further forwards the lookup request to the redirection server in San Francisco. San Francisco replies with the address of the Web page completing the lookup request. It is the task of the client's redirection server in San Diego to actually retrieve the Web page from the San Francisco Web server. Finally, the San Diego server can decide to cache the address. The client browser will benefit from caching, for example, when requesting the inline images of the document.

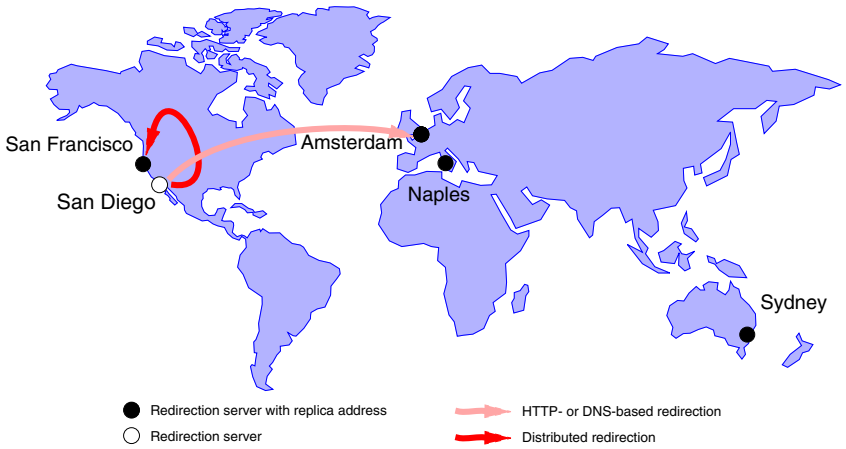

Fig. 1. Using distributed redirection to access a close-by replica.

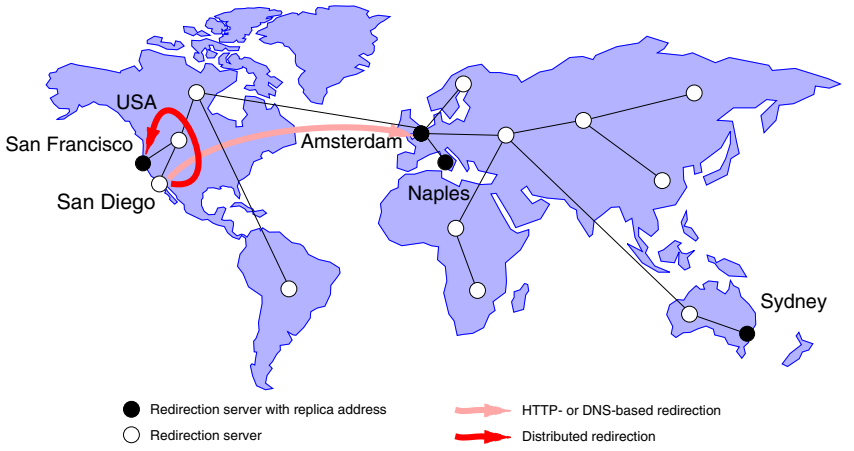

Fig. 2. Building a hierarchy of redirection servers.

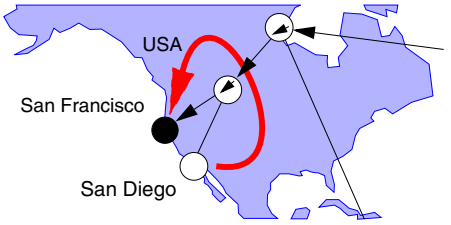

Fig. 3. A forwarding pointer at the USA server.

This scenario shows that by contacting its local redirection server, a client browser implicitly initiates a lookup for a replica at the lowest level of the redirection service. In the best case, the address of a replica can be found at this server (local replica or cached address). If not, the forwarding of the lookup request takes place. Each step up in the hierarchy of redirection servers broadens the search. Having lookups always starting locally at the client site and gradually expanding the search area guarantees that the potential local and close-by replicas are found first. This also guarantees us to find a replica wherever it is stored. The forwarding of the requests along the hierarchy goes no further than necessary and allows us to avoid unnecessary communication with parties that are far apart. In addition, by keeping the number of levels in the hierarchy relatively small, we can also keep the latency minimal when forwarding the requests. Note that this scheme works well even in the presence of updates. Updates to the (contents of the) replicas themselves such as consistency management are of no influence on the redirection service. Updates related to adding or removing a replica and therefore its address in the distributed redirection service are the only visible updates at the redirection-mechanism level. They are scalably handled by the forwarding mechanism.

4. Detailed design

The redirection service relies on two main components: a hierarchical collection of redirection servers and the mechanisms of the redirection server itself. This section details what the hierarchy of redirection servers is and how it is built. It also describes how a redirection server works and how it makes use of the hierarchy.

4.1. A hierarchy of redirection servers

The collection of redirection servers is distributed world-wide and organized hierarchically in such a way that each part of the desired area is taken care of by a redirection server. The redirection servers are themselves organized on a world-wide collection of *redirection hosts*. Each redirection server belongs to a single *domain* and operates at one given level of the hierarchy. In our example, a domain corresponds to a geographical region such as

a city, a country, or a continent, as shown in Fig. 4. A domain therefore carries a notion of locality. The hierarchy takes the form of a tree and is constructed as follow: the leaf domains are aggregated into larger subdomains, which in turn are aggregated as well and eventually the highest-level domain covers the entire network. Each domain is allocated at least one redirection server. However, we can expect multiple redirection servers and hosts per domain. The root domain, for example, is likely to have thousands of redirection servers distributed all over the world. For each Web page, one given server of the collection acts as root server and pages originating from different leaf domains will generally have different root servers, as suggested in [22]. The full redirection service is therefore organized as a collection of trees of redirection servers rather than as a unique hierarchy. Note that this full distribution balances the load across *all* the servers of the redirection service. The hierarchy of domains, however, is unique. Fig. 4 shows the hierarchy of redirection servers for the Globule page. The page has one redirection server in each domain of the hierarchy. Together, they form the hierarchy of redirection servers for the Globule Web page. The root server *World* is run by a host located in Amsterdam, Amsterdam being the home location of the Web page.

For enforcing locality, a server stores information only on replicas that reside in its own domain. The address of a given replica is therefore to be found at *one* redirection server. Moreover, storing addresses only at leaf servers makes it unnecessary to maintain consistency within the redirection service. For example, if the addresses of a given document were stored at random servers—leaf or intermediate servers—it would be necessary to search for addresses each time an update would have to occur. Looking up or deleting an address would imply a tree-wide search respectively for finding the address the closest to the client of for fetching all the copies of the address to be deleted. In contrast, using a single address location per replica—in our case a leaf server—preserves the efficiency of access. Storing addresses only at leaf servers enforces further the locality of accesses and favor local clients: the address in the leaf is close to the clients as they query their local leaf node directly; the address is also close to the replica that was placed in that particular domain since the traffic, in terms of client requests, was sufficiently high. As a whole, the world-wide collection of redirection servers stores the addresses (URLs) of all the replicas of the Web pages willing to participate in the service.

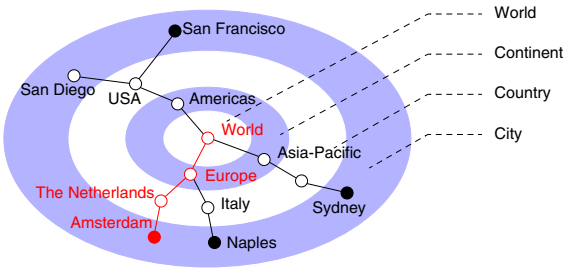

Fig. 4. The hierarchy of domains and a hierarchy of servers.

In order to be able to find a given address starting from any redirection server in the hierarchy, intermediate redirection servers store *forwarding pointers* to other redirection servers located in one of their subdomains. The presence of such a forwarding pointer guarantees that a replica address will eventually be found and that the replica lies in a subdomain of the considered intermediate redirection server. In our example, the USA server holds a pointer to a redirection server in the San Francisco subdomain (see Fig. 3).

The hierarchy of domains of the redirection service reflects geographical locality. However, the locality metrics can also be expressed in terms of network distance such as latency. From now on, we assume that geographical and network distances are equivalent. Of course, this is extremely inaccurate. However, it has an impact only on the way the hierarchy is built. Neither does it influence the locality in the treatment of requests, nor does it change the way we manage the hierarchy of domains or the redirection servers. Choosing another locality metric would only lead us to building the redirection service hierarchy in a different way (see for example research on latency-based topologies [11,13,21]).

Finally, for each Web page, the redirection service is brought up with an initial configuration for the hierarchy of domains, for example, a four-level

hierarchy, as shown in Fig. 4. This initial configuration is a rough estimate of what the redirection service needs once the entire service is up and running. It may be that this initial configuration shows not to be very appropriate and that the locality has to be improved by creating or removing domains.

4.2. The redirection server

A redirection server has to handle two kinds of tasks: answer incoming requests and manage the location information for the replicas to be found in its own domain. The following subsections describe how this is achieved.

4.2.1. Basic request handling

The above scenario showed that a redirection server can receive requests from client browsers or from other redirection servers which we call *client redirection servers*. These requests are known as *lookups*. They do not modify the information stored in the redirection service but allow clients to retrieve addresses of replicas of Web pages. In order to let the Web servers hosting replicas add and maintain replica information, a redirection server also has to support *update* requests. An update corresponds to either an *insert*, which stores the address of a replica in the redirection service or to a *delete*, which removes an address from the redirection service. Each redirection server has to handle these three types of requests.

The technique for handling requests in the redirection service is the same for both updates and lookups (for details, see [2,23]). Requests are always initiated at a leaf redirection server. In the case of update requests, the request is forwarded only upwards. For an insert request, the upper domain has to be contacted to ask permission to store an address. If the permission is not granted, the upper domain has to care for the insertion of the address itself. This could mean that other addresses of replicas of this particular document are already stored at an upper level. This can occur in the case of mobile documents or objects as explained in [2]. In the case of the World-Wide Web, the documents should be fairly static and the permission should be granted. The upper domain then installs a pointer towards its child domain. This happens recursively until a server is reached that already holds information for the considered Web page. In such a case there is no need for forwarding the request any further. It simply means that the upper level already has a forwarding pointer installed. Installing forwarding pointers guarantees that any inserted address can be found following a path of pointers from the root to the server where the address is actually stored.

In the case of a delete request, the upper domain has to be contacted only if the record for the Web page at the current redirection server becomes empty. In such a case, the pointer at the next higher-level server has to be deleted. This recursively happens up to the root redirection server if necessary. This mechanism guarantees that following a path of forwarding pointers always leads to an address and never to an empty record.

Finally, in the case of a lookup, the request is forwarded upwards in the hierarchy until it reaches a redirection server that holds information about the Web page that is being looked up. In the best case, this redirection server is the local leaf redirection server and a replica address is immediately found. When a pointer is found, the lookup request is further forwarded downwards to the redirection server referred to by the pointer and eventually reaches the leaf redirection server that stores the address. The presence of a pointer guarantees us to find an address in one of the subdomains. This address is eventually sent back to the requesting client browser. Section 4.2.3 will describe alternatives for sending back replica addresses.

Each redirection server acts independently when dealing with its contents or with the requests it receives. It makes use of local information as much as possible in order to reduce the communication overhead. However, during an update or a lookup request, another redirection server may have to be contacted. This redirection server can be unreachable because of software or hardware faults. In such a case, the requesting redirection server can make use of a simple fall-back mechanism. Whenever a lookup request takes too long to proceed, the initiating server can take the

decision of contacting directly the home server of the document (i.e., the server whose address is contained in the URL). This fall-back mechanism prevents a client from indefinitely waiting for a redirection server that is currently unavailable. In the case of update requests, the client does not have to wait until the full completion of the request but can get an answer directly after the update has been completed locally.

In addition to the above tasks, a redirection server has to handle the registration of Web servers willing to participate in the service. This encompasses registering replicas of Web pages and offering space for hosting replicas. When a Web server is participating, we assume that all its Web pages are registered in the redirection service. This does not mean that all the Web pages are actually replicated. The replication granularity is not enforced by the redirection service but chosen by the administrator of a participating Web site. This administrator can very well use differentiated replication strategies on a per-document basis, as suggested in [16]. However, registering all the pages of a participating Web site means that for each Web page of a participating Web server, there is at least one reference to the Web page to be found in the redirection service: the original copy of the Web page, located at the home site. We return to the registration procedure in more details in Section 5.2.

4.2.2. Server selection and placement

The placement of the redirection servers is also important. Consider the following example. Fig. 4 shows the hierarchy of servers for the Globule Web page. The root redirection server is best placed on a host physically located in Amsterdam and so is the leaf server Amsterdam. Consider a lookup request for the Globule Web page traveling up to the root server in Amsterdam. At this point, the lookup request should follow the forwarding pointers down to the leaf server in Amsterdam. It would be counter-productive to have this request traveling away from Amsterdam to the European level, hosted for example in Germany, then to the Dutch level, hosted in Rotterdam and finally back to Amsterdam where the leaf server resides. Instead, all the redirection servers for the root, the European level, the Dutch level and the Amsterdam leaf server should be run on the same host located in Amsterdam. Forwarding a lookup along the forwarding pointers on this branch will have a minimal cost (no networks delays). A given host can very well run several redirection servers possibly acting at different levels of the hierarchy. Moreover, these servers can be implemented in a single multi-threaded redirection server, if need be. A given redirection server (or thread) will be identified by both the IP address of its host and a server identifier.

4.2.3. Caching

DNS-based redirection makes use of caching mechanisms to treat subsequent queries on a given DNS domain more efficiently. Caching mechanisms can also be applied to distributed redirection. A redirection server can store the address of a replica it has served for later use. We are considering here only the caching of addresses of replicas of Web pages. The caching of the page itself is out of the scope of this paper.

Depending on whether or not address caches are in use at the redirection servers, update and lookup requests can be handled in three ways. To enable caching at all the levels of the hierarchy, we can follow what DNS calls a recursive scheme. The response to a lookup request follows the same path the query used through the hierarchy. Each intermediate server can cache the resulting address on the way back. In order to put less load on the redirection servers, it is also possible to use an iterative scheme as for DNS. All the requests within the redirection service are thus initiated by the leaf server. It is in charge of successively contacting the redirection servers until the address is found, no forwarding takes place. This scheme makes caching possible only at leaf servers. Alternatively, we can follow a hybrid approach where the query follows a recursive scheme and the reply follows an iterative scheme. That is to say, the redirection server holding the address answers directly to the initiating redirection server. This has the advantage of putting less load on the intermediate servers when the reply is sent back. It is also cheaper in terms of messages and distances traveled by the messages than going through all the intermediate servers.

The time-to-live of each address in the cache is a crucial parameter: a short time-to-live can dramatically reduce the cache hit percentage and make the cache practically useless. A long time-to-live will act no better than HTTP-based redirection when replicas are deleted faster than their addresses in the cache. The user will not keep references to removed replicas, but the cache will without the user being aware of it. Nevertheless, in the case of distributed redirection, the time-to-live value can be provided directly by the Web server hosting the replica. Each participating Web server hosting a replica of a Web page has to fulfill a contract determining precisely what it should store, keep up-to-date and how long it should maintain a replica. This time value can be given to the redirection server where the address of the replica is stored and further used as time-to-live value in the cache of the client redirection server. The standard lookup procedure can therefore be short-cut by using the address cache and it can be guaranteed than an address found in the cache is always valid. It is important to note that the contents of a replica can still change without invalidating the cached addresses.

Enabling the caching mechanisms in the redirection service appears very appealing. Both DNS-based or two-tier DNS redirection are using such caching mechanisms but have the disadvantage that cache entries become stale and that the home Web server still has to be contacted after each cache flush. In the case of distributed redirection, even in the case of a cache miss, we can continue to exploit locality by gradually letting the request travel up in the hierarchy.

4.3. Building hierarchies of redirection servers

We mentioned in Section 4.1 that the redirection service is organized as a collection of hierarchies of redirection servers. Therefore, a given redirection server can be part of several hierarchies, serving for different Web pages or sites. Prior to the forwarding of a request, a redirection server has to select the hierarchy it will use for the considered Web page. In other words, a redirection server has to select the parent to which it will forward the request. The parent selection is achieved using a function mapping the URL of a Web page to a given parent server. The mapping function can be implemented by means of a mapping table or a hash function. This implies that a redirection server maintains a list of its parent servers and can select one among those when needed. Note that the same holds for the child servers: a server has to maintain a list of child servers per hierarchy.

Maintaining a list or a cache of parent and child servers means that the structure of each hierarchy of redirection servers has somehow to be distributed. A hierarchy of redirection servers is potentially composed of hundreds of thousand servers. It is therefore not reasonable to assume that the positioning of the servers in a given hierarchy will be broadcast at once by the root to each server composing the hierarchy. On the contrary, we assume that the hierarchy is constructed on-the-fly. Whenever a redirection server from domain D_i (with $0 < i \leqslant 3$, i.e. leaf or intermediate domain, D_0 being the root domain) receives a request for a page that is not mapped to a hierarchy yet (i.e. has no known parent server), the redirection server in domain D_i contacts the root server and gets the IP address and identifier of the parent server in domain D_{i-1}. Once the parent is known, the request can be forwarded up as described in Section 4.2.1. As an optimization, the requesting redirection server from domain D_i can first select a parent server in domain D_{i-1} and then contact the root. The root will either install the proposed server as server for the domain D_{i-1} or, if a server for the domain D_{i-1} is already known, reject the proposed server and send the IP address and identifier of the correct parent to the server from domain D_i. Note that we assume that the domains are static, i.e. the domain of a server does not change and that a given server from domain D_i knows some servers from domain D_{i-1} so that it can select parent servers if need be.

The selection of parent or child servers can be optimized in two ways. First, whenever the address of a replica of a Web page is installed at a leaf domain D_3, the parent and child servers on the path from the root server to the leaf server in domain D_3 are selected and immediately installed at the relevant servers. Second, the installation of new

parent or child servers can be piggybacked with other messages in order to avoid contacting the root server as much as possible.

The Web page of a non-participating Web site can never be mapped to a hierarchy. This means that a leaf redirection server receiving a client request for such a page will never have a parent server for this given page. In the case of a delete, the request will simply be rejected. In the case of an insert, the leaf server will contact directly the root server and the forwarding of the insert downwards will care for the installation of the forwarding pointers and the installation of the parent server wherever it is appropriate. In the case of a lookup, the leaf redirection server will contact the supposed root of the hierarchy, i.e. the Web page's home server. The leaf redirection server can simply send an HTTP request for the desired Web page. The non-participating Web server will send back the page contents as usual. The leaf redirection server then acts as a proxy and returns the requested page to the client browser. Note that the HTTP protocol can also be used in the case of a participating Web server, only the answer to the request will differ. The root server can decide to return directly the page contents as well as the IP address and identifier of the parent server, the address of a close-by replica as well as the IP address and identifier of the parent server, or only the IP address and identifier of the parent asking the leaf server to contact its parent.

There is of course a tradeoff between the volume of information concerning the hierarchies that we want to disseminate and the latency we add to the client requests when lacking this information. As an optimization of the above protocol, the information concerning the hierarchies can lazily be distributed to the redirection servers and be stored in a local database. Doing so allows us to look for parent servers in the local database thus offloading the root server and removing the parent lookup from the critical path, i.e. the client request. The database can efficiently be implemented by using a modified DNS server. For example, the modified DNS server can store a DNS TXT record along with the domain name of each participating Web server, similarly to what is described in [4]. The TXT record can, for example, contain a parent server IP address and identifier to be used with this particular domain, or a path to the root server. This means that *one* (Web server) domain name is mapped to *one* unique hierarchy. Another domain name, for example a virtual server supported by the same Web server, can use the same hierarchy simply by having the same IP addresses and identifiers stored in the DNS TXT record. It could also be possible use a finer-grain mapping by storing URL prefixes in the DNS TXT record, allowing the use of several hierarchies for one given domain name. However, we do no think this is appropriate in the case of the Web servers.

5. Interacting with the redirection server

A redirection server can interact with three kind of entities: Web clients such as browsers, Web servers and other redirection servers. In Section 4, we described the possible interactions between redirection servers. This section presents how a browser and a Web server can interact with a redirection server and how the redirection server is integrated into the Web environment, avoiding as much as possible modifications at the end-user side.

5.1. Web-client and redirection-server interaction

The client browser interacts with the redirection server to look for replicas of Web pages. This interaction has to occur as transparently as possible. As such, we have decided to use only well-known and widely-used protocols such as HTTP and DNS.

First, a client should not be aware it is dealing with a replica of the Web page it requested. Therefore the client should not be able to keep an explicit reference towards a replica of a Web page, for example by letting its user bookmark it. Not preventing this can lead to dangling pointers when the replica is removed. This is unacceptable if the original page is itself still accessible. Furthermore, discovering new replicas closer to the client would require an explicit action from the end user, which is also unsatisfying. This is what we call *replication transparency*.

It is the task of the client's redirection server to maintain replication transparency. This is achieved by not displaying the addresses of the replicas to the client browser. The client sees only the original location (home location) of a Web page. It has no way of discovering the address of a replica when using the redirection service and subsequently cannot access the replica directly. The redirection server at the client side therefore takes care of fetching the replica of the Web page for the client and acts towards the client as if it were the original Web server.

Second, a client browser should not be aware its requests are going through a redirection service. The end user should have to do the least possible to take benefit of the redirection service. This makes the deployment and the use of the redirection service at client sites easier. We can achieve this by carefully integrating the components of the redirection service with the existing environment, Web applications and protocols. This is what we call *transparency of use*.

To satisfy the transparency of use requirement, we decided to integrate our distributed redirection scheme with the DNS service at the client site. A distributed redirection server has therefore to act as authoritative DNS server. A client browser automatically accesses the redirection service by contacting its authoritative DNS server, as it is the case with today's DNS-based redirection. Note, however, that the use of DNS in this context remains confined to the client site. Fig. 5 shows the message exchanges between the client browser, its authoritative DNS server (component of the redirection service) and the local redirection server. Note that this authoritative DNS server can also be charged of mapping URLs to hierarchies of redirection servers.

Let us see how an end-user request for loading the Globule page is handled when using the redirection service. For loading the page, the browser must achieve two tasks. First, it has to resolve the DNS-domain name www.globule.org into the IP address of a Web server by contacting its authoritative DNS server. Second, it has to construct an HTTP request for the page and send it to the Web server whose address was returned. Since redirection has to take place locally at the client side, the redirection service has to be integrated in between these two steps. We decided to act at the DNS level (this choice is discussed in Section 7). We let the client's authoritative DNS server resolve the browser's DNS request into the address of its local redirection server (see Fig. 5, message exchange 1). Without noticing it, the client browser is therefore asked to contact the redirection service which it believes to be the Web server of the Globule page. This approach realizes

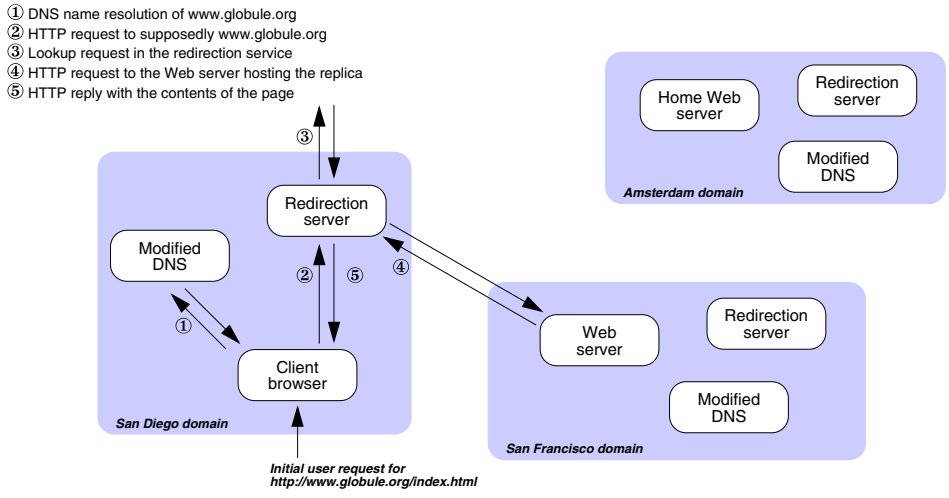

Fig. 5. Components used for distributed redirection.

transparency of use. As for the second step, the browser sends an HTTP request to its local redirection server (message 2) which looks for a replica as explained in Section 4 (message exchange 3). The lookup can recursively lead to several message exchanges with other redirection servers (not shown in the figure). The client's local redirection server is in charge of fetching the requested Web page (message exchange 4) and returning it to the client (message 5). Using an HTTP-redirect at this stage would violate replication transparency. The redirection server at the client side therefore takes care of fetching the replica of the Web page for the client and acts towards the client as if it were the original Web server. Note that Web proxies display a similar behavior. Moreover, the HTTP reply the redirection server sends back to the browser has to use the original URL to designate the page and thus preserve replication transparency. It also means that the URLs contained in a replicated Web page are not rewritten to match the location of the replica. Any subsequent request goes through the redirection server and is not bound to a given replica. Rewriting would even be counter-productive in the presence of caching at the redirection server side.

Fig. 5 shows the components taking part in the redirection service as well as the message exchanges between these components. Four entities are used: the *Web browser* (the client), a modified *DNS server*, a *redirection server* and a *Web server*. The modified DNS server and the redirection server are both installed at each participating client and server sites. A Web server site is said to participate if it hosts replicas of Web pages or has replicas hosted at some other sites. A client site is said to participate if it has a modified DNS server and a redirection server locally installed. Fig. 5 shows the case where no Web proxy is installed in the browser configuration.

The servers necessary for the redirection service can be integrated into a single component. The modified DNS server may just be a front-end to the redirection server, as shown in Fig. 6. The redirection server component has therefore to act as (1) a DNS server, resolving DNS names; (2) a Web proxy, receiving client browser requests and fetching replicas of pages from Web servers on behalf of client browsers; and (3) a redirection server, performing lookups in the redirection service.

In our example, the Globule Web page was known from the redirection service and a replica address was eventually found. It may, however, happen that no replica of the requested Web page is found in the redirection service. Since all the Web pages of a participating site are registered in the redirection service, this can only mean that the page has disappeared from the home Web server. In such a case, the client's redirection server simply sends back an appropriate HTTP error code. In the case of a non-participating Web site, the lookup never gets initiated and therefore does

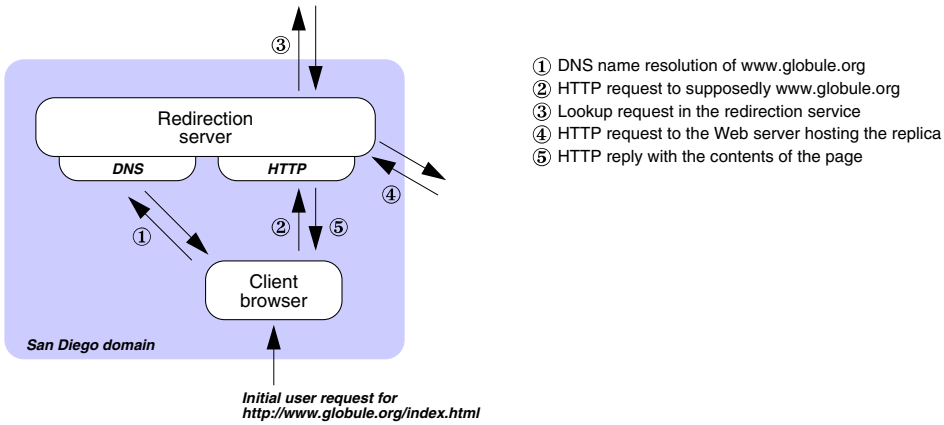

Fig. 6. Single redirection server component.

not lead to a lookup miss. Instead, an HTTP request is issued and sent to the supposed root (i.e. the home server). The client's local redirection server then acts as a regular Web proxy and eventually forwards the reply to the client.

Note that when receiving an HTTP client request, a redirection server may have to resolve the domain name contained in the request. This happens only if this particular domain name cannot directly be mapped to a hierarchy by the client's local redirection server. This means either that the home Web server is not participating in the redirection service or that the parent server for this hierarchy is still unknown. During the name resolution, extra information can also be found about the hierarchy, most likely a parent server or a path to the root (see Section 4.3). If a parent server is found, the client's redirection server issues a lookup request in the hierarchy, otherwise, it uses the resolved domain name to contact the home Web server.

An optimization can be applied to the basic redirection protocol. We saw that the DNS, proxy and redirection servers can be integrated into one single entity. In such a case, the client browser exchanges two sets of messages with the same entity but using different protocols. We can improve efficiency by relaxing the transparency constraint: we can let the client browser consider the redirection server as a Web proxy (see Fig. 7). The browser proxy configuration has thus to be adjusted accordingly. However, to preserve transparency, the redirection service could be directly integrated at the router or switch level, as it is the case for transparent caches. It would then intercept TCP traffic for port 80. In the proxy configuration, the protocol does not fundamentally differ from previous cases. The browser has only to send an HTTP request to the redirection server which performs the subsequent lookup and DNS name resolution if necessary. For more details about implementation considerations and deployability, we refer the reader to [1,14,15].

5.2. Web-server and redirection-server interaction

A Web server has to declare all the pages it wants to replicate and how much storage space it is offering for the replicas of other Web servers. To do so, the Web server has to contact its local redirection server. The latter is in charge of inserting the addresses of the declared Web pages so that they can be located by others. We call this procedure *registration*. At the end of the registration process, the Web server is considered as a participating server. This implies that it is to be found in the database of participating sites which the redirection servers keep, that the Web pages that were

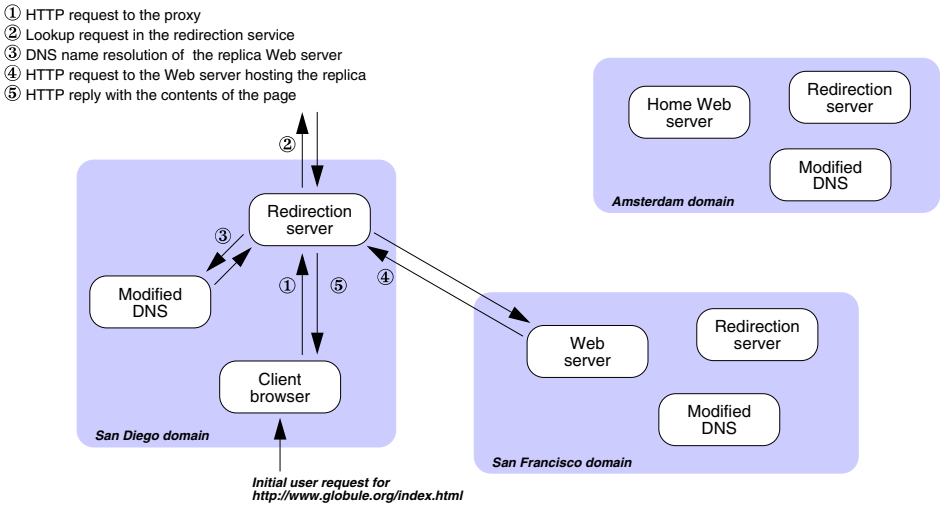

Fig. 7. Redirection server configured as a proxy.

declared are automatically replicated and that the addresses of the replicas can be looked-up at client request.

In addition to dealing with Web server registration, a redirection server has to handle update requests concerning the Web servers' pages. That is to say, the administrator of a participating Web server may want to let new pages be replicated and add replicas to the redirection service or instead remove some. This can be achieved using the update operations described in Section 4. It is likely that both registration and management of registered Web pages (update requests) will be done via a separate management tool serving as a friendly user interface to the local redirection server. The description of this tool is out of the scope of this paper.

6. Simulating distributed redirection

To validate the distributed redirection mechanism and estimate the overheads and gains of the method, we conducted a series of simulations. One important aspect is to evaluate the gain in user-perceived latency booked with distributed redirection compared to directly accessing the home location of a Web page. Another important point is to show that the load on the home Web server decreases and that this load is evenly distributed among distributed redirection servers.

The simulation is trace-driven and uses HTTP traces from our department Web server `www.cs.vu.nl` at the Vrije Universiteit. It spans more than one year. Based on the IP addresses found in the HTTP trace, the simulator sets up a hierarchy of redirection servers rooted at the Vrije Universiteit for the `www.cs.vu.nl` Web server. Beforehand, we estimate the latency between each pair of parent and child redirection servers using a modified version of King [9]. The address of each Web page is inserted at the leaf server for `www.cs.vu.nl`. Addresses of replicas are inserted at leaf servers where there is enough demand. Finally, each client (i.e. IP address) from the HTTP trace is associated to its leaf redirection server. In other words, a client request is assumed to originate from the client local leaf server. The simulator does not require to map each client IP address to a leaf server. A detailed description of the experiment can be found in [3]. Again we emphasize that we wish to support a fine granularity in redirection, as HTTP-based redirection does, while maintaining scalability and transparency. In that sense, we want to combine the benefits from both HTTP- and DNS-based redirection mechanisms (DNS or two-tier DNS redirection) into a single scheme.

Once the hierarchy is constructed and replicas are placed, the simulator replays the clients requests found in the HTTP trace. For each request, it logs the latency induced by traversing the hierarchy, the number of hops the request traveled in the hierarchy and at which server the address of the requested document was found. In addition, each redirection server records the total number of operations it had to handle during the trace replay. By operations, we mean either a lookup initiated directly by a client or a lookup forwarded in the hierarchy of redirection servers. This gives us an estimate of the load of each redirection server. We ran several simulations using the same hierarchy but different replica placements.

Fig. 8 shows the results of the simulation for a trace were all the documents are replicated. The policy for placing replicas is as follow: each document requested two times by the same leaf server gets replicated at that location. This replication threshold is of course not realistic in a real environment but allows us to get a large number of replicas from our Web server trace, which is mandatory for testing the benefits of the distributed redirection. However, this low replication threshold only influences the placement of the replicas and not the redirection mechanism in itself. In addition, once the replicas are placed, we extract from the HTTP trace all the requests concerning documents requested just once (i.e. not replicated). These requests are not replayed during the simulation. We rejected in total close to 6% (3,667,549) of the requests (over 61,670,630) and 61% (1,596,990) of the documents (over 2,615,952). In the following, we refer to this simulation as "*selected documents, threshold 2*".

Fig. 8 shows that more that 34% of the documents addresses are served locally (latency equals

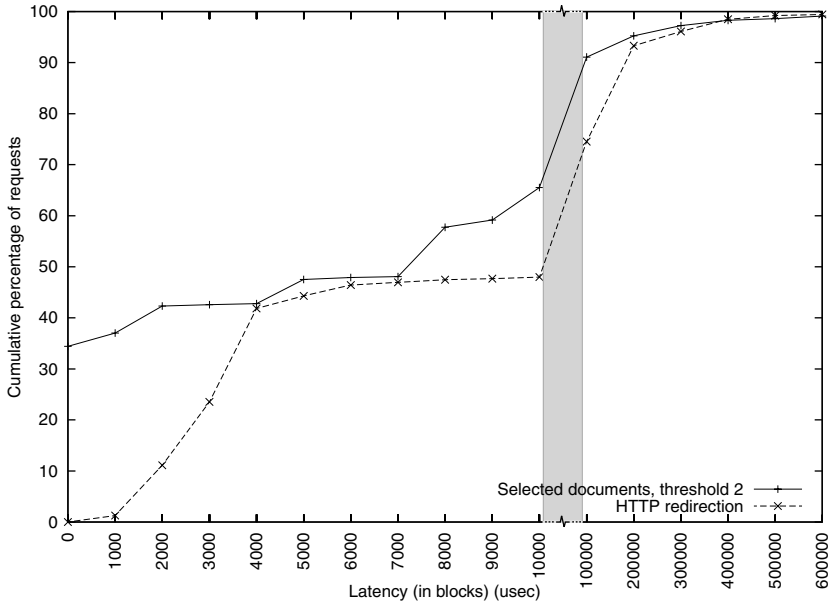

Fig. 8. Proportion of requests serviced within a given latency for the "*selected documents, threshold 2*" experiment.

zero) and that close to 43% of the addresses are found with a latency less than 4000 μs. Past this latency threshold, distributed redirection and HTTP redirection (i.e. direct access to the home Web server) perform similarly. Note that only the latency necessary to the redirection is taken into account and not the latency necessary for fetching the document. The latter depends on the location of the selected replica. In the case of HTTP-based redirection, the redirection part boils down to a direct access to the home Web server; with distributed redirection, to a direct access to the closest distributed redirection server holding a replica address for the requested document.

We compare our approach to HTTP-based redirection for two reasons. First, we wish to compare our approach to one that offers redirection at the granularity of Web pages, just like ours. Only HTTP redirection does this. Second, HTTP redirection provides the minimal number of hops, namely only one hop to the home Web server, after which the document can be returned. If our approach performs better in terms of latency, we will certainly demonstrate an improvement.

In order to evaluate the effect of rejecting requests, we performed a simulation with the same replica placement as used in "*selected documents, threshold 2*" but without rejecting any HTTP requests from the trace. In the following, we refer to this simulation as "*all documents, threshold 2*". Fig. 9 compares the output of "*selected documents, threshold 2*" with "*all documents, threshold 2*". It shows that including documents having no replica and being requested only a few times (approximatively 6% of the total number of requests) does not affect the overall performance of the distributed redirection.

Fig. 10 shows the result of a simulation in which we increased the replication threshold to five requests. It leads to a simulation setting where close to 90% of the documents are not replicated. With respect to the distributed redirection mechanism, this is a worst-case scenario. The majority of requests will travel through the hierarchy, up to the root server, thus accumulating latencies. In the following, we refer to this simulation as "*all documents, threshold 5*".

Fig. 10 shows that even with a small number of replicated documents, the distributed redirection performs better than HTTP redirection up to 3000 μs. Close to 16% of the documents addresses are served locally (latency equals zero) and at

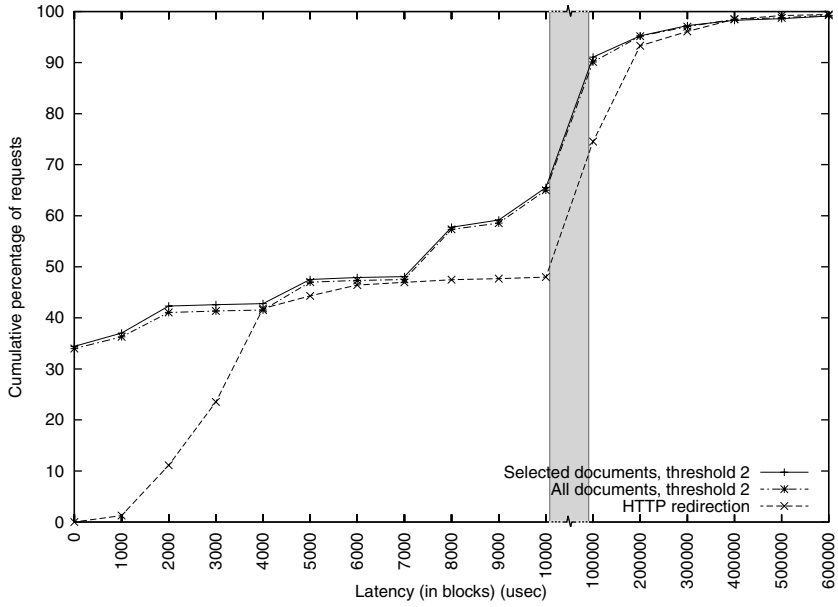

Fig. 9. Proportion of requests serviced within a given latency: comparison of the "*selected documents, threshold 2*" and "*all documents, threshold 2*" experiments.

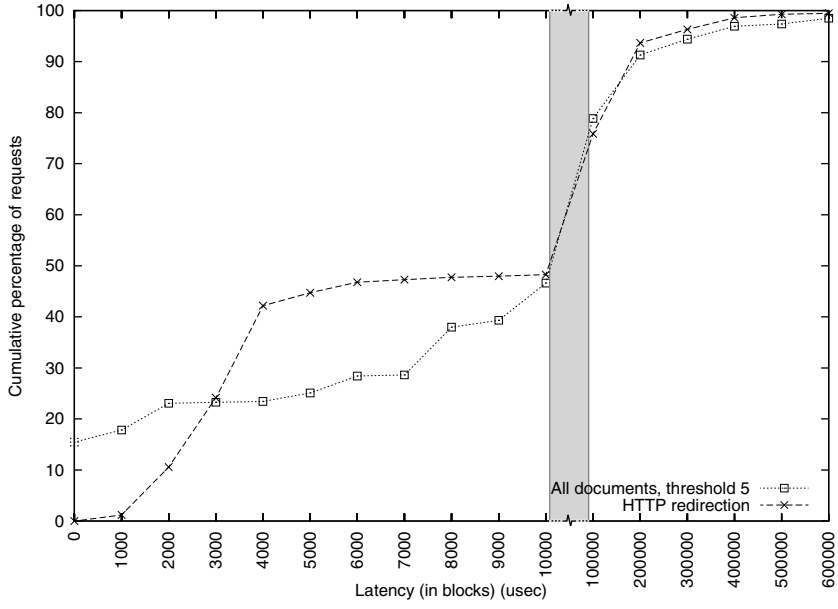

Fig. 10. Proportion of requests serviced within a given latency for the "*all documents, threshold 5*" experiment.

3000 μs, 23% of the requests have been served against 24% for HTTP redirection.

Fig. 11 shows the number of hops for the different simulations. We count hops as follow: one hop

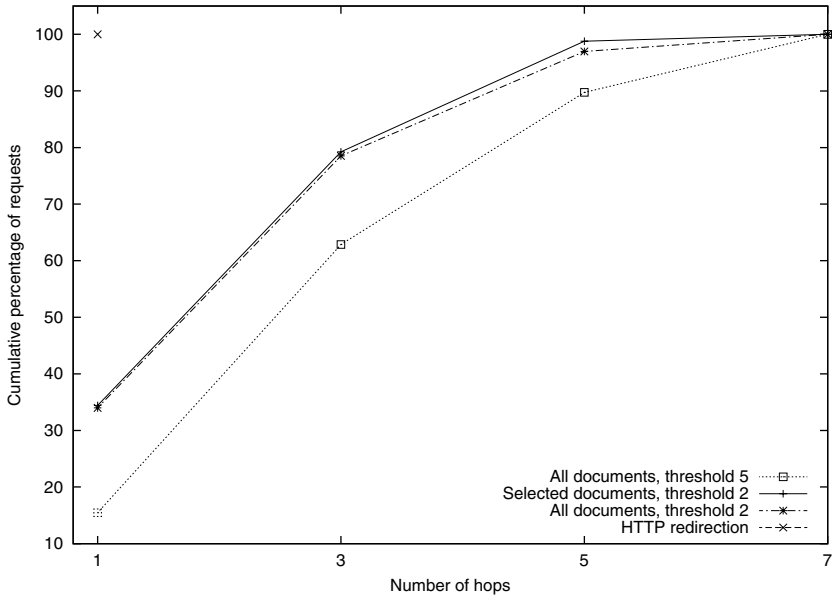

Fig. 11. Proportion of requests serviced within a given number of hops.

is counted for transferring the HTTP request from the client browser to its leaf redirection server. After that, each time a redirection server forwards the lookup request to its parent server, one extra hop is added. In the case of HTTP redirection, the number of hops is always one.

The conclusions we can draw from the different simulations is that most of the requests served in few hops (one or three) show a redirection latency smaller than the one of HTTP redirection. They are likely to improve the user perceived latency, assuming the Web server of the target replica serves the request with a reasonable latency. The lookups requiring more hops in the hierarchy of redirection servers are too costly and do not really improve the performance experienced with a direct access to the home Web server.

Figs. 12 and 13 show the latency for the "*selected documents, threshold 2*" simulation when including server time. As an estimate for the server time, we use either 1000 μs or 3000 μs, respectively referred as optimistic or pessimistic. For each replayed request, we multiply the number of hops by the server time.

Fig. 12 shows the output in the case of an optimistic server time. The curves are slightly shifted compared to Fig. 8. Close to 34% of the requests are now serviced in 1000 μs. The distributed redirection still performs better than HTTP redirection up to 5000 μs (more than 42.3% of the requests in the case of distributed redirection, 41.8% in the case of HTTP redirection). As previously, past this point, distributed redirection and HTTP redirection perform similarly.

Fig. 13 shows the output in the case of a pessimistic server time. The client-perceived redirection latency still remains interesting for the requests served locally (34%). However, the curves show that the server time affects the results quite heavily when a request has to visit several redirection servers.

Let us now concentrate on the overall load placed on the distributed redirection servers. In the following, we consider the "*selected documents, threshold 2*" simulation. Our simulations show that the load on the various distributed redirection servers is not balanced. The load here is measured in terms of number of processed operations. The imbalance comes from the fact that the clients' HTTP requests in our Web server trace are themselves not balanced. The most loaded leaf redirection server is the VU leaf server since most of the clients are located at the Vrije Universiteit and

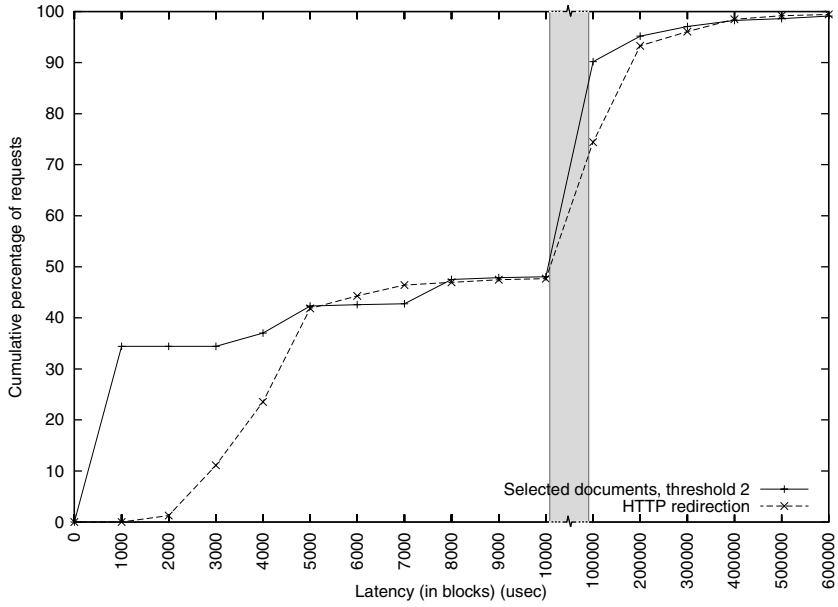

Fig. 12. Proportion of requests serviced within a given latency with optimistic server time for the "*selected documents, threshold 2*" experiment.

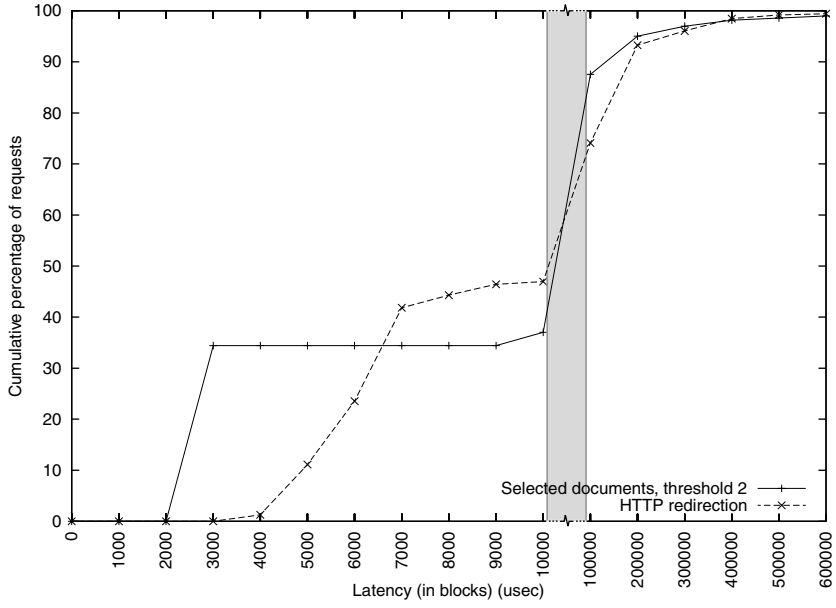

Fig. 13. Proportion of requests serviced within a given latency with pessimistic server time for the "*selected documents, threshold 2*" experiment.

because we did not replicate documents *within* the Vrije Universiteit's network. However, the load on the distributed redirection servers is decreasing when we reach higher levels (root, continent) in

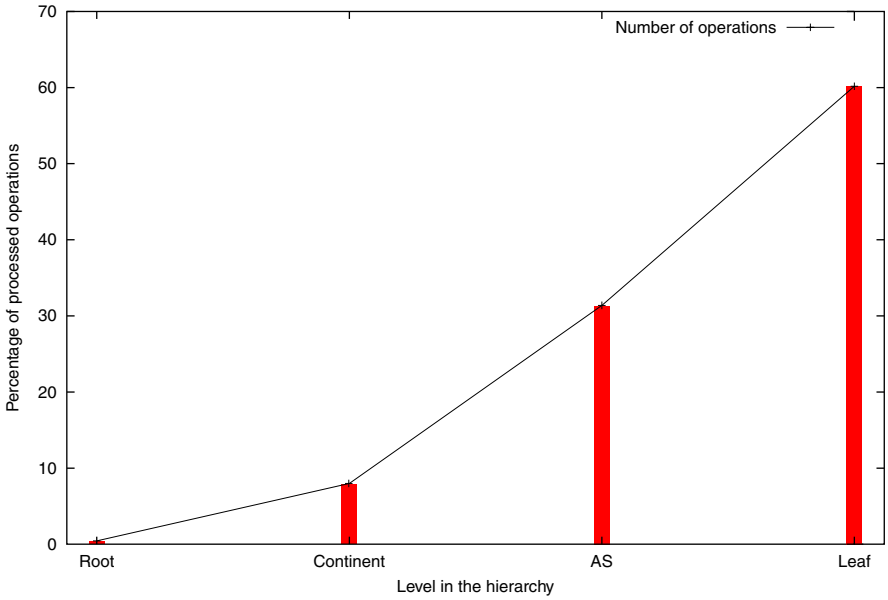

Fig. 14. Distribution of operations among the different levels for the "*selected documents, threshold 2*" experiment.

the hierarchy. The root server gets to see only a small proportion of the client requests, namely 0.12%. Fig. 14 shows the repartition of operations in the different levels of the hierarchy.

The conclusions we can draw from the above figures is that while the load is not evenly distributed among the (leaf) redirection servers, we still offload the root server. Getting the overall load evenly distributed may mean reconfiguring the distributed-redirection servers hierarchy on the fly: some domains are clearly too large (in number of clients) and some others are too small. This comes from the fact that the hierarchy of redirections servers has been built using geographic and routing properties and not by looking at the access patterns of the clients. The placement of the replicas of course also has an impact on the distribution of the load. This means that installing a new replica close to a set of clients may trigger a reconfiguration of the hierarchy in that particular zone.

7. Discussion concerning the design

In the context of the Web, there are a number of alternatives for intercepting and redirecting queries. The level at which the redirection is performed can also vary: browser, proxy, DNS, home Web server. This section justifies why we have opted for the DNS level.

One of our goals is to achieve transparency of use. An end-user should have to do the least to benefit from distributed redirection. In addition, it should be easy for the administrator of a local network to install and maintain the distributed redirection components.

Integrating the redirection mechanisms in a browser is not an option with respect to transparency of use: forcing users to install a customized browser is anything but transparent. Moreover, customizing a browser implies a major effort in developing and releasing new versions, whether the browser is developed from scratch or based on existing publicly available source code. For these reasons, our redirection service is not directly integrated into a browser. However, this would be the best option with respect to efficiency and integration in the current Web. It would also be the best solution to accurately position the client hosts and therefore to enforce a better locality when redirecting requests (in our design, the location of the client is assimilated to the location of its authoritative DNS server).

Since we want the redirection decision to be taken as close to the client as possible, redirecting at the home Web server is not ideal as the home server can be located far away from the client. Moreover, a home Web server can redirect client requests to any suitable replica and not necessarily the closest. This depends on the home Web server own redirection policy. We consider that redirecting at the home Web server side is already "too late". This method, therefore, does not provide a satisfactory solution. It can, however, be used as a fall-back mechanism if a redirection server becomes unreachable.

Working at the DNS level has the advantage that no end-user has to take special actions to use the redirection service. The authoritative DNS server for the client's domain is a modified server that takes care of the redirection or redirects the client to a separate process. The disadvantages are that any DNS request at the client side will go through the modified DNS and that the client side administrator has of course to install the redirection server components and configure them as authoritative DNS server.

Finally, the Web proxy approach offers a slightly less transparent solution. An end user has to configure his browser: either he explicitly sets its proxy or he specifies that he wants to use the standard proxy configuration at his site. The administrator has of course to install the redirection server as proxy. As presented in Section 5.1, the DNS and proxy approaches can be easily combined. The end user is then free to choose whether he wants to configure his browser or benefit from a fully transparent distributed redirection mechanism.

8. Conclusion and future work

The redirection mechanisms used in today's World-Wide Web such as HTTP-based redirection, DNS-based redirection or TCP handoff exhibit characteristics that make them not fully satisfactory. The main concern is that with any of these methods, a *home-based* approach is used. The request of a client is in most cases redirected only after it has reached the Web page's home location. Not only does this put a load on the home Web server and does it generate traffic on the network, it also induces latency that can be perceived by the end user.

We devised a scheme where the redirection is fully distributed and combines the benefits of HTTP- and DNS-based redirection. An important aspect is that network locality is preserved: the redirection decision takes place as local to the client as possible and the selected replica of the requested Web page remains close to the client. In such a way, we avoid unnecessary communication for both finding a replica and contacting it. Latency is kept low. This lets the system scale well, while there is no need for temporal locality anymore as it is the case with DNS-based redirection schemes. Distributed redirection also provides a fine-grained redirection mechanism as HTTP-based redirection does, while preserving transparency: a user is never aware that his requests are being redirected.

Distributed redirection makes use of a world-wide collection of redirection servers organized as a collection of trees, one per Web page or group of Web pages from the same leaf domain. Leaf servers store addresses of replicas and perform lookup requests on behalf of clients. A redirection server supports both DNS and HTTP protocols for interacting with clients, as well as its own protocol for looking up and updating addresses of replicas. Each participating client or server site has to run its own redirection server.

Future work encompasses additional experiments and performance measurements of our redirection scheme, for example, by comparing it with DNS redirection and two-tier DNS redirection. DNS redirection mechanisms benefit from caching. This means that we need to enable the caching of replica addresses in the distributed redirection service as well as in the simulator. Experiments will include various scenarios with which we will evaluate the influence of the time-to-live of the DNS cache entries for DNS-based and two-tier DNS redirection. We will compare these results with the ideal time-to-live value used in distributed redirection (the real time-to-live of the replica of the document specified at installation time). Using a simulation again, we will coun-

ter the non-reproducibility effect implied by using caches. We expect to show that using the real time-to-live value of a replica significantly benefits to the user.

Further extensions relate to making the hierarchy of distributed redirection servers more dynamic, for example to let the redirection servers adapt their load. Our experiments have shown that the load is not balanced among the servers, simply because the requests are not balanced. This comes from the fact that the hierarchy of redirection servers has been built using geographic and routing properties and not by looking at the access patterns of the clients. The placement of the replicas of course also has an impact on the distribution of the load. This means that installing a new replica close to a set of clients may trigger a reconfiguration of the hierarchy in that particular zone.

The distributed redirection mechanisms have to integrate seamlessly in the current World-Wide Web. A redirection server will run as an Apache module and be used transparently by being configured as an authoritative DNS server. We are currently in the process of implementing such a module, which aims at being integrated with another Apache module supporting document replication currently in development within the Globule project [14]. The replication module is using a more peer-to-peer approach. This is another motivation for making the hierarchy more dynamic and trying to avoid as much as possible to distribute information about the hierarchy. Hints about bringing dynamicity into the hierarchy are given in [1].

Integration with the current World-Wide Web also encompasses supporting dynamic Web documents. Commercial services such as online stores do not deliver static Web pages but generate them based on history of requests, client profiles and request parameters. Such dynamic Web documents are composed of both code (e.g. EJBs, CGI scripts, PHP, ASPs) and data stored either in databases or files. Replicating such pages requires replicating both the application code and its data.

The complexity of the replication mechanism lies in the trade-off between fast access to a dynamic page and maintenance of the consistency of the replicated data. In other words, the overhead of maintaining the consistency of the data should not counter-balance the benefits of replicating the dynamic page. As such, replicating the data everywhere is not suited for applications with a high percentage of data updates.

Akamai, for example, tackles this problem by enabling fragment caching [6]: the responses for popular requests are cached, which means that the dynamic document need not be regenerated but is simply retrieved from the cache. This mechanism is suitable for requests that do not modify the application data and are not unique. An alternative to fragment caching is to use on-demand application replication as proposed in [19]. On-demand application replication replicates (chunks of) data only where frequently accessed. This reduces the data-consistency management overhead while improving the user-perceived latency. The mechanism ensures strong consistency between the replicas (code and data) and is fully transparent to both the user and the application programmer. On-demand application replication can be further combined with fragment caching as suggested in [19] allowing the system to perform well for a wide range of application workloads and access patterns.

Our redirection technique is orthogonal to replication mechanisms. On-demand application replication as proposed in [19] uses a DNS-based redirection mechanism that can be combined with distributed redirection to retrieve code, data or even fragments. It is, however, important to note that the time-to-live of a fragment and therefore its address in the distributed redirection service will be considerably shorter than regular replicas of static or dynamic pages. As such, fragment addresses will generate more update traffic in the distributed redirection service. Furthermore, the naming scheme for accessing fragments or replicas (both code and data) in the distributed redirection service has to take the structure of the URLs of dynamic pages into account. As such, the redirection service does not access a document using its full URL anymore but only its program name (e.g. CGI, PHP). The parameters of the program listed in the URL are not considered for retrieving or inserting pages in the distributed redirection service.

Finally, another possible extension of the distributed redirection scheme would be to support replica or object mobility. In such a case, it could be necessary to store addresses also at intermediate nodes and not only at leaf servers. For a highly mobile object, the mobility pattern can be analyzed and the intermediate server storing its address be strategically chosen on the path of the object, as proposed in [2]. Internal mechanisms for supporting mobility are partly present in the distributed redirection service, for example when a leaf server willing to install the address of a replica in the redirection service requires permission to store the address. Additional mechanisms for moving addresses in the hierarchy and gathering information about mobility patterns are described in [2].

References

[1] A. Baggio. Distributed redirection for the Globule platform. Technical Report IR-CS-010, Vrije Universiteit, Oct. 2004. Available from: <http://www.cs.vu.nl/globe/techreps.html>.

[2] A. Baggio, G. Ballintijn, M. van Steen, A.S. Tanenbaum, Efficient tracking of mobile objects in Globe, The Computer Journal 44 (5) (2001) 340–353.

[3] A. Baggio, M. van Steen. Distributed redirection for the World-Wide Web (extended version). Technical Report IR-CS-009, Vrije Universiteit, Oct. 2004. Available from: <http://www.cs.vu.nl/globe/techreps.html>.

[4] A. Bakker, I. Kuz, M. van Steen, A.S. Tanenbaum, P. Verkaik, Global distribution of free software (and other things), in: SANE, Maastricht, The Netherlands, May 2002.

[5] B. Cain, A. Barbir, F. Douglis, M. Green, M. Hofmann, R. Nair, D. Potter, O. Spatscheck, Known CN request-routing mechanisms, Internet draft, 2002.

[6] J. Challenger, P. Dantzig, A. Iyengar, K. Witting, A fragment-based approach for efficiently creating dynamic Web content, ACM Transactions on Internet Technology (2005).

[7] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, B. Weihl, Globally distributed content delivery, IEEE Internet Computing 6 (5) (2002) 50–58.

[8] R. Fielding, J. Gettys, J.C. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol–HTTP/1.1, RFC 2616, June 1999.

[9] K.P. Gummadi, S. Saroiu, S.D. Gribble, King: Estimating latency between arbitrary Internet end hosts, in: Proceedings of the SIGCOMM IMW 2002, Marseille, France, November 2002.

[10] P. Mockapetris, Domain names—concepts and facilities, RFC 1034, November 1987.

[11] T.E. Ng, H. Zhang, Predicting Internet network distance with coordinates-based approaches, in: Proceedings of the 21st INFOCOM Conference, New York, NJ, USA, June 2002, pp. 170–179.

[12] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. Nahum, Locality-aware request distribution in cluster-based network servers, in: Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, October 1998, ACM, pp. 205–216.

[13] M. Pias, J. Crowcroft, S. Wilbur, S. Bhatti, T. Harris, Lighthouses for scalable distributed location, in: Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), Berkeley, CA, USA, February 2003.

[14] G. Pierre, M. van Steen, Globule: a platform for self-replicating Web documents, in: Proceedings of the 6th International Conference on Protocols for Multimedia Systems, Enschede, The Netherlands, October 2001, pp. 1–11.

[15] G. Pierre, M. van Steen, Design and implementation of a user-centered content delivery network, in: Proceedings of the Third IEEE Workshop on Internet Applications (WIAPP 2003), San Jose, CA, USA, June 2003.

[16] G. Pierre, M. van Steen, A.S. Tanenbaum, Dynamically selecting optimal distribution strategies for Web documents, IEEE Transactions on Computers 51 (6) (2002) 637–651.

[17] J. Postel, J. Reynolds, File transfer protocol, RFC 959, October 1985.

[18] M. Rabinovich, O. Spastscheck, Web Caching and Replication, Addison-Wesley, 2002.

[19] S. Sivasubramanian, G. Pierre, M. van Steen, Replicating Web applications on-demand, in: Proceedings of the IEEE International Conference on Services Computing, Shanghai, China, September 2004.

[20] W. Stevens, TCP/IP Illustrated, vol. 1: the Protocols, Addison-Wesley, 1994.

[21] M. Szymaniak, G. Pierre, M. van Steen, Scalable cooperative latency estimation, in: Proceedings of the Tenth International Conference on Parallel and Distributed Systems (ICPADS), Newport Beach, CA, USA, July 2004.

[22] M. van Steen, G. Ballintijn, Achieving scalability in hierarchical location services, in: Proceedings of the 26th International Computer Software and Applications Conference (CompSac), Oxford UK, August 2002, pp. 899–905.

[23] M. van Steen, F. Hauck, G. Ballintijn, A. Tanenbaum, Algorithmic design of the Globe wide-area location service, The Computer Journal 41 (5) (1998) 297–310.

Aline Baggio is currently a postdoctoral fellow at the Delft University of Technology. Her research focuses on localization algorithms for wireless sensor networks and support for sensor mobility. From 1999 to 2004, she worked at the Vrije Universiteit in Amsterdam. There, she concentrated on adding support for mobility and replication in distributed location services. An extension of this work has been applied to the World-Wide Web under the form of distributed redirection. She received her Ph.D. in Computer Systems in 1999 from the University Pierre et Marie Curie in France.

Maarten van Steen is full professor at the Vrije Universiteit Amsterdam. His research concentrates on large-scale distributed systems, notably content delivery networks and gossip-based peer-to-peer systems.