Mirroring Resources in the World Wide Web

Francisco Vilar Brasileiro *Tércio E. de Albuquerque Fonsêca* *Raquel Meneses da Costa*

fubica@dsc.ufpb.br *tercio@dsc.ufpb.br* *raquel@dsc.ufpb.br*

Universidade Federal da Paraíba
Centro de Ciências e Tecnologia
Departamento de Sistemas e Computação
Laboratório de Sistemas Distribuídos
Av. Aprígio Veloso 882
58.109-970, Campina Grande, Paraíba, Brazil
http://www.dsc.ufpb.br/~lsd

Abstract

One of the main problems faced by the users of the World Wide Web is that of a broken link in a hypertext, normally caused by the unavailability of a particular resource. The introduction of redundancy is the key mechanism to solve this problem. Unfortunately, the current infrastructure of the Web is based on a resource identification scheme that maps a resource identifier on a unique physical location; this scheme does not favour the implementation of a support for the replication of Web resources. In this paper we discuss how URNs (Uniform Resource Names), a resource identification scheme proposed by a special work group of the Internet Engineering Task Force (IETF), can be employed to implement a support for the mirroring of Web resources. Our proposal is based on the implementation of a proxy and a gateway that together allow conventional browsers and servers to access and manage mirrored resources transparently, allowing the provision of highly available Web resources.

Keywords: fault tolerance; high-availability; URN; replication.

1. Introduction

In the last few years the Internet has presented a fantastic growth; the number of machines connected to the net has increased from little less than 2 million in mid 1993 to almost 30 million in the beginning of 1998 (source: Network Wizards,

http://www.nw.com/). Much of this growth is due to the development of the World Wide Web (or simply the Web) [3].

The technologies developed to support the Web offer the necessary infrastructure that allows users to explore an enormous amount of information kept in machines spread around the world, using tools with sophisticated and yet easy to use graphical interfaces, and opening an enormous amount and variety of opportunities for the use of global networks of computers.

On the other hand, the Web relatively uncontrolled nature that has propitiated its fast growth is also responsible for some problems faced by its users. In particular, one of the most visible problems is that of a broken link in a hipertext [6]. A broken link occurs when an attempt to access a resource fails due to the inexistence or unavailability of the resource in the indicated address. Broken links leave users unsatisfied, causing bad impression and loss of opportunities for the information suppliers.

A frequent source of broken links is that caused by the unavailability of resources due to failures in the computational infrastructure that supports the Web. A server machine crash, the partition of the communication network that links a user (the client) to a server, or even an overload in the system, can prevent the access to a given resource. In order to tolerate faults, it is necessary to introduce redundancy into the system [8]. In the case of the Web this means the replication of servers and resources, as well as the introduction of alternative routes linking clients to the servers that maintain the resources.

When the unavailability of resources is caused by a failure or a temporary overload in the server, the replication of the server can solve the problem through the implementation of load-balancing schemes that handles server failures and load sharing. There are some successful experiences in this direction [10, 20].

On the other hand, the availability of a route between a particular pair of client and server is a function, amongst other factors, of the actual number of alternative routes linking the client to the server and the bandwidth and reliability of these routes. In turn, the existence of alternative routes to the resources, with adequate bandwidth and reliability, varies enormously from region to region, and often the inclusion of new alternative routes linking a particular resource to its clients is not feasible. In Brazil, for instance, the main national Internet backbones have little redundancy and are overloaded most of the time. Further, in many parts of the country the available

technology cannot be used to revert the situation, within a short time, at a competitive cost. This scenario is not different in many other countries.

A possibility for increasing the number of alternative routes between a client and a particular resource, without the need to create new routes, is to replicate resources in other localities where there already exist physical paths to the client (this technique is commonly used for balancing the load on ftp servers and is known as information mirroring).

Unfortunately, the more widely used Web resource identification scheme is the URL – Uniform Resource Locator [13], which maps each resource identifier to a single instance of the resource located in a well defined server, whose address is part of the URL; consequently, most of the software that implements the current infrastructure of the Web does not support the mirroring of resources in a transparent way.

In this paper we present components that can be inserted in the current infrastructure of the Web to allow the access and the maintenance of mirrored resources transparently. Our proposal uses a proxy [14] that implements a resource identification scheme based on URNs (Uniform Resource Names) [18], making possible for clients to access mirrored resources using conventional browsers.

The utilisation pattern that has dominated the initial development phase of the Web, based essentially on the diffusion of read-only information, has not contributed to the focusing of much attention on the issue of maintaining consistency when updating Web resources. However, new utilisation patterns based on much more complex interactions between clients and servers, and the introduction of redundancy into the system, brings with them much more concern regarding this problem. In our project we have also designed and implemented a gateway [14] that is responsible for implementing the consistent management of mirrored resources, allowing the maintenance of both optimistic and pessimistic consistency semantics [5], depending on the characteristics of the resources to be mirrored.

The remaining of the paper is structured in the following way. In section 2 we study the Web model presenting its main components and their inter-relations; we also show the modifications that must be made in the current infrastructure of the Web to allow the mirroring of resources. In section 3 we present our design for the implementation of the components that will support the access and the maintenance of mirrored Web resources. Finally, section 4 concludes the paper with our final remarks.

2. System Model

2.1. Components and Functioning

The Web is formed by instances of a restricted set of well-defined components that communicate amongst themselves via the HTTP protocol (HyperText Transfer Protocol) [14, 15]. The two main components of the Web are the user agent, i.e. the client or browser, and the Web server, or simply the server. In addition, there are intermediate components that can perform actions in behalf of both the client and the server.

Figure 1, shows a communication chain that contains a client, a server, and the two more common intermediate components in this chain of communication, namely proxies and gateways. A proxy is a forwarding agent for the requests sent by the client to the server. From the clients perspective the proxy is seen as a server; at the same time it acts as a client for the destination server, being able to rewrite part of the request message before retransmitting it to the server. The gateway, on the other hand, works as a receiver agent for client requests, acting as a superior layer implemented above other servers and being able to perform protocol translation.

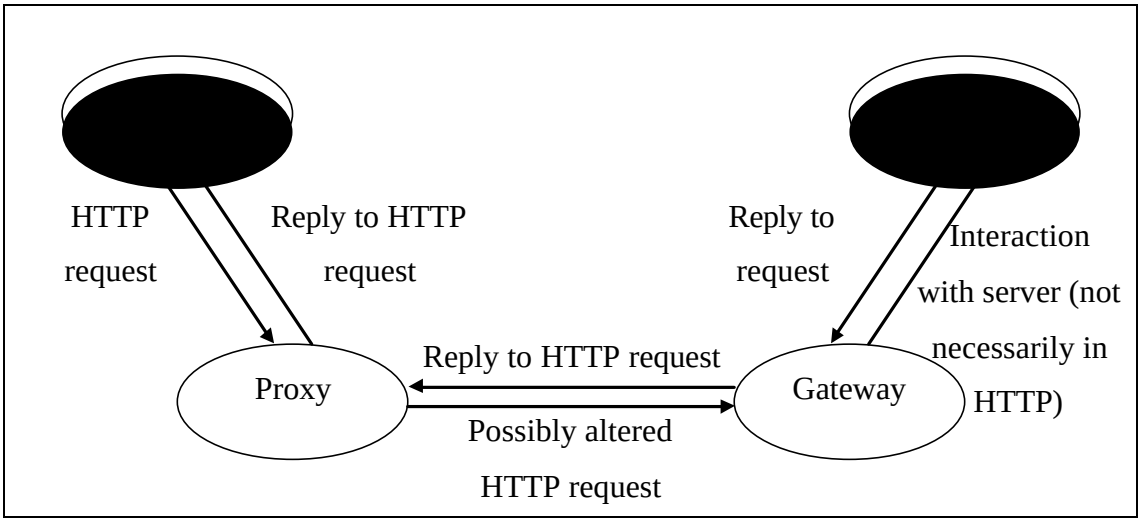

Figure 1: chain of communication with intermediate components

Any one of the components presented in the chain of communication shown in Figure 1 can use an internal memory cache to speed up the processing of requests. If the reply for a particular request is present in the cache of some intermediate component, the chain of communication illustrated in Figure 1 is shortened, allowing a faster processing of the request.

Version 1.1. is the current version of the HTTP protocol. It is proposed in RFC (Request For Comments) 2068, which defines the following set of methods:

- OPTIONS: This method requests information on the available options of communication in the chain of communication identified by the URI[1] contained in the request. This method allows a client (or a proxy) to get information on the options and requirements associated with a particular resource, before carrying out any action on the resource.
- GET: The GET method is used to retrieve a resource from a server. The URI of the resource is supplied as an argument for the method. When the execution of the method is performed successfully by the server, the result is a reply message that contains an entity representing the requested resource. If the URI identifies a process, the reply for the method is the information produced by the process, rather than the process code. The way the method is processed can vary depending on the parameters that are passed in the request message.
- HEAD: The HEAD method is similar to the GET method, but the server does not send all the information associated with the resource. A resource is composed of some meta-information and the information; the HEAD method returns only the part associated with the resource meta-information. This method can be used to test the validity of a hypertext link or to get information on accessibility and date of the last update of a resource, without having to retrieve the resource.
- POST: The POST method is used to request that the destination server accept the entity included in the request as a new subordinate of the resource identified by the URI passed as an argument. This method was designed to allow the execution of tasks such as: posting messages to mailing lists, bulletin boards, etc; add information to a database; provide information submitted to a process that treats data associated to a form; etc. The processing effectively carried out on the entity depends on the resource identified by the URI contained in the request.

[1] URI - Universal Resource Identification, is the generic term used by the HTTP protocol to refer to a resource identification scheme [12].

- PUT: This method is used to request the server to store the entity contained in the request; the resource updated is identified by the URI passed as an argument in the request. If the resource identified by the URI already exists, the received entity must be considered as being a modified version of the existing resource and must be used to substitute it. The basic difference between the POST method and the PUT method lies in the interpretation that is given to the URI that is sent in the request. In the POST method, the URI identifies the resource that is going to process the entity contained in the request; this resource can be for example a process that deals with the fields of an HTML form. On the other hand, the URI contained in a PUT request identifies the entity sent to the server.
- DELETE: The DELETE method is used to request to the server the removal of the resource indicated by the URI passed as an argument.
- TRACE: This method is normally used for tests and disgnostic of the system. The TRACE method allows a client to send a request to a server and have this message (that may have been rewritten by an intermediate) sent back as a reply to its request.

2.2. Mirroring Resources

Currently the identification of resources in the Web is realised almost exclusively through URLs (Uniform Resource Locators) [13]; URLs use the location of the resources as part of their names. For example, a resource identified by the URL http://www.anyserver.backbone/resource-path is physically located in the machine identified by the Internet address www.anyserver.backbone. In this way, URLs supply a one-to-one mapping between the name and the physical location of the copy of a particular resource. As previously argued, having only a single copy of a resource reduces the reliability of the Web, since failures in server machines and real partitions (caused by failures) or virtual partitions (caused by overload) in the communication network, can make resources unavailable to all clients, or to part of them.

An alternative to solve this problem is to supply the Web with support for the mirroring of resources, making multiple copies of a resource available to the clients. Such an architecture requires an identification scheme that is independent of the resource physical location. This scheme must allow the conversion of a given URI in one or more URLs, that identify the multiple physical copies of the same resource. This would allow

the distribution of copies of a resource in several sites spread out in the Web, allowing clients to always retrieve the “closest” copy available for a particular resource; this would also contribute for a reduction on the traffic in the Web, yielding a better performance to clients.

2.2.1. Uniform Resource Names

Uniform Resource Names, or simply URNs [18], are URIs that have the characteristics discussed above. The aim of a URN is to supply a unique, global, persistent and location independent identification, used for accessing the characteristics of a resource or accessing the resource itself. A URN is composed of a URC (Uniform Resource Characteristics), that contains resource meta-information, and one or more URLs, that define the physical location of the instances of the resource.

The URC is used to describe the resource. A URC contain information such as title and author of an HTML document published in the Web, for example, as well as resource administrative information, such as its date of creation. Currently there is no agreement regarding the set of meta-information that can be used to identify resources. However, this structure must contain enough information to facilitate the management of the resource, and at the same time must be sufficiently concise not to impose an excessive overhead in the management of resources. It is also desirable that the structure can be used for all existing resources, therefore it is necessary to prevent problems of interoperability due to the existence of different sets of meta-information for different types of resources [7].

The syntax proposed for URNs offers ways of coding characters of data in a form that can be sent in the existing protocols. All URNs must have the following syntax:

urn: <NID>:<NSS>

where NID is the Namespace IDentifier and NSS is the Namespace Specific String [18].

The NID determines the syntactic interpretation of the NSS, and distinguishes a particular URN scheme from other existing naming schemes. The NSS in turn is defined by the NID, i.e. the NSS is dependent and governed by the rules of the NID [7]. The NSS has three basic properties: i) its meaning is valid only in the context of its NID; ii) it is unique in the space identified by the NID; and iii) it has no predetermined structure - any structure that it may have, will be associated with the NID and will not be based in any physical network.

An important fact to consider is the attribution of URNs to resources. The organisation responsible for the NID of a particular URN is also responsible for this attribution, having to create its scheme of name attribution, that determines NSSs that identify every resource in a unique and global way.

The existence of a URN resolution service accessible to all clients is necessary to make it possible to use URNs. This service, implemented by a resolver, must be in accordance with the following requirements [21]:

- It must be designed to allow the definition of long-lived URNs;
- It must be able to be changed without affecting the URN structure;
- It must be scalable;
- It must be sufficiently modular so that some parts (ex. resolution algorithms, databases, protocols, etc), can be modified independently one from the others;
- It must be decentralised; an unique organisation must not have the absolute control of all the namespace of a URN.

2.2.2. Supporting Mirrored Resources

Once established the URN standard to be used, the mirroring of resources can be implemented by copying instances of the resources in the localities indicated by the URLs associated with their URNs, and providing the Web with components that are able to invoke the appropriate resolver to map a particular URN to its set of corresponding URLs. Proxies are the ideal candidates to implement these functionality. Figure 2 details the architecture proposed.

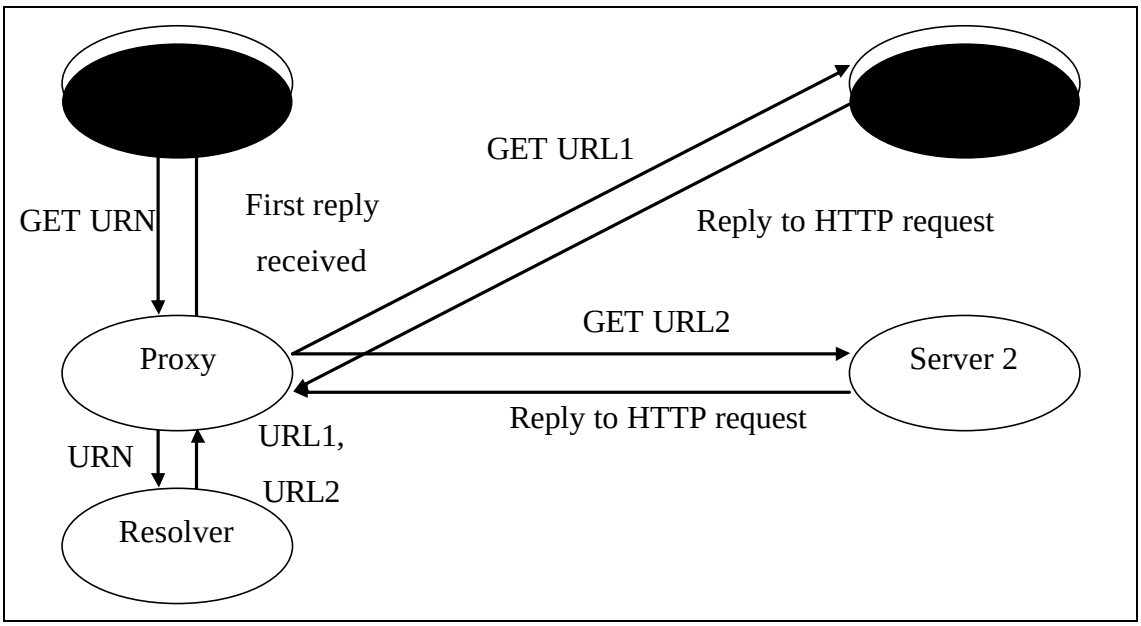

Figure 2: mirroring resources with the aid of a proxy

As it is shown in Figure 2, client requests are intercepted by the proxy that calls a resolver to discover the localities where the resource identified by the URN contained in the request is stored. Having received the set of URLs associated with the resource, the proxy can choose one of the locations and retrieve a copy of the resource using the HTTP protocol. To mask the unavailability of an instance of a resource, the proxy can make multiple requests (either in parallel, or in sequence) to the different servers that keep copies of the resource identified by the URN. The first instance successfully retrieved is sent to the client. Section 3 gives more details on the operations that can be carried out by the proxy, including a load-balancing mechanism.

Some methods defined by the HTTP protocol perform update operations on the resources stored in the server; when these methods are applied on mirrored resources, there is a possibility of introducing inconsistencies. For example, if a request for a DELETE method is not executed on all instances of the resource, it is possible that a copy of the resource continues available; similarly, inconsistencies can be introduced if a PUT request does not update all copies of a mirrored resource.

It is worth noting that even if the proxy always sends the requests to all servers that maintain instances of the resource, there is still the possibility for a failure to generate inconsistencies. The inconsistencies discussed above, can only be adequately treated if the proxy and the servers storing instances of mirrored resources can effectively collaborate in the processing of the HTTP methods. In other to make this processing transparent to servers, any additional functionality required can be implemented by a gateway as shown in Figure 3.

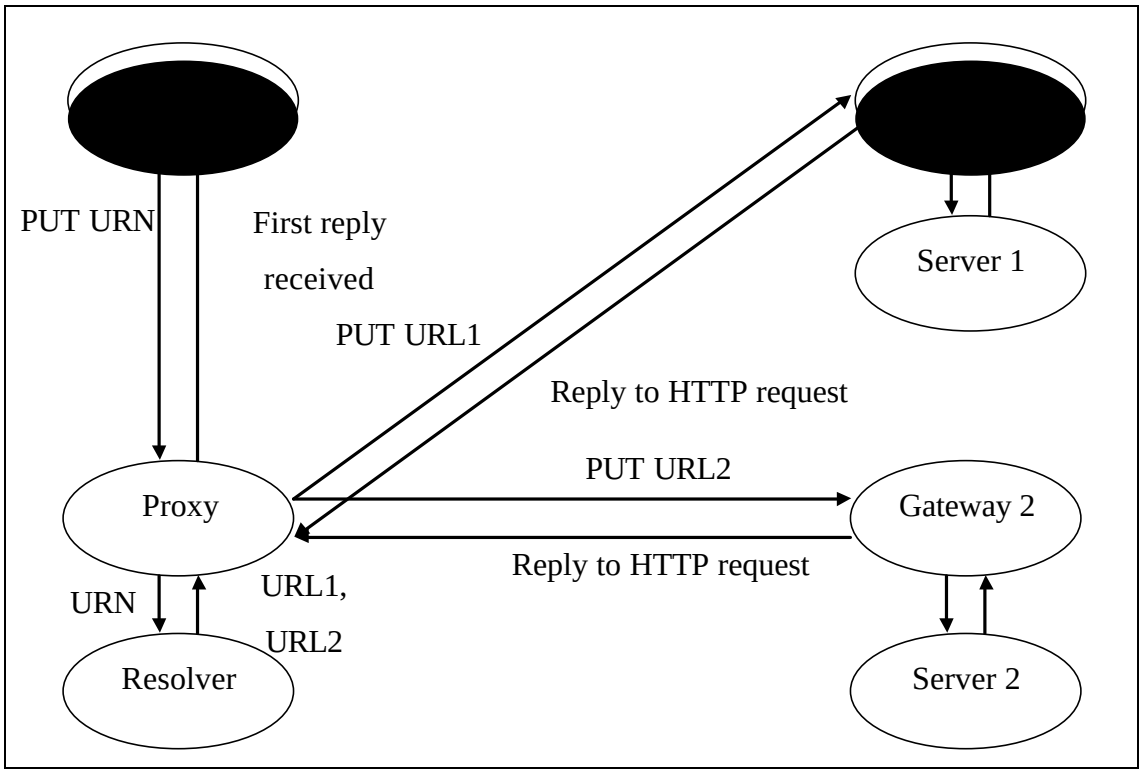

Figure 3: mirroring resources with the aid of a proxy and gateways

Next section discusses with details the functioning of proxies and gateways that must be introduced in the infrastructure of the Web to support mirrored resources.

3. Design and Implementation of a Support for Mirrored Resources in the Web

We have seen in the previous section that the infrastructure responsible for supporting mirrored Web resources must be able to allow the access to the several instances of mirrored resources; besides it must also guarantee the consistency of these resources. It was also shown that these functionalities can be transparently implemented by proxies and gateways, and that the implementations of proxies and gateways must be able to process HTTP requests containing URNs as resource identifiers; also, requests containing URLs must be processed in the traditional way. Further, an HTTP request may contain any one of the methods discussed in section 2.1.

In our project all HTTP methods are handled by the proxy and gateways, however, the functionality of the non-idempotent methods OPTIONS, TRACE and POST is not guaranteed. The replies to methods OPTIONS and TRACE are dependent on the route linking the client to the server, therefore, as this route can change from one request to another (as we will see shortly), the same request might produce a different result every time it is issued. In the case of the POST method, since the action performed to handle the method is determined by the server and is usually dependent on the resource (normally a program or script) indicated by the URI contained in the request, it is possible that inconsistencies are generated in the processing of this method; it is up to the resources implementers to guarantee that multiple calls to a resource will be handle consistently (however, in many cases no extra effort will be necessary).

The GET and HEAD methods are entirely implemented by the proxy, while the PUT and DELETE methods have their functionalities implemented part in the proxy and part in the gateways. Section 3.1 presents the URN standard used, describing the syntax of the NID and the functioning of its resolver, as well as the proxy load-balancing module responsible for the implementation of the methods GET and HEAD. Section 3.2 discusses the two consistency semantics available for mirrored resources and the impact that each of these semantics causes in the implementation of methods PUT and DELETE.

3.1. Supporting Access to Mirrored Web Resources

3.1.1. Identification Scheme for Mirrored Resources

The NID proposed for our resource identification scheme based on URNs is the WMR (Web Mirrored Resources). The WMR NID defines an NSS composed of three fields: a resource management domain field; a group field; and a resource name field. In our project, resources are grouped together and each resource group belongs to a particular management domain, thus the first field of a WMR URN is the name of the Internet management domain of that particular group of resources - this field facilitates the generation of global and unique URNs. Clustering in a single group resources that are normally accessed in succession (a frequent access pattern), increases the granularity of the mirrored information, diminishing the number of resolution requests to the URN resolver. The last field of the NSS is used to identify a particular resource within a given group. The syntax of the WMR NSS is the following:

NSS ::= <management domain> / <group> / <resource name>

As an example, we can have the following URN:

urn:wmr:www.anyserver.backbone/students/paul.html

that could be used to identify an HTML page that contains the information of the student Paul; the copies of Paul's home-page are stored in all servers that are responsible for mirroring the group students, whose management domain is www.anyserver.backbone.

3.1.2. Resolution Service

The name resolution service is implemented by a resolver and must follow the principles presented in section 2.2.1. In our project the resolver is divided in two parts, one local and another global. The local URN resolver is implemented within the proxy itself. After receiving a URN resolution request, the local resolver extracts from the URN the name of the management domain and the name of the group. With this information, the local resolver first verifies if the mapping can be carried out using the information kept in its cache; when this is not possible the local resolver issues a request to the global resolver passing the resource's management domain and group as parameters.

We propose that resources should be mirrored within some well defined set of networks forming a boundary which limits the scope of a particular WMR domain. Using this approach, each WMR domain must implement its global URN resolver. For instance, one could propose WMR.BR to be the WMR domain used to mirror Web resources within the networks contained in the Internet domain .br, and within this domain create a service for resolving WMR.BR URNs. The service provided by the global resolver must be highly available, so that it is always possible to obtain a reply to requests that are submitted to it.

In our project the global resolver is implemented using the well-understood Internet DNS (Domain Name Server) service [1]. For each WMR domain it is necessary to create an Internet sub-domain (for example, wmr.br for the WMR.BR URN domain) responsible for resolving URNs for that particular WMR. High availability of the resolver service provided by the DNS service is attained by replicating the service in carefully chosen servers of the corresponding WMR domain. The DNS service already offers the necessary support for the replication of the servers of a particular domain.

The databases of a DNS server are composed of several types of registers, where a number of information is stored. Domain names, such as www.anyserver.backbone, are used as keys for searches on the information stored by DNS servers. In our project we use TXT registers of the DNS (within which any literal information can be stored and associated to a domain name) to store the names of the Internet domains that keep copies of a particular group of resources. Figure 4 shows an extract of the database of a global DNS resolver.

```
project.dcc.ufmg.br     TXT "www.dcc.ufmg.br, www.pmbh.org.br"
research.di.ufpe.br     TXT "www.di.ufpe.br, www.ic.ufrgs.br, www.dcc.ufmg.br"
```

Figure 4: extract of the database of a global DNS resolver

In order to invoke the service of the global resolver, the local resolver concatenates the name of the group to the name of the management domain of the resource and issues a request for a TXT record, getting as reply a list with the name of the servers where the copies of the resources associated with that pair (management domain, group) are located. Having received the names of the servers, the URLs can be formed in the following way:

http://<server name>/<resource name>, for each server name received.

Taking the database shown in Figure 4, and given the URN urn:wmr.br:www.di.ufpe.br/research/projectX/papers.html, the request to the global resolver implemented by the DNS server of the Internet domain wmr.br would be made with the key research.di.ufpe.br, which would return the following server addresses: www.di.ufpe.br, www.ic.ufrgs.br and www.dcc.ufmg.br; the URLs formed would be then:

http://www.di.ufpe.br/projectX/papers.html,

http://www.ic.ufrgs.br/projectX/papers.html, and

http://www.dcc.ufmg.br/projectX/papers.html.

3.1.3. Load-Balancing Mechanism

Once identified the set of URLs associated with a particular URN, the proxy must start the processing of the method contained in the request issued by the client. In the case of the methods GET and HEAD ideally the proxy should send the request to the server which could answer more quickly to the proxy request. Unfortunately the proxy has no way to foresee neither which servers are available, nor what is the load that is being

submitted to a particular server, much less what is the traffic that is being generated through a particular route. Another possibility then, is to establish multiple connections in parallel with all the servers that store copies of the resource, and return to the client the reply sent by the first server to answer the request. The problem with this approach is that the extra traffic generated increases the load in the network, reducing the performance, or even, in an extreme situation, causing a virtual partition of the proxy with the rest of the network.

In our project, we have decided to implement a dynamic load-balancing mechanism that tries to reduce the response time of the requests made by the clients without causing a considerable impact on the network performance.

The load-balancing mechanism works in the following way: the first time that the proxy makes a mapping of a URN on the corresponding URLs, it sends the request to all servers that keep copies of the resource; the proxy then measures the time that each server takes to answer the request (those which had not answered within a predefined timeout interval, have this timeout annotated as their response time) and sends to the client only the reply of the server that answers more quickly (this reply is sent to the client as soon as it is available to the proxy); future requests to mirrored resources in the same management domain and group of the mapped URN, will be retransmitted only to the server that has the fastest response time.

The proxy always measures the response time of a request (even when the request is sent only to the fastest server), and always uses the server that has the smallest response time for the last request that was sent to it. When a request is sent to a single server and a reply is not received within the timeout interval, the proxy keeps issuing requests to the next fastest server, until a reply is received, or all servers have been attempted, when an error is returned to the client.

3.2. Supporting the Management of Mirrored Resources

Mirrored resources can be updated via methods PUT, POST and DELETE. As the processing of methods PUT, POST and DELETE modifies the state of the resources stored on the servers that execute them, in order to maintain all copies of a resource consistent, it is necessary to guarantee that all servers process all the requests containing methods PUT, POST or DELETE, and that these requests are processed in the same order. Therefore, these methods must be handled differently from the way discussed in

the previous section. The next sub-section shows how we have implemented these methods in our project.

3.2.1. Consistency Semantics for Resource Update

The consistency of mirrored resources can be affected by several factors: the unavailability of part of the servers that store copies of the resources; the possibility of concurrent execution of update methods on the same resource; and the possibility of a network partition isolate some copies of the resource, creating a situation where execution of multiple update methods on a particular resource alter different subgroups of copies of the resource. There are several protocols that can be used to keep the consistency of replicated information; they can be broadly classified into two groups: pessimistic and the optimistic protocols.

The pessimistic approach [4, 11, 9] is concerned with the maintenance of replica consistency even if that comes to compromise their availability. In this type of strategy the modifications are carried out synchronously and use a single transaction to modify all copies. The problem in performing synchronous updates is that if at least one copy is not available, or fails during a modification, the transaction either aborts or stays blocked until the unavailable copy recovers.

Optimistic strategies [5], on the other hand, do not impact the availability of the replicated data. Any operation can be executed in any partition that contains a copy of the resource. They assume that conflicts rarely happen. In this approach, the updates can be carried out asynchronously, i.e. it is allowed for each copy to be modified by an independent transaction; in this way, if the operation on a particular copy fails, the other operations on the other copies can be executed independently. Inconsistencies caused by updates carried out in distinct partitions and the effects of two concurrent transactions operating on copies of the same resource must be treated afterwards in an application dependent way.

The optimistic semantics satisfies the requirements of the majority of the resources currently published in the Web, over which read methods such as GET and HEAD are executed much more often than update methods such as PUT and DELETE; besides, for many clients, reading outdated information does not cause any concern. However, new utilisation patterns for the Web are appearing each day, where update operations are as frequent as read operations, and where the maintenance of the consistency of the

information is crucial. We have decided, therefore, to allow the maintenance of mirrored resources with the two types of consistency semantics discussed above. Thus, copies of resources with pessimistic consistency semantics will only be changed by update methods if all resource replicas are available. On the other hand, resources with optimistic consistency semantics can have their copies updated independently, even when some of the copies are unavailable. It is worth noticing that it is up to resource providers decide whether a particular resource should be replicated using one or the other consistency semantics.

Request messages with the DELETE method have a header that specifies the consistency semantics of the resource (note that the HTTP protocol does not define this header; it is introduced by the proxy and handled by the gateways transparently to client and servers that implement HTTP). Request messages with the POST method contains a header with the same information mentioned above, plus the version of the resource (this information is used to resolve collisions, as we will see shortly). A request message for the PUT method is similar to a POST request, with the difference that it must also include the entity that represents the data of the resource to be stored at the server. When the resource does not exist, and is being created by a PUT request, the header of the request message also contains the addresses of the servers that will store the copies of the resource. In this case, the proxy must update the resolver database, before sending the requests to the servers. For this operation to succeed, the DNS server that implements the global resolver of the corresponding WMR domain must allow dynamic updates of its database [16, 17].

When the consistency semantics of the resource is optimistic, the processing of PUT, POST and DELETE methods is very simple. After the resolution of the URN (when it is necessary), the proxy sends the requests to all servers that hold a copy of the resource; this is performed in successive transactions, one for each replica; if some server is not available (i.e. if a transaction aborts), the proxy updates a list of pending requests (kept in stable storage), that is processed from time to time in background. Client requests issued by the proxy are received by gateways; each gateway processes the request and sends the reply to the proxy; the first successful reply received by the proxy is then returned to the client, while the others are discarded (if all transactions fail, the proxy returns an error message to the client, and discards any pending request that has been logged for that request).

The operations carried out in the databases of the Web servers that keep mirrored resources can be executed concurrently, thus simultaneous accesses to a file in the same database can cause interferences. These interferences can be easily prevented through some concurrency control mechanism based on file locking [2]. However, even with the implementation of the concurrency control mechanism described above, the update of resources with optimistic consistency semantics can generate collisions. A collision occurs when the same resource, that is physically mirrored in two or more servers, is modified during an asynchronous period of latency, i.e. after the time where the first modification happened, a second modification occurs and is processed in a server before the propagation of the first modification has been concluded.

This situation can happen if the update of a resource with optimistic consistency semantics is performed when there is a partition in the network. In this case, as discussed previously, the proxy that performs an update continues sending requests in background to the inaccessible gateways, until these are accessible and execute the update. As the same resource can be updated by more than one client, it is possible that a copy of a resource is updated in a different order in respect to another copy of the same resource.

The problem of collisions in our system is solved in the following way: firstly, when a mirrored resource is created, it is established a degree of priority between each replica; each time a mirrored resource is updated its version number is incremented; in this way, collisions are detected whenever a request is issued to update a request that already has a version number larger than the version number contained in the request; when a collision is detected all pending requests for that resource, but the one to the higher priority replica, are suspended; if the higher priority request is ever successfully handled, then all suspended request for that resource are rescheduled (this request may now overwrite a previous update performed by another client); when an attempt to update the higher priority replica fails because the replica has already been updated by another client, then all suspended requests for that resource are cancelled; in this way the valid update is always the one carried out by the client that first succeed in updating the highest priority replica.

When the resource consistency semantics is pessimistic, the processing of update requests is a little more complicated. The functioning of the proxy is similar to the previous case, i.e. the proxy sends the requests to all servers that keep copies of the

resource, however, this time all requests are sent within the same transaction scope. Initially, each gateway must update a log (in stable storage) with the operations that must be carried out on the resource; after that the gateway must reply to the proxy that it is ready to carry out the update; when the proxy receives these reply messages from all gateways it sends a new request to all gateways asking them to commit their update; then the gateways perform the operation and remove the request from their logs. If at least one gateway does not answer to the proxy within an specified timeout interval, then the proxy sends a message asking the gateways to abort the operation and returns an error to the client. If a gateway that has sent a ready to commit message does not receive an abort or a commit message from the proxy within a predefined timeout interval, then it contact the other gateways to find out rather it should commit or abort. Failures in the proxy as well as in the gateways are treated when these components recover and process their logs. It is worth noting that it is not possible to have collisions when the resource has a pessimistic consistency semantics, since the operation either is carried out in all servers or it is not carried out at all.

4. Related Work

The Internet Engineering Task Force has formed a working group to study the URN issue. There are several propositions being developed by this group for the implementation of systems that support the URN naming scheme. Our approach follows the trend established by the majority of these proposals that use the DNS service to implement URN resolvers [19]. However, unlike these approaches which concentrate their efforts only in the development of the URN infrastructure (i.e. naming schemes, resolvers, etc.), our work goes a step further and proposes a way of adapting the current infrastructure of the Web to allow access and management of mirrored resources. In our system this is performed in a completely transparent way by clients that can continue accessing the Web using conventional browsers and resource providers that use conventional Web servers to store and manage their resources.

Another approach pioneered by a group of researchers at the University of Newcastle has opted to extend the current model of the Web [6]. Their design is very much influenced by the object oriented technology and argues that a Web resource should have a more complex interface with customised methods, instead of the simple file oriented interface supported by the HTTP protocol. Although the ideas defended by these researchers can solve many problems of the Web and add new perspectives for its

utilisation, they impose a complete change in the underlying technology supporting the Web, which might turn out to be unfeasible.

5. Conclusions

The enormous growth of the Web in the last few years has prompted the interest of the organisations that started to rely on a new media to offer information and services in a world-wide scale and at an extremely low cost. Following this trend, each day new businesses opportunities are made possible via the Web. However, in some countries, deficiencies in the infrastructures of the communication networks and in the information and access provision services, has brought problems to the Web users, particularly for those that provide services via the Internet, as these deficiencies diminish the availability of the resources stored in the Web.

In this work we have presented the design of two components that can be easily introduced in the current infrastructure of the Web, to minimize the effects of resource unavailability. Higher availability of the resources is attained through the mirroring of resources that can be accessed via alternative routes. Using a proxy that implements a URN resource identification scheme, it is possible to tolerate failures in servers and communication channels in a transparent way; moreover, the management of the resources can be implemented trivially, using methods PUT and DELETE of the HTTP protocol.

Prototypes of the proxy and the gateway have been developed and tested at our local environment. We have chosen a particularly unreliable site to replicate; initial results have shown availability figures close to 100% for a three-fold replicated experience. Since our tests were mainly local, i.e. within our Campus' LAN, it was not possible to notice any substantial variation on the response time of client's requests.

We are now installing the prototypes in several sites of both an academic and a commercial backbone (the RNP - Brazilian National Research Network, and the Embratel - Brazilian Telecommunications Company, backbones, respectively); our goal is to test the prototypes in a real production environment and assess the performance impact of the load-balancing mechanism as well as the fault tolerance properties of the system. We also plan to test other load-balacing mechanisms.

References

[1] P. Albitz, and C. Liu. DNS and BIND, 2nd edition. O'Reilly & Associates, Inc., USA, 1997.

[2] P.A. Bernstein, V. Hadzilacos, and N. Goodman. Consistency Control and Recovery in Databases Systems. Addison-Wesley, USA, 1987.

[3] T. Berners-Lee, R. Cailiau, A. Luotonen, H.F. Nielsen, and A. Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76-82, 1994.

[4] P. Danzig, D. DeLucia, K. Obraczka, and E. Tsai. A Tool for Massively Replicating Internet Archives: Design, Implementation, and Experience. In Proceedings of the *16th International Conference on Distributed Computing Systems*, pages 675-664, 1996.

[5] S.B. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in Partitioned Networks. *ACM Computing Surveys*, 17(3):341-370, 1985.

[6] D. Ingham, S. Caughey, and M. Little. Fixing the Broken-Link Problem: The W3Objects Approach. *Computing Networks and ISDN Systems*, 28(11):1255-1268, 1996.

[7] R. Iannella, H. Sue, and D. Leong. BURNS: Basic URN Service Resolution for the Internet. In Proceedings of *the Asia Pacific WWW Conference*, Hong Kong & Beijing, 1996.

[8] P. Jalote. Fault Tolerance in Distributed Systems. PTR Prentice Hall, 1994.

[9] S. Jajodia, and D. Mutchler. A Pessimistic Consistency Control Algorithm for Replicated Files which Achieves High Availability. *IEEE Transactions on Software Engineering*, 15(1):39-46, 1989.

[10] E.D. Katz, M. Butler, and R. McGrath. A Scalable HTTP Server: The NCSA Prototype. *Computer Network and ISDN Systems*, 27(2):155-164, 1994.

[11] M. Little, and D. McCue. The Replica Management System: a Scheme for Flexible and Dynamic Replication. In Proceedings of the *2nd Workshop on Configurable Distributed Systems*, Pittsburg, USA, 1994.

[12] T. Berners-Lee. Universal Resource Identifiers in WWW. RFC 1630, CERN, 1994.

[13] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators. RFC 1738, CERN, Xerox Corporation, University of Minnesota, 1994.

[14] T. Berners-Lee, R.T. Fielding, and H.F. Nielsen. Hypertext Transfer Protocol - Hypertext Transfer Protocol - HTTP 1.0. RFC 1945, MIT/LCS, UC Irvine, 1996.

[15] R.T. Fielding, J. Gettys, J.C. Mogul, H.F. Nielsen, and T. Berners-Lee. Hypertext Transfer Protocol - HTTP 1.1. RFC 2068, DEC - Digital Equipment Corporation, MIT/LCS, 1997.

[16] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. Dynamic Updates in the Domain Name System (DNS UPDATE). RFC 2136, ISC, Bellcore, Cisco, DEC, 1997.

[17] D. Eastlake. Secure Domain Name System Dynamic Update. RFC 2137 CyberCash, Inc., 1997.

[18] R. Moats. URN Syntax. RFC 2141, AT&T, 1997.

[19] R. Daniel, and M. Mealling. Resolution of Uniform Resource Identifiers using the Domain Name System. RFC 2168, Los Alamos National Laboratory/Network Solutions, Inc., 1997.

[20] B. Sauers. Designing Highly Available Cluster Applications. Technical Report, Hewlett-Packard Company, 1996.

[21] E. Slottow. Engineering a Global Resolution Service. MEng. Thesis, ANA - Advanced Network Architecture, MIT, June 1997.