Distributed redirection for the World-Wide Web (extended version)

Aline Baggio and Maarten van Steen

October 2004
Technical report IR-CS-009

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.

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 the replicated Web pages. Instead, it is often necessary to explicitly *redirect* clients towards a given replica.

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 in a *home-based* way. A client is redirected only after its request has reached the *home* server, that is to say, the host named in the document's URL. The decision where to redirect a client to is therefore centralized.

One important disadvantage of centralized redirection mechanisms 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 principle 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 the benefits and drawbacks of our redirection scheme with respect to other Web redirection mechanisms. And finally, Section 8 concludes and gives some future work directions.

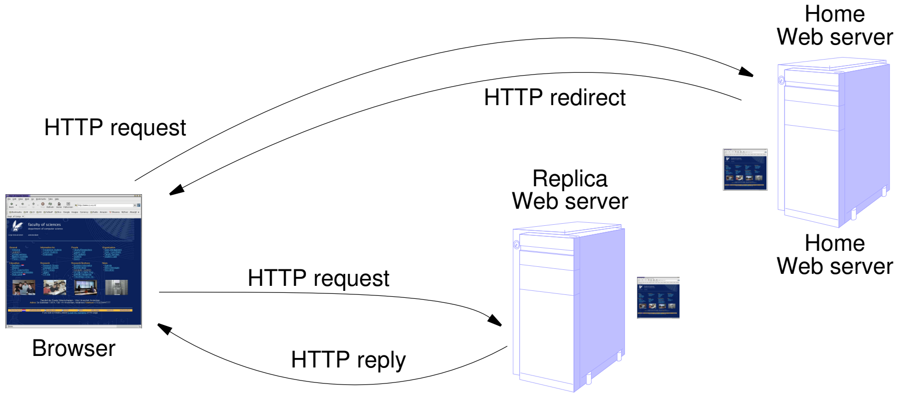


Figure 1: HTTP-based redirection

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 specifying the server where a replica of the requested page can be found [5, 9] (see Figure 1). 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 [9]. 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, 15, 23] 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 (Figure 2), the later as TCP splicing (Figure 3).

Finally, the Domain Name System (DNS) can be used for redirection purposes [5, 13, 21]. 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) (see Figure 4). 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 [8] redirects a client in two steps. The client's DNS request reaches first the home DNS server, as in the previous case. The client's DNS is then asked to contact a second DNS server closer than the home DNS server. This second DNS server chooses

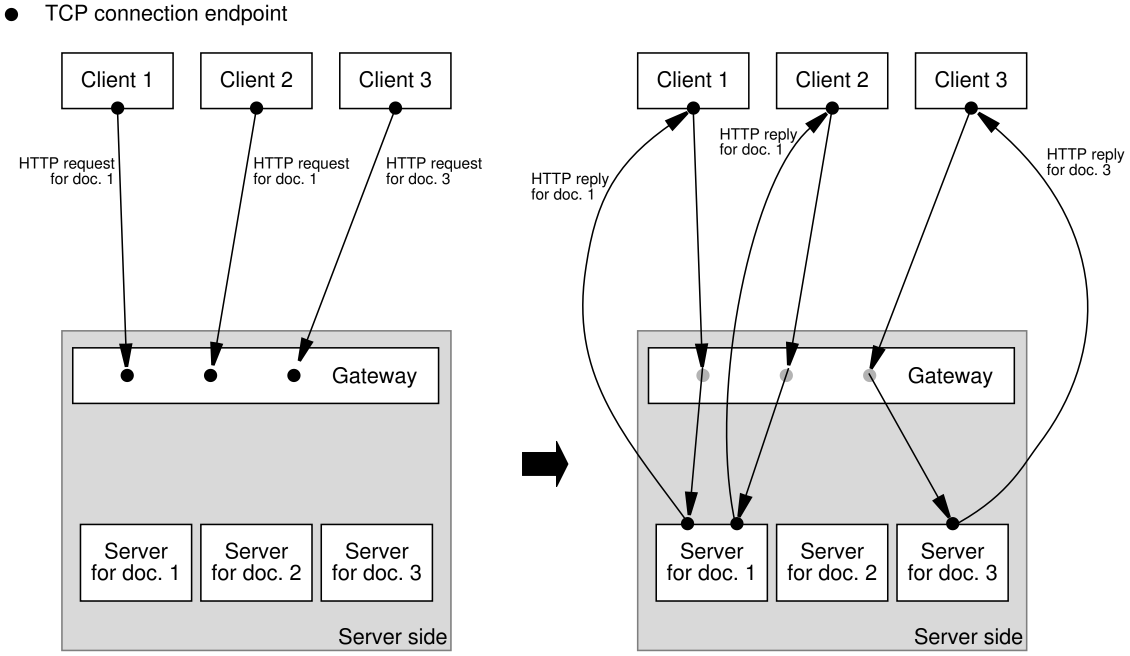


Figure 2: TCP handoff

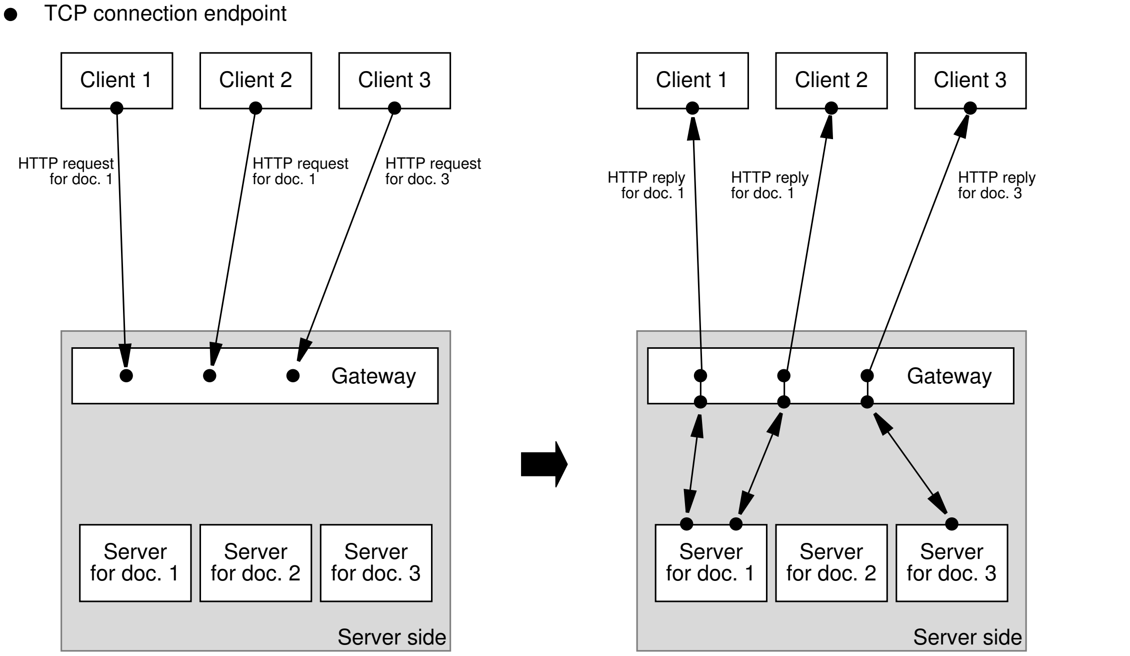


Figure 3: TCP splicing

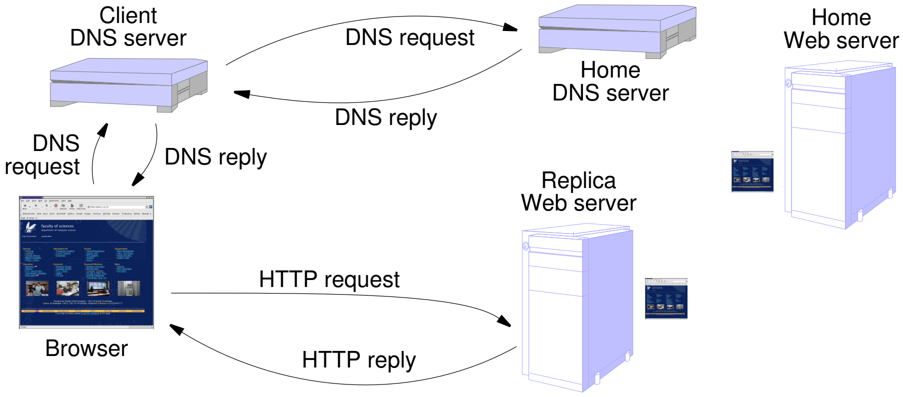


Figure 4: DNS-based redirection

the replica to which the client is redirected. Until the client's 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 can not 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 [19].

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 with these transport-level redirection mechanisms makes the methods more suited for long-lived sessions such as FTP [5, 20] 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

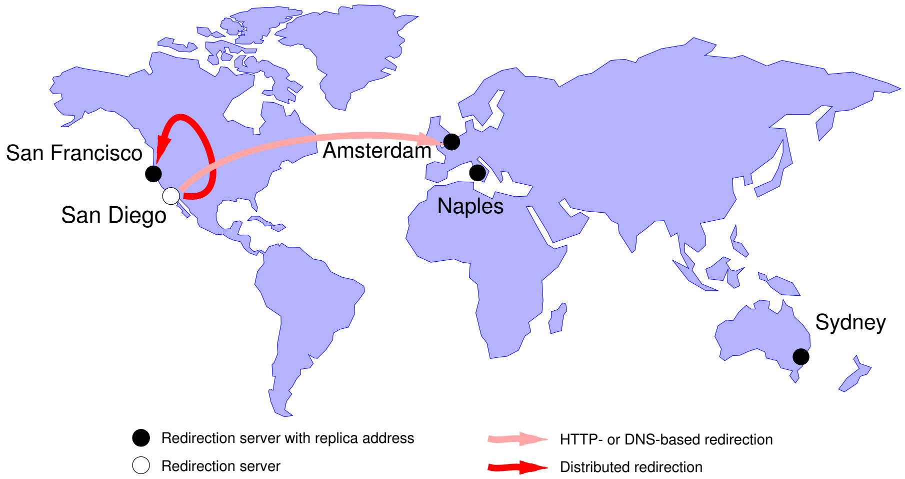


Figure 5: Using distributed redirection to access a close-by replica

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 requesting 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.

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 Figure 5). 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. Figure 6 shows the hierarchy for the page http://www.globule.org/index.html.

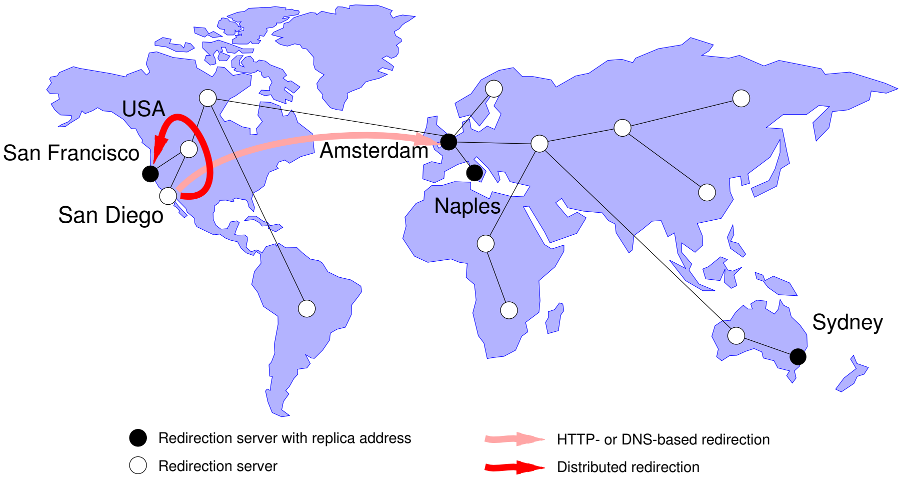


Figure 6: Building a hierarchy of redirection servers

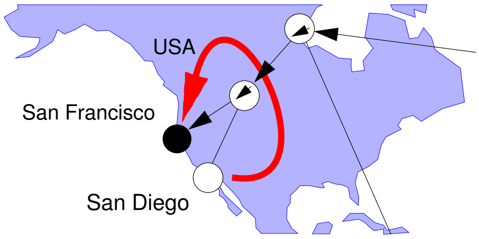


Figure 7: A forwarding pointer at the USA server

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 thus does not have an address for the Web page. However, as shown in Figure 7, 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.

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 us 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

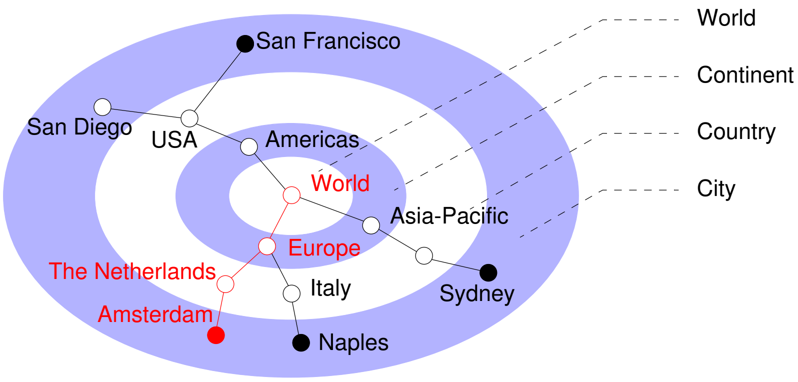


Figure 8: The hierarchy of domains and a hierarchy of servers

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 globe 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 Figure 8. 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 to 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 [28]. 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. Figure 8 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. This hierarchy is organized around the Amsterdam redirection server which is the home location of the Web page. In other words, the root server *World* is located in Amsterdam.

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.

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 US server holds a pointer to a redirection server in the San Francisco subdomain (see Figure 7).

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 [14, 16, 25]).

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 Figure 8. 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, 29]). Requests are always initiated at a lowest-level 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 the alternatives for sending back replica addresses.

Each redirection server acts independently when dealing with its contents or the 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 [19]. 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 also has its importance. First of all, it is best to choose the Web page home location as root server. Second, not only the placement of the root is of importance, consider the following example. Figure 8 shows the hierarchy of servers for the Globule Web page. The root redirection server for the Globule Web page is 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). As it is the case for Web servers, 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 an 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 makes the caches 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 $0 < i \leq 3$, 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 redirection server 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 [3]. 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.

5.1.1 Maintaining transparency

First, a client should not be aware it is dealing with a replica of the Web page it requested. A client should not be able to keep an explicit reference towards a replica of a Web page, for example by 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. Figure 9 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 client browser's DNS request into the address of its local redirection server (see Figure 9, 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

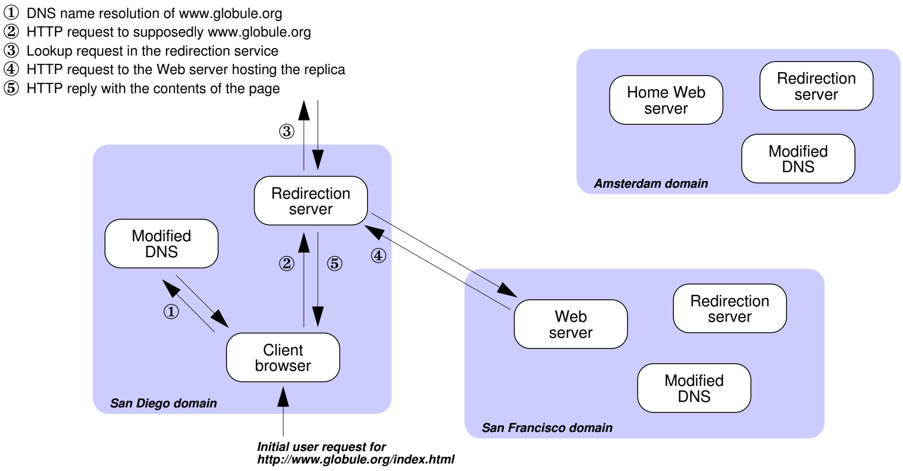

① DNS name resolution of www.globule.org
② HTTP request to supposedly www.globule.org
③ Lookup request in the redirection service
④ HTTP request to the Web server hosting the replica
⑤ HTTP reply with the contents of the page
Home Web server
Redirection server
Modified DNS
Amsterdam domain
Redirection server
Modified DNS
Client browser
San Diego domain
Web server
Redirection server
Modified DNS
San Francisco domain
①
②
③
④
⑤
Initial user request for http://www.globule.org/index.html

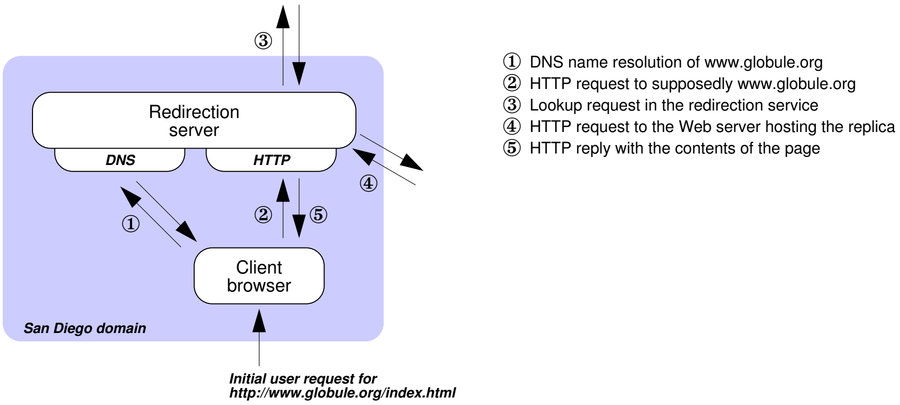
Figure 9: Components used for distributed redirection

realizes 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.

Figure 9 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. Figure 9 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 Figure 10. 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 server, the lookup never gets initiated and therefore does not lead to a lookup miss. Instead, an HTTP request is issued and sent to the supposed root



③
Redirection server
DNS
HTTP
④
②
⑤
①
Client browser
San Diego domain
Initial user request for http://www.globule.org/index.html
① DNS name resolution of www.globule.org
② HTTP request to supposedly www.globule.org
③ Lookup request in the redirection service
④ HTTP request to the Web server hosting the replica
⑤ HTTP reply with the contents of the page

Figure 10: Single redirection server component

(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 Figure 11). 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, 18, 17].

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 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

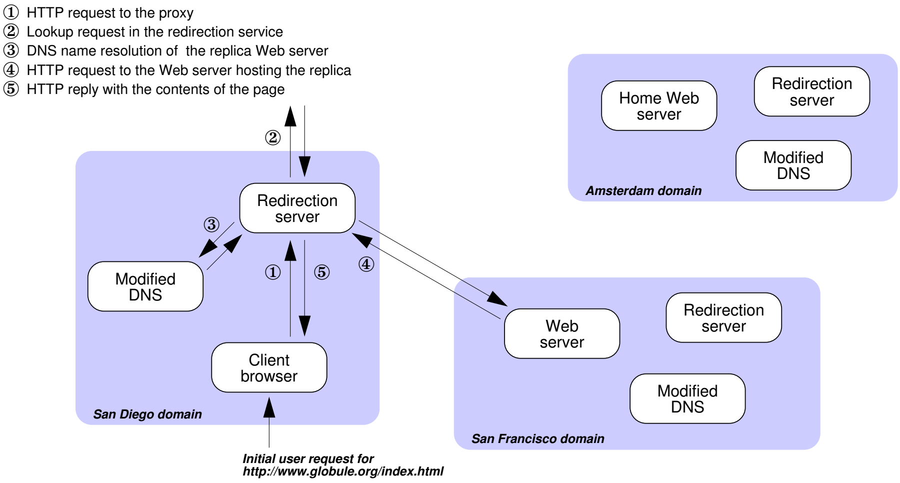

① HTTP request to the proxy
② Lookup request in the redirection service
③ DNS name resolution of the replica Web server
④ HTTP request to the Web server hosting the replica
⑤ HTTP reply with the contents of the page
Home Web server
Redirection server
Modified DNS
Amsterdam domain
②
Redirection server
③
Modified DNS
①
⑤
④
Client browser
San Diego domain
Web server
Redirection server
Modified DNS
San Francisco domain
Initial user request for http://www.globule.org/index.html

Figure 11: Redirection server configured as a proxy

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.

5.3 Integration in today's World-Wide Web

Despite the functional requirements described above, we want our distributed redirection mechanisms to integrate seamlessly in the current World-Wide Web. We consider integration at three different levels: at the server implementation level, at the replica naming level and at the application level.

To facilitate installation and deployment, we decided to implement our redirection server as a module in the Apache server. Apache is a well-known and widely-used Web server. It has the advantage of being easily extendable with new features. Moreover, similar experiments with integrating replication [17] or redirection [24] features in Apache modules have shown the approach to be feasible.

Using an Apache module means that the administrator of a site willing to participate has to install the module by recompiling its Apache Web server. Once the redirection service module is installed, it is set as authoritative DNS server. All the client DNS requests automatically go through the redirection server without any action to be taken by the end users.

Another integration point is the naming scheme. To find addresses of replicas of Web pages, the redirection service has a need for unique, location-independent identifiers for each Web page. An identifier is associated with a number of location-dependent addresses that denote the physical location of the replicas. The de facto naming scheme in use in today's World-Wide Web (i.e., URLs) does not meet these requirements. However, considering the large number of users and applications using this naming scheme, proposing a new naming scheme does not comply with our goal of seamless integration.

We propose to keep URLs as both human-friendly names and addresses of replicas. The redirection service is in charge of translating these names into identifiers and then use the identifiers to look for replica addresses. An identifier is directly derived from a URL. It is composed of a hash of the URL and by the identifier of the home Web server contained in the URL. The home Web server identifier is used to avoid collisions due to the hash function. As a matter of fact, each URL has to be given a *unique* identifier to be able to look for the correct replica addresses in the redirection service. The home location identifier could be replaced by the result of another hash function. An identifier could therefore be computed as the result of two independent hash functions. For a given URL identifier, several replica addresses can be registered in the redirection service. The address of a replica is simply stored under the form of a URL.

Finally, as for integration at the application level, each participating Web server has to register its pages with the redirection service. In other words, a participating Web server has to publish the addresses of its pages in order to make them available through the redirection service. The registration of pages can be achieved automatically by having a Web page indexer running at the participating site from time to time and by letting it register the pages it finds if they are not already. However, such an indexer has also to take into account pages that have disappeared or pages that have moved. Disappearing pages are dealt with by adding leases to addresses of replicas in the redirection service and letting the indexer run periodically. The address of a replica that is not reinserted before its lease expires is deleted from the redirection service, thus facilitating garbage collection and keeping the indexer stateless – there is thus no need to maintain a list of registered pages.

Saying that a page has moved means that the URL of the page has changed while its content remained identical. This happens in most cases after the directory structure has been changed on the home Web site, for example, a directory was renamed or moved. As a consequence of the move and of the URL change, the page identifier changed as well. Providing continuity of service in the redirection service for a given page is important. This means that even if a page has moved (i.e. is known under a new page identifier), the redirection service should still be able to find replicas, even if the users are still referring to the page using the old URL and therefore the old identifier. This cannot be achieved using only naming or unique identifiers. Either it is necessary to use an extra service mapping URLs to identifiers, or a Web page identifier has to be embedded in the page contents. The first approach has the advantage of being fully transparent but adds yet another level of indirection. It uses a name service mapping multiple, potentially broken, URLs to multiple valid identifiers. After that the redirection service maps the identifiers to still existing Web pages. The second approach does not require this extra level of indirection but is very user-dependent. At any point the author of a Web page can remove the identifier embedded in the HTML code.

6 Simulating distributed redirection

To validate the distributed redirection mechanism and estimate the overheads and gains of the method, we built a trace-driven simulation. This section describes the simulation settings and presents results.

6.1 Experimental settings

The goal of the simulation is to show how the distributed redirection performs in a realistic Web environment without having to deploy it in real. One important aspect is to evaluate the gain in user-perceived latency booked with distributed redirection compared to a direct access to a Web page (i.e. no redirection, direct access to 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 the Web pages' replicas.

The simulation is based on HTTP traces from our department Web server `www.cs.vu.nl` at the Vrije Universiteit. It spans more than one year (57 weeks from June 7th 2002 to July 20th 2003). This HTTP trace logs for each request received by our Web server the following data: the time at which the request was performed, the IP address of the requesting client, the HTTP method, the URL and the file name of the page, the HTTP code returned by the server, the size of the document, its last modification date and the time it took to process the request at the server. Prior to the simulation, a considerable amount of data preprocessing is necessary: the HTTP trace is filtered, the hierarchy of redirection servers for our Web server is built and the placement of the Web pages' replicas is determined.

Filtering: Since not all the HTTP requests are suited for our redirection experiment, the HTTP trace gets first filtered. The only requests to be replayed in the simulator are the well-formatted, successful requests, namely the GET requests with return code 200 with well-formatted URLs (no space, accents or other illegal characters). From these requests, we produce a filtered HTTP trace with, per request, the IP address of the client and the file it requested.

Building the hierarchy: We use a four-level domain hierarchy, as shown in Figure 8, formed by the levels Root, Continent, AS and Network. During the trace filtering, we extract the clients IP addresses. Each client is given an Autonomous System (AS) using BGP router information [27]. The IP addresses not found in the BGP routing tables are placed in the fake Autonomous System 0. Each AS is given a continent, based on NetGeo information from Caida [4]. The partitioning into continents is as follows: Africa, Asia, Europa, North America, Oceania, South America, Unknown. The continent "Unknown" is used for all the ASes for which we do not have a geographical mapping and for AS 0.

Knowing the client IP addresses to AS and the AS to continent mappings, we can place each client into the appropriate domain. First, per AS, the client IP addresses are grouped according to their network prefix into so called networks groups. The latency within a network group remains small and enables the distributed redirection mechanism to benefit from locality. The grouping mechanism uses BGP router information to group the clients using a 16- or 8-byte mask (matching IP addresses on $x.x.*.*$ or $x.x.x.*$). The minimal size of a network group is 20 addresses and the maximum size 300. If a given AS has less than 20 IP addresses, it is not divided but forms one unique network group. If a given network group has more than 300 IP addresses, it is further divided (the prefix becomes $x.x.x.*$). If, even with the 8-byte mask ($x.x.x.*$), a network group still has more than 300 addresses, it is not further divided as it makes no sense to have network groups containing only one IP address. The network groups form the fourth level of the hierarchy. The third level of the hierarchy is formed by the ASes. These ASes are aggregated into continents and they form the second level of the hierarchy. Finally, the continents are aggregated and the root forms the first level of the hierarchy.

Whereas the hierarchy of domains is common to all the participating Web servers, the hierarchy of redirection servers depends on the Web server we are considering. It is built as follows. For each network group, one IP address is selected to act as a redirection server for this group. One IP address per AS is selected among the network group redirection servers to act as an AS redirection server. The same happens for the continent level: for each continent, one of the AS redirection servers is selected to act as continent server. Finally, a redirection server among the continent servers is selected for being the root server. Since we are building a hierarchy for the `www.cs.vu.nl` Web server, some of the redirection servers have to be explicitly placed in Amsterdam, home of `www.cs.vu.nl`, as explained in Section 4.2.2. This means that the redirection servers of the `www.cs.vu.nl` AS (AS 1103), the continent server "Europe" and of the Root server are all allocated to the same IP address. This IP address has to be part of one of the Vrije Universiteit network groups (IP address matching 192.31.231.$*$ or 130.37.$*.*$).

At this stage, all the redirection servers are known. The filtered HTTP trace is rewritten: each client IP address is substituted with the IP address of its local redirection server (i.e. network group server). Seen from the redirection service point of view, the local redirection server is the representative for the client. This means that the filtered trace contains only references to network group servers and not to clients anymore. With respect to the simulation, this substitution accelerates the replay: the simulator does not have to map a client IP address to a given network group server.

Placing the replicas: Once the hierarchy of redirection servers is built, we generate the placement of each unique Web page. Each page is first placed at the home server `www.cs.vu.nl`, that is to say at the Vrije Universiteit network-group server that was appointed as Root. For example, in Figure 8, the addresses of all the `www.cs.vu.nl` Web pages would be available at the Amsterdam server. In addition, a replica of the page is placed in each network group where there are enough clients and therefore enough demand. For example, for each network group where the number of requests for a given page is greater than five, a replica is placed. At this point, we can choose to differentiate network groups located at the Vrije Universiteit and network groups located elsewhere: we can choose to place extra replicas at the Vrije Universiteit or not. To do so, we use a list of IP addresses of the network-group servers located at the Vrije Universiteit that we extracted when building the hierarchy. This way we can generate two possible placements of the replicas, one with replicas placed in the Vrije Universiteit network groups and one without.

Measuring the latencies: As an additional step, we measure the latency between each pair of parent and child redirection servers. These latency values are stored along with the hierarchy of redirection servers. We also measure the latency between each network group redirection server and the root redirection server. These latency measurements will be used during the trace replay to estimate the latencies of the requests with or without redirection.

To measure the latency, we used a modified version of King [10]. King takes benefit of the existing DNS infrastructure to evaluate the latency between any arbitrary couple of IP addresses. More precisely, King gives an estimate of the latency between IP addresses α and β by measuring the latency between α's DNS and β's DNS. In order to be able to measure a latency, King needs that at least one DNS server is recursive. The changes we made to King concern mainly two aspects. First, while King always returns the minimal latency value it found, we made it possible to get the maximal value and the average where the minimal and maximal values are removed. For our experiment, we used averages of 10 latency measures (i.e. average on 8 values, maximum and minimum excluded). Second, when taking two arbitrary IP addresses, it can happen that these two addresses are in the same local-area network and are likely to use the same DNS server. In such a case, King returns some incoherent latency values and sometimes even negative latencies. To be able to detect errors and to speed-up the latency measurement process, we check for potential DNS matches before trying to evaluate the latency.

Due to name-resolution errors, unreachable hosts, lack of recursive DNS servers, or matching DNS servers, errors or missing latencies always remain, leading to gaps in our latency measurements. To reduce the number of gaps, when building the hierarchy, we selected as redirection server in priority pingable IP addresses. It can still happen that some of the selected redirection servers are unreachable but we avoid as much as possible selecting an unreachable server in the AS, continent or root level of the hierarchy.

The last step consists in filling the remaining gaps in the latency measurements. This is done by calculating statistics on the latencies per level and by using average values per network group, AS. At the continent level we did not have to fill gaps thanks to the careful selection of redirection servers. In addition, some of the gaps are easily corrected, for example, the latency of two redirection servers having a DNS server in common is set to a fixed value. These servers are assumed to be in the same network. In some cases, the same IP address is acting at different levels of the hierarchy. In that case, the parent-child latency is set to zero.

6.1.1 Figures about the simulation

- Vrije Universiteit Web server trace
  - 57 weeks (up to July 2003)
  - 1,633,404 unique client IP addresses
  - 2,615,952 unique documents
  - 61,670,630 “correct” requests (GET, 200, well-formatted URLs, etc.)
- Building the simulation
  - 4-level hierarchy of redirection servers: root - continent - AS - network group
  - 180,686 redirection servers
    - 1 root server
    - 7 continent servers (6 continents + 1 “unknown”)
    - 10261 AS servers
    - 170417 leaf servers (network groups)
  - 1,633,404 ping checks (independent of the hierarchy)
  - 351,102 king latency measurements (hierarchy-dependent: latency child-parent & leaf-root) x 10 (average)
  - from 3,718,179 up to 7,688,606 replicas (& number of insert requests) (dependent on the replica placement policy)

– 61,670,630 lookup requests

6.2 Running the simulation

The simulator is built to reproduce the behavior of the collection of redirection servers, in our case for the Vrije Universiteit Web server `www.cs.vu.nl`. 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.

The simulator is fed with the hierarchy of redirection servers, the placement of the replicas of the Web pages and the filtered HTTP trace. The first step of the simulation consists in building the hierarchy and placing the replicas of the Web pages both at the home server and in the other networks groups if need be. Placing the addresses of the Web pages is done by invoking `insert` requests, as described in Section 4.2. For each insert, we log the latency and the number of hops in the hierarchy the request had to travel.

Once the hierarchy is constructed and replicas are placed the simulator replays the clients requests found in the filtered HTTP trace. Each request from the trace leads to a lookup request initiated at the specified network-group server. For each lookup, the simulator logs the latency of the request, number of hops in the hierarchy it traveled and at which server the address of the requested document was found. If during the lookup several replicas (i.e. forwarding pointers) are found, the request follows the branch with the smallest latency. In addition, each redirection server records how many lookup requests it had to handle during the trace replay, for both client lookup requests or forwarded lookup requests. This gives us an estimate of the load of each redirection server "inside" the hierarchy as well as that of the network group servers.

6.3 Result analysis

We ran the simulation a couple of times with the same hierarchy but with different replica placement or document set. The results of our simulations are as follows.

Figure 12 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"*.

Figure 12 shows that more that 34% of the documents addresses are served locally (latency equals zero) and that close to 43% of the addresses are found with a latency less than 4000 μ seconds. 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 the access to the closest distributed redirection server holding a replica address for the requested document. HTTP-based redirection (or here HTTP direct access such as in a classical Web environment) provides a reference point for our mechanism as it combines both the worst and the best cases. HTTP-based redirection is the worst case as each request travels invariably to the home Web server for being redirected. HTTP direct access is the best case as the request need no more forwarding once it has reached the home Web server and the document can be served directly.

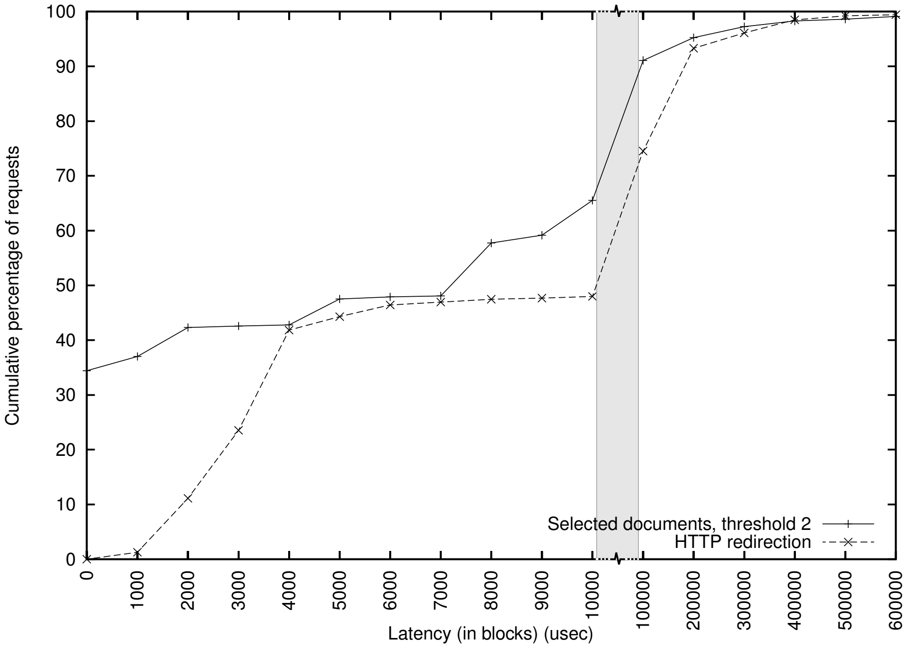


Figure 12: Proportion of requests serviced within a given latency for the *"selected documents, threshold 2"* experiment

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"*. Figure 13 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.

Figure 14 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"*.

Figure 14 shows that even with a small number of replicated documents, the distributed redirection performs better than HTTP redirection up to 3000 μ seconds. Close to 16% of the documents addresses are served locally (latency equals zero) and at 3000 μ seconds, 23% of the requests have been served against 24% for HTTP redirection.

Figure 15 shows the number of hops for the different simulations. We count hops as follow: one hop 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.

Figures 16 and 17 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 μ seconds or 3000 μ

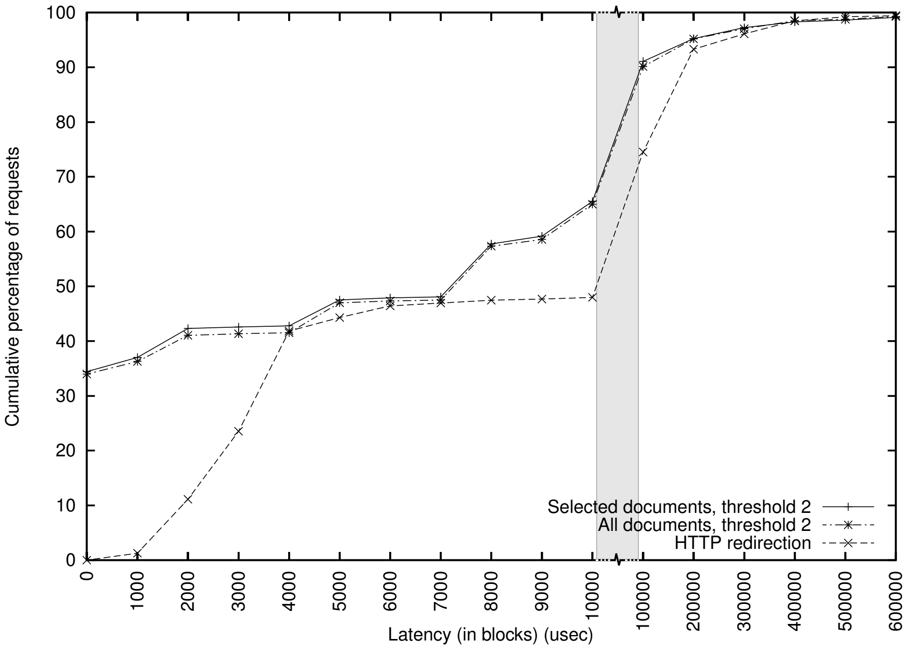


Figure 13: Proportion of requests serviced within a given latency: comparison of the *"selected documents, threshold 2"* and *"all documents, threshold 2"* experiments

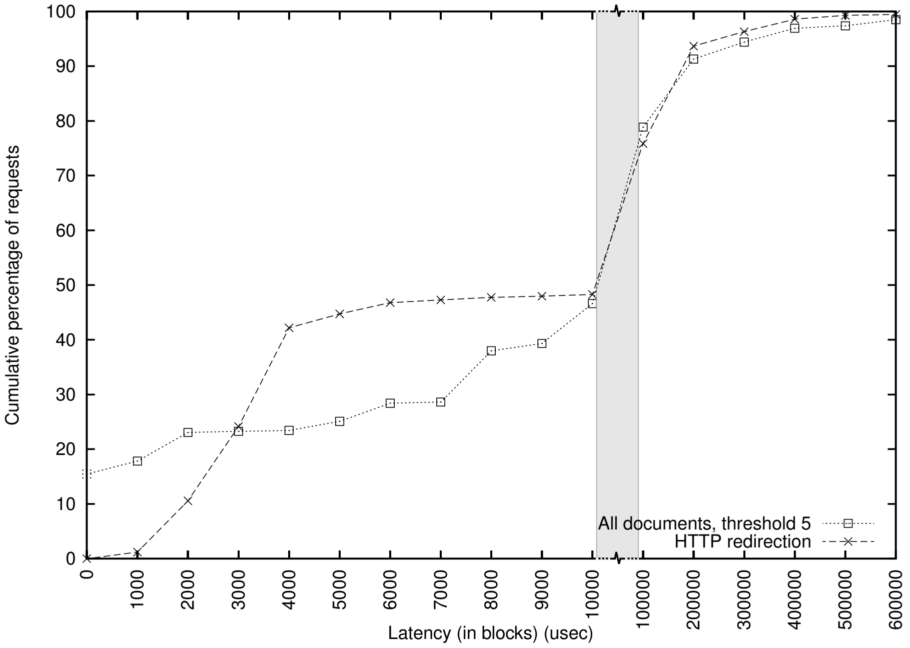


Figure 14: Proportion of requests serviced within a given latency for the *"all documents, threshold 5"* experiment

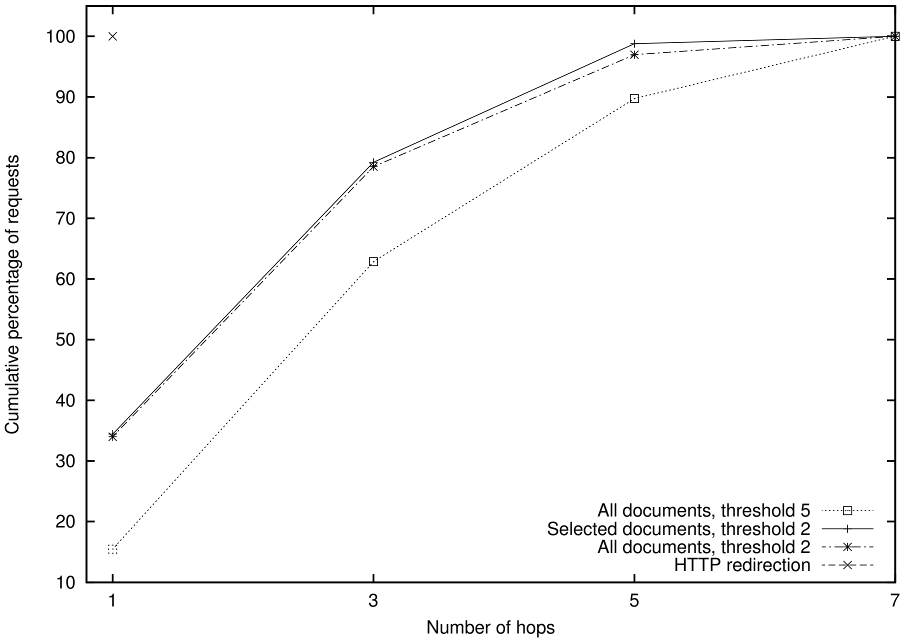


Figure 15: Proportion of requests serviced within a given number of hops

seconds, respectively referred as optimistic or pessimistic. For each replayed request, we multiply the number of hops by the server time.

Figure 16 shows the output in the case of an optimistic server time. The curves are slightly shifted compared to Figure 12. Close to 34% of the requests are now serviced in 1000 μ seconds. The distributed redirection still performs better than HTTP redirection up to 5000 μ seconds (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.

Figure 17 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 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 the hierarchy. The root server gets to see only a small proportion of the client requests, namely 0.12%. Figure 18 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

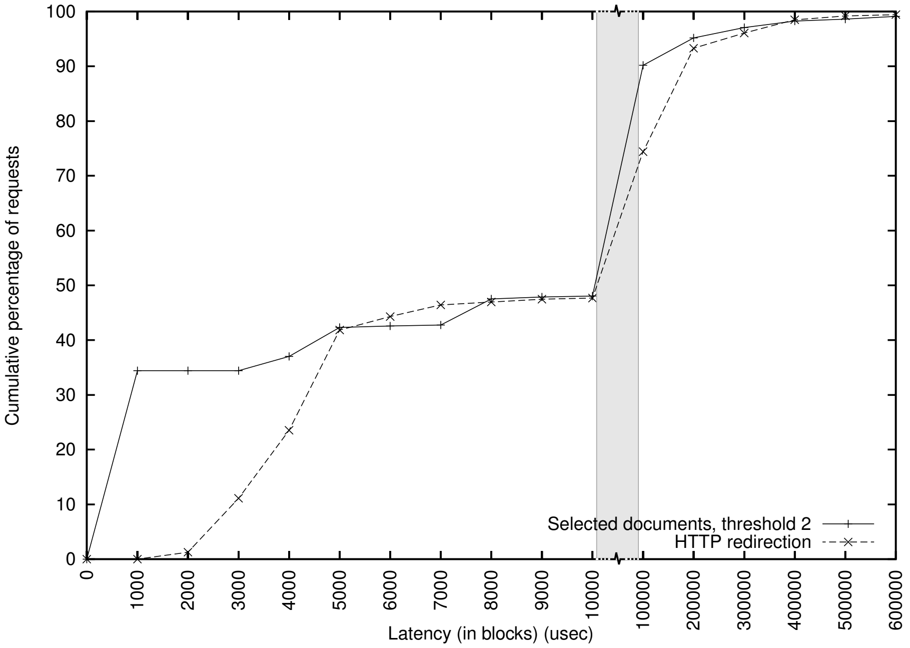


Figure 16: Proportion of requests serviced within a given latency with optimistic server time for the *"selected documents, threshold 2"* experiment

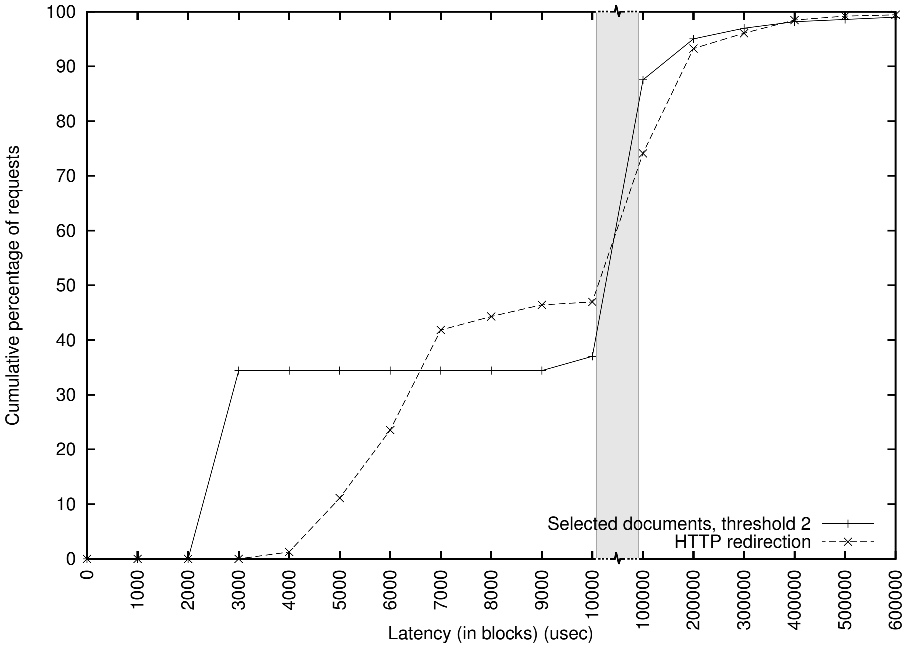


Figure 17: Proportion of requests serviced within a given latency with pessimistic server time for the *"selected documents, threshold 2"* experiment

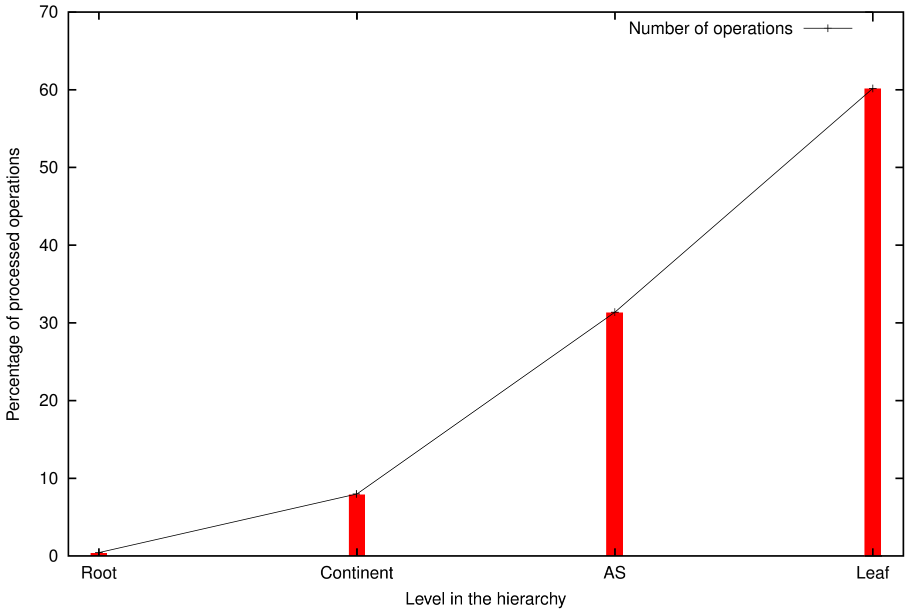


Figure 18: Distribution of operations among the different levels for the *"selected documents, threshold 2"* experiment

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

This section discusses the merits and disadvantages of distributed redirection. It gives a detailed comparison of the distributed redirection with other Web or non-Web redirection mechanisms as well as other mechanisms such as Web caches.

7.1 Web redirection mechanisms

Section 2 gave an overview of the the redirection mechanisms currently in use in today's World-Wide Web, namely application-level, transport-level and DNS-based redirection. Table 1 shows a comparison of the distributed redirection mechanism with HTTP-based redirection, DNS-based redirection and TCP handoff. This table refines an earlier study on DNS-based redirection [24] and incorporates remarks from [5]. In this table, the number of stars expresses how well or badly a redirection mechanism supports a given feature. A rate of five stars denotes the best mechanism for the considered criterion. The other mechanisms are rated relative to the best. A single star rates a redirection mechanism that does not support the considered feature or does it in a very unsatisfactory way.

The first property denotes how transparent the redirection is to clients. HTTP-based redirection performs the worst here: references to replicas are explicitly displayed to clients. TCP handoff performs the best. Since redirection is achieved at the network level, clients are not aware they are redirected. DNS-based redirection is less transparent than TCP handoff since it operates only at the domain-name level. The redirection is short-cut if the clients use IP addresses to access Web pages rather than names. Distributed redirection is as transparent as DNS-based redirection but has the advantage that it it cannot be short-cut if used as a proxy.

No.	Property	HTTP-based redirection	TCP handoff	DNS-based redirection	Distributed redirection
1	Redirection transparency	*	*****	***	****
2	Level of integration	*****	*****	*****	*****
3	Scalability	****	**	*****	*****
4	Deployment	*****	*	****	****
5	Redirection granularity	*****	***	***	*****
6	Client loc. identification	*****	*****	***	***
7	Multiple response	*	*	*****	****
8	Server control	*****	****	****	*
9	Simplicity (implementation)	*****	**	****	****
10	Replica selection (static)	*****	***	***	****
11	Replica selection (dynamic)	*****	***	***	*
12	Installation transparency	*****	*****	*****	**
13	Load on the primary	*	**	**	*****
14	Responsiveness	*	**	***	*****
15	Support for massive replication	**	*	**	*****

Table 1: A comparison of redirection mechanisms

The level of integration of the four redirection protocols in the existing World-Wide Web infrastructure is equally good. All four are based on well-known protocols and require no protocol modification.

The third criterion evaluates how scalable the different redirection mechanisms are. Both DNS-based and distributed redirection perform well with respect to scalability. The DNS name resolution mechanisms are widely used in today's Internet and have proved to be very efficient and scalable. An important aspect is to be found in DNS caching features. Distributed redirection is based on scalable mechanisms similar to those of DNS. It relies less heavily on caching, however, the load of redirecting is fully distributed over the whole collection of redirection servers. HTTP-based redirection implies that each HTTP request goes to the home server before being redirected. In that respect, it is less scalable than DNS-based and distributed redirection. Finally, TCP handoff shows a behavior absolutely not suited for wide-area networks. The traffic generated by the redirection mechanisms breaks down the scalability to only local networks.

The deployment property gives a hint about the ease of installation and maintenance of each redirection method. HTTP-based redirection scores the best here since it relies entirely on the HTTP protocol. No extra software is required. This is not the case with DNS-based and distributed redirection. Both mechanisms can be implemented by means of an Apache module and require Apache to be recompiled. With TCP handoff, modifications of the operating system and network gateways are necessary. It thus rates the worst.

The fifth property denotes the level of granularity at which each of the four mechanisms achieve redirection. HTTP-based redirection shows to have a fine granularity. The selection of the replica can be made on a per-page basis. This enables flexibility when redirecting. It scores the best, together with distributed redirection which also supports fine-grained redirection. TCP handoff and DNS-based redirection work, respectively, at the TCP end-point (IP address and port number) or DNS-domain levels which provides a coarser-grained redirection mechanism.

The sixth criterion expresses how accurately we can establish the location of the client browser. This piece of information is important when the replica selection is based on geographical distance. To select a replica as close to the client as possible, the client location has to be evaluated. HTTP-based redirection and TCP handoff perform the best here. In both cases, the address of the client host is known when the redirection takes place. DNS-based redirection provides a less accurate evaluation: only the location of the client's authoritative DNS server is known, which is usually rather inaccurate [12]. Distributed redirection is as inaccurate. A client is represented by its redirection server, which is likely to be located in its own local network (the redirection server should be co-located with the local Web server).

The multiple response property specifies whether it is possible to get the addresses of several replicas as part of a reply. Both HTTP-based redirection and TCP handoff do not support this feature. With DNS-based redirection, it is possible to get the addresses of several hosts to redirect to. It gets the highest mark. Distributed redirection also allows a client to get a multiple response but is slightly less efficient than DNS-based redirection. The hierarchy of redirection servers has to be recursively browsed through possibly up to the root to find the requested number of addresses.

The server control criterion shows how much control the redirection method leaves to the home server of the Web page. HTTP-based redirection scores the best since each URL can be independently redirected by the home server itself. TCP handoff and DNS-based redirection perform somewhat worse as they allow only per-address or per-domain-name redirection. However, the redirection is still decided by the server side (gateway or DNS server). Distributed redirection completely removes the control of the redirection from the home server. This can be seen as a disadvantage if the redirection has to be achieved on criteria such as the load of the different replicas, although such decisions can easily be taken by redirection servers.

The ninth property denotes the internal complexity of the implementation. It is kept to a minimum with HTTP-based redirection since redirection is directly supported by the HTTP protocol. TCP handoff shows here to be a rather complex solution involving operating system and network gateway modifications. Both DNS-based and distributed redirection approaches exhibit similar properties for the complexity of their implementation. They score a bit worse than HTTP-based redirection.

The tenth property expresses how easy it is to select a replica based on static criteria such as location or specific replica properties. HTTP-based redirection makes it easy to select a specific replica based on some static properties of this replica. It scores the best. TCP handoff and DNS-based redirection are slightly worse as they allow only coarse-grain redirection. Only the initial request to a given domain or address is actually selected. For the other Web pages with the same address or domain name, such as the inline images of a Web page, the client is redirected to the same replica. The distributed redirection approach enables a fine-grain redirection but changing the selection criterion requires a hierarchy reconfiguration (the hierarchy of redirection servers is normally built on a locality criterion).

Replica selection can also be based on dynamic criteria such as the load of each replica. This is represented by the eleventh criterion. HTTP-based redirection also makes it easy to select a specific replica based on dynamic properties. For example, the home location knows all the available replicas as well as their estimated load (we can not take into account the load induced by the clients accessing directly the replicas). It is therefore easy to use this information to redirect new incoming requests. TCP handoff and DNS-based redirection are slightly worse as in the case of static criteria. Distributed redirection makes it more difficult to deal with dynamic criteria. The dynamic information, for example, the load, has to be distributed and the hierarchy of servers may have to be adapted (or, if not, we are forced to browse up to the root for each lookup request). However, any redirection server can take decisions based on local and as such incomplete information. For example, a redirection server having a forwarding pointer to a replica address can count the number of clients it redirected to that specific address. In other words, the load can be taken into account but cannot reflect the global load on a Web page or even on a given replica.

The twelfth property expresses how transparent the installation of the redirection mechanism is. It quantifies the actions that have to be taken at the client side (browser) to make use of the redirection mechanisms. Nothing has to be installed at the client side for using HTTP redirection, TCP handoff or DNS-based redirection. In these three cases, actions are taken at the server side (gateway or DNS server), if not at the server itself. With distributed redirection, the client needs to contact the redirection server, whether it is acting as the authoritative DNS server or as the client's proxy. In both cases, the system administrator of the client site has to install the redirection server (Apache module) and configure it as the authoritative DNS server. Due to this, distributed redirection scores the worst here.

The thirteenth criterion evaluates the load set on the home Web server or on the Web server side. The HTTP-based redirection performs the worst as any client request as to be dealt with by the home Web server. TCP handoff and DNS-based redirection perform slightly better in the sense that requests reach the home server side (DNS or gateway) and not the server itself. However, there is still traffic that potentially travels large network distances. The distributed redirection performs the best here since the redirection is fully distributed. The home Web server or server side do not have to be contacted at all since the redirection occurs locally to the client.

The fourteenth criterion examines responsiveness in terms of expected latency. HTTP-based redirection performs the worst. For each request, the home Web server has to be contacted, process the request, choose the replica and send back the redirection message. TCP handoff is slightly better since only the server-side gateway has to be contacted. DNS-based redirection is also slightly better as it can benefit from DNS caches. Distributed redirection shows here to be the best. The good responsiveness comes from the fact that mostly local communication is involved. A client contacts its local redirection server which is in charge of finding the closest possible replica. A lookup goes no further than necessary in the hierarchy. The latency is kept to a minimum.

The last criterion evaluates how easy it is for each method to support massive replication. HTTP-based redirection relies on the home Web server for achieving the redirection. First, the home server has to know about all the replicas. Second, it has to handle all the incoming client requests. When one of its pages becomes very popular, the home server can therefore easily become a bottleneck. HTTP-based redirection thus does not score well for this property. However, TCP handoff scores worse. As for HTTP-based redirection, all the requests have to travel to the home server side. Moreover, TCP handoff scales only to local-area networks, it cannot easily support massive replication of Web documents. DNS-based redirection benefits from its caching capabilities. However, as for HTTP-based redirection or TCP handoff, the traffic and the load generated at the home Web server side makes it inappropriate for supporting a very large number of replicas. Distributed redirection shows a good behavior as the load is fully distributed over the hierarchy of servers. It gets the highest mark.

As a summary, the most important problems of the different methods are as follows. HTTP-based redirection shows good behavior in most of the criteria but it severely lacks transparency of redirection and responsiveness. TCP handoff is complex and not usable in a wide-area environment. DNS-based redirection enables only coarse-grained redirection and gives only an approximation of the client location. Distributed redirection can be slightly more accurate in client location identification but it does not show such a dramatic change compared to DNS-based redirection. Moreover, the control let to the home Web server and the possibility of selecting replicas on dynamic criteria are more limited than with HTTP- or DNS-based redirection. Using an entirely decentralized mechanism makes it more difficult – but not impossible – and costly to take into account changing information over the different replicas since those have to be distributed. Nevertheless, dynamic information can be gathered independently at each redirection server, such as the load expressed in number of clients having requested a given page or address. This local information can be used to take decisions on where to redirect a client. If need be, this local information can be transfered to other redirection servers, potentially up to the root, using piggy-backing techniques.

The bad score of distributed redirection at properties eight and eleven can be dealt with. In any of the three other Web redirection mechanisms, controlling to which replica the client is redirected is considered as a task of the home Web server. The home Web server has therefore to gather information about the load of its different replicas and redirect the clients accordingly. We consider that this should be dealt with by the replication policy associated to each replicated Web document. A Web page, its inlined images and other embedded or related documents can be seen as a whole, let us call it a Web object, as in [11]. Each Web object is given a replication object in charge of implementing its replication policy. The replication object has to gather load information, for example, and creates new replicas as the need arise. This separation of concerns lets the home Web server concentrate on its main task: serve Web pages.

7.2 Other Web mechanisms

Web caches provide an alternative for redirection. Each HTTP request that a client generates may first travel to the local cache in order to check whether the Web page has already been downloaded. Hierarchical caches are based on the same principle [7]. The caches are organized as a tree and requests are forwarded to higher-level caches. If a Web page was not found in the local cache, the request is forwarded to the upper-level cache. Hierarchical caches are commonly used. However, they do not guarantee that a request returns the closest replica. They do not guarantee either that a replica is inevitably found if there is one somewhere in the cache hierarchy.

Cooperative caches aim at short-cutting the forwarding of requests and at taking benefit of pages cached at low-level caches [26]. Cached data is partitioned (and sometimes replicated) among the different sites participating in the cooperative cache. By maintaining a directory of cache entries at each site, a cached Web page can be found in at most two hops: one hop to the local cache and to fetch the actual location of the cached page, one hop to the cache where the page is currently stored. However, two problems arise with cooperative caches [30]. First, the maintenance of the directory of cache entries is often costly. Second, the cache hit percentage appears to be low in the current Web, making it not worth it to maintain the directory.

Cooperative caches show similarities with distributed redirection: information is kept on where a given Web page can be found, often rather than the page itself. However, a lookup in the redirection service guarantees us to find a replica wherever it is stored if it has been registered in the service. Distributed redirection can hopefully provide benefits of cooperative caches without the cost of it.

7.3 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 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 the network locality is preserved: the redirection decision has to take place as locally to the client as possible and the selected replica of the requested Web page has to remain 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 counter 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 [17]. 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 bookstores, computers hardware shops or music stores do not deliver static Web pages but generate them based on history of requests, clients' profile 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 relies in the trade-off it has to make 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. The CDN 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, for example, a request for the local weather is time and location dependent. An alternative to fragment caching is to use on-demand application replication as proposed in [22]. On-demand application replication replicates (chunks of) data only where frequently accesses. This reduces the data-consistency management overhead while improving the user-perceived latency. The mechanism cares for a 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 [22] allowing the system to perform well for a wide range of application workloads and access patterns. The redirection technique is orthogonal to the redirection mechanism itself. On-demand application replication as proposed in [22] uses DNS-based redirection mechanism can but 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.

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. http://www.cs.vu.nl/globe/techreps.html.

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

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

[4] Caida. Netgeo – the internet geographic database. http://www.caida.org/tools/utilities/netgeo/.

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

[6] J. Challenger, P. Dantzig, A. Iyengar, and K. Witting. A fragment-based approach for efficiently creating dynamic web content. *To appear in the ACM Transactions on Internet Technology*, 2004.

[7] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. A Hierarchical Internet Object Cache. In *Annual Technical Conference*, pages 153–163, San Diego, CA, Jan. 1996. USENIX.

[8] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. *IEEE Internet Computing*, 6(5):50–58, September-October 2002.

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

[10] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating latency between arbitrary internet end hosts. In *SIGCOMM IMW 2002*, Marseille, France, Nov. 2002.

[11] I. Kuz. *An Approach to a Scalable Wide-Area Web Service*. PhD thesis, Vrije Universiteit, 2003.

[12] Z. M. Mao, C. D. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang. A Precise and Efficient Evaluation of the Proximity between Web Clients and their Local DNS Servers. In *Annual Technical Conference*. USENIX, June 2002.

[13] P. Mockapetris. Domain names - concepts and facilities. RFC 1034, Nov. 1987.

[14] T. E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *21st INFOCOM Conference*, pages 170–179, New York, NJ, USA, June 2002. GNP.

[15] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Request Distribution in Cluster-Based Network Servers. In *8th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, San Jose, CA, Oct. 1998. ACM.

[16] M. Pias, J. Crowcroft, S. Wilbur, S. Bhatti, and T. Harris. Lighthouses for scalable distributed location. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, USA, Feb. 2003.

[17] G. Pierre and M. van Steen. Globule: a platform for self-replicating Web documents. In *6th Int. Conf. on Protocols for Multimedia Systems*, pages 1–11, Enschede, The Netherlands, Oct. 2001.

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

[19] G. Pierre, M. van Steen, and A. S. Tanenbaum. Dynamically selecting optimal distribution strategies for Web documents. *IEEE Transactions on Computers*, 51(6):637–651, June 2002. http://www.cs.vu.nl/~gpierre/publi/DSODSWD_toc2002.php3.

[20] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959, Oct. 1985.

[21] M. Rabinovich and O. Spastscheck. *Web Caching and Replication*. Addison-Wesley, Reading, MA., 2002.

[22] S. Sivasubramanian, G. Pierre, and M. van Steen. Replicating web applications on-demand. In *IEEE International Conference on Services Computing*, Shanghai, China, Sept. 2004.

[23] W. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[24] M. Szymaniak. DNS-based client redirector for the Apache HTTP server. Master's thesis, Warsaw University and Vrije Universiteit, June 2002.

[25] M. Szymaniak, G. Pierre, and M. van Steen. Scalable cooperative latency estimation. In *Tenth International Conference on Parallel and Distributed Systems (ICPADS)*, Newport Beach, CA, USA, July 2004.

[26] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Design Considerations for Distributed Caching on the Internet. In *19th International Conference on Distributed Computing Systems*, pages 273–284, Austin, TX, June 1999. IEEE.

[27] University of Oregon. The route views project. http://www.routeviews.org/.

[28] M. van Steen and G. Ballintijn. Achieving scalability in hierarchical location services. In *26th International Computer Software and Applications Conference (CompSac)*, pages 899–905, Oxford UK, Aug. 2002.

[29] M. van Steen, F. Hauck, G. Ballintijn, and A. Tanenbaum. Algorithmic Design of the Globe Wide-Area Location Service. *The Computer Journal*, 41(5):297–310, 1998.

[30] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. R. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. In *Seventeenth Symposium on Operating Systems Principles*, pages 16–31, Kiawah Island Resort, SC, USA, Dec. 1999.