Quality of Service in Internet Web Services: Issues and Solutions

Marco Conti, Enrico Gregori, Willy Lapenna*

Consiglio Nazionale delle Ricerche
Istituto CNUCE, Via G. Moruzzi 1, 56100 Pisa - Italy
Tel: +39 050 315 3059 Fax: +39 050 3138091
e-mail: {Marco.Conti, Enrico.Gregori, Willy.Lapenna}@cnuce.cnr.it

Abstract

Several techniques have been developed to meet the demand for faster and more efficient access to the Internet. Among them a main role is acquired by the replication of the Web services. Replication can be addressed both by a cluster of servers and by servers geographically distributed in the Internet. In this paper we deal with geographical replication. Two approaches are possible for geographical replications: server-side and client-side. We deal with the client-side approach to face the QoS improvement issue. We classify and compare the client-side techniques that are present in literature. From this analysis we design a promising technique named MinimumLatency-based (MIL-based) approach to obtain the best performance in terms of perceived user's QoS.

1 Introduction

Currently, a large fraction of users accesses network resources through web clients/browsers. The majority of today's Web services are generally not concerned about the levels of quality of service presented to their users. However, it exists a small but increasing number of web sites that to maintain their popularity and reputation, tends to be concerned about the quality of service experienced by their users. Furthermore, new web applications can require the delivery of multimedia data in real time (e.g. streaming stored video and audio), and the information transfer through Internet is becoming one of the principal paradigm for business: electronic sales, banking, finance, collaborative work, are few examples of this. The Quality of Service (QoS) perceived by its users is thus becoming a dominant factor for the success of an Internet based Web service. The principal QoS attributes these users perceive include those related to the service "responsiveness", i.e. the service availability and timeliness. A service that is frequently unavailable may have the effect of tarnishing the reputation of the service provider or result in loss of opportunity. Furthermore, from the user's perspective, a service that exhibits poor responsiveness is virtually equivalent to an unavailable service.

The performance perceived by the users of a Web service depends on the performance of the protocols that operate between web clients and servers. The main issue is to minimize the User Response Time (URT), i.e. the elapsed time between the generation of a request from a browser, for the retrieval of a Web page, and the rendering of that page at that browser's site. Hence, the URT includes both the communication and the processing delays involved in servicing a browser request.

Solutions currently investigated in the literature range from new protocols to enhance the network QoS (e.g. differentiated services) to mechanisms running into the end systems to enhance the QoS perceived by the users (e.g. caching, prefetching, Web servers' replication, etc.). While, in the medium-term, it is envisaged that Internet evolves to a universal transport service that will be able to provide (when it is required) a delivery of the traffic with QoS guarantees, currently, an IP network provides a best effort service. Hence, in the short term the middleware must cope with the network insufficient bandwidth and high latency. The issues that need to be addressed to improve the QoS are: i) the decrease of the document retrieval latency, ii) the increase of data availability, for example, through replication of the information, iii) the reduction of the amount of data to transfer, and iv) the redistribution of network accesses to avoid network congestion.

Several techniques have been developed to meet the demand for faster and more efficient access to Internet Web services. These techniques include caching, prefetching and pushing. However they are only a partial solution to introduce QoS in Web services. The more advanced approach of autonomous data replication has been recently introduced to attack the problem ([Bye99], [Car96], [Car97], [Con99], [Con01], [Dyk00], [Ghi01], [Rod00], [Say98], [Say99], [Zeg00]).

The primary aim of replication is to increase data availability and to reduce the load on a server by balancing the service accesses among the replicated Web Servers (WSs). Specific strategies that have been proposed to distribute the processing load among replicated WSs. The vast majority of the load distribution strategies for replicated WSs assume that these WSs are replicated within a cluster of servers, and that the access to them is governed by the DNS server of the cluster. These strategies aim

* corresponding author

to maximizing the WSs throughput, rather than minimizing the user response time, and pay little attention to such issues as client latency time over the Internet.

A more general approach, based on the geographical distribution of the information to increase both the service availability and the timeliness, has been recently proposed. This approach is based on the implementation of a Web service, by replicating Web servers (WSs) at distinct sites distributed across the Internet, rather than within a single cluster, and ensuring that each client (or browser) takes advantages of this replication. This is a new and promising area both from a research (see [Con01]) and industrial (see for example Akamai) standpoint. It is worth noting that the strategies such as caching pushing, and prefetching, used for reducing the response time in non-replicated web services, can be adopted to enhance the performance of a replicated web service, as well. Therefore, combining the paradigms of caching, prefetching and replication promises to be a synergetic way to provide QoS guarantees to web-based applications.

When we consider replicated Web servers we intend that all the content of a server is exactly present at the other replicas. This may introduce consistency problems. Hereafter, we will not consider the data consistency problem introduced by the replication.

In this paper we will analyze and contrast several solutions (both client-side and server-side) for achieving QoS improvement in Web servers' replication, focusing our attention on the client-side solutions. We define in detail the different phases to download data from replicated WSs. Our claim is to address an optimal strategy that has to be adopted by clients to reach a QoS improvement. This optimal strategy is achieved by identifying (for each phase in which client is involved) the best choice when it contacts a site and downloads data.

This paper is structured as follows. In the next section we show the replication strategies in term of local or geographical replication. We define and contrast server-side and client-side solutions. In Section 3 we deeply analyze the client-based geographical replication approach. In addition we propose our choices to build an "optimal" technique. The features of this technique that we call MIL-based approach are showed in Section 4.

2 Approaches to Replicated Web Services

Currently, there are two different approaches to deploy a replication of a Web service and it depending on the place where the replication is managed:

- ***Servers' clustering***
- ***Geographical replication.***

2.1 *Servers' clustering*

We indicate with a Web cluster a set of servers housed together in a single location that represents a Web site. In a Web cluster there is an entity, i.e. a front-end component, that is called in many different ways such as ***Web switch*** or ***dispatcher*** for example. This dispatcher behaves as a proxy, that intercepts all the incoming requests. The Web site is visible to the network only through the dispatcher IP address, called ***Virtual IP address*** or ***cluster address***. When a client loads in his browser an URL, this makes a DNS request to resolve the URL. The IP address returned from the DNS (to resolve the host name) is the Virtual IP of the dispatcher. So all the requests to that URL are sent to the dispatcher of that Web site. Then the Web cluster is visible only through one IP address that can hide the presence of dozens of machines that constitute the cluster. When the dispatcher receives a request, it chooses the server into the cluster that will process the request. A dispatcher can appear to the servers' in the cluster like a switch that processes only the incoming data, or like a network gateway that processes the outgoing data too. The incoming requests are spread among the pool of servers in different manners. This spreading is more easily performed when HTTP protocol is used. In fact it is related to small requests and it doesn't save state information, so there is no need to maintain consistency among the servers [Sch00].

On the market there are many products based on this approach such as: Cisco LocalDirector [Cis97], Dynamic Local Director's Hewlett-Packard in partnering with Cisco [HP-a], Alteon ACEdirector [Nor-b], Arrow Point's Web Switches, IBM's Web Accelerator, HP Open View.

2.2 *Geographical Replication*

In the geographical replication we have replicated WSs that are spread in the whole Internet, in multiple locations. In reality, instead of having geographically scattered WSs, we have several geographically distributed clusters. Each cluster is seen from the outside like only one server. For this reason when we talk about WSs it is clear that behind each WS there is a cluster of servers. When we have several replicated WSs we need to define a policy to bind the client that makes a request to one or more of these servers. This selection policy can be implemented at two different places. According to this location we can define two main approaches: i) server-side approach and ii) client-side approach. The client-side and server-side approaches are distinguished by the place in which the choice to bind the

client is taken. If it is taken close to the client we have a client-side solution, otherwise we are speaking about a server-side solution.

2.2.1 Server-side solutions

With the name server-side solutions we intend all the techniques that manage the traffic distribution towards the replicated WSs without the client's decision. In this case the choice is made by servers, routers or DNS, etc. Commercial products deployed by Cisco[1] and Nortel[2] (e.g. Distribute Director [Cis99] and ACEdirector with GLSB [Nor-c], respectively) are examples of this approach. Generally, these techniques are accomplished by deploying appliances that are close to the WSs. Unfortunately, as often happens for the commercial products, the technical information that are available are limited because they are proprietary solutions. Therefore their analyses cannot be done in an accurate manner. Starting from a qualitative analysis, we found some common features of server-side solutions to reach the QoS issues. These features are the following [Con01a]:

- They are implemented by using a specific appliance for every replicated site.
- A replicated site generally coincides with a cluster and the appliance operates as the cluster front-end.
- The appliances can be used as authoritative DNS servers or they can redirect the HTTP requests by HTTP redirection.
- Front-end servers exchange information by using a lightweight UDP based protocol.
- The front-end server checks the availability of the servers belonging to the cluster it manages by probing them.
- Fault tolerance is reached by increasing the number of the appliances at the same site.
- No changes in client or server software/hardware are required.

There are other features not in common to all the products. However, these features are not relevant for our analysis.

2.2.2 Client-side solutions

We refer to a client-side solution when the decision to bind a client to a particular server (or more servers) is made at the client-side, i.e. close to the client (e.g. into the client browser, client-side proxy, etc.). A client-based approach has the advantage that the client-side solution has an overall network vision (in term of congestion of the links involved in the data transfer between server and client) that coincides with that observed by the browser. Another advantage of this approach is that it seems appropriate when the group of server is heterogeneous or widely dispersed across the network [Dyk00]. Naturally, this approach makes the (easily achievable) assumption that when the browser makes a DNS request to obtain an URL resolution, it obtains all the available IP addresses of the replicated WSs [Che01]. Generally, the client-side solutions are deployed via a software included in the browser package or like an applet to download.

2.2.3 Server-side vs Client-side

By contrasting server-side and client-side solutions we can find that the server-side implemented solutions have these main limitations:

i) they require proprietary hardware and/or software;
ii) scalability is limited and generally requires additional appliances.

In addition, the server-side solutions to implement a latency-based metric require to probe the clients. This probing towards clients can present problems due to firewalls i.e. the ping doesn't answer even if the host contacted is reachable or it can give erroneous results.

Moreover, it may happen (like in Cisco Distribute Director [Cis99]) that a decision (or a measure) is based on the nameserver's location and not on the client location. This can lead to poor decisions when clients and DNS servers are not proximal. It has been shown that often the site closest to the client's DNS resolver is not so close to the client: 60% of users are not in the same network as their DNS [Sha01].

There are products such as Nortel's Personal Content Director (PCD) that aim to address this problem. Specifically, by deploying PCD, it seems possible to detect the real client proximity and to select the ***best*** server taking into consideration the link latency and the packet loss, as well as the content availability, server ***health*** and load [Nor-a]. Unfortunately, due to the lack of documentation, it is not clear how it is achieved.

The client-side solutions are not affected by the above limitations. In the next section we will perform a depth analysis of the client-side solutions.

---

[1] www.cisco.com

[2] www.nortel.com or www.alteonwebsystems.com

3 Client-side solutions

In the literature we can find several client-side techniques to face the problem of improving the QoS in a geographically distributed WSs environment. An important issue of servers' replication is to evaluate the degree of parallelism that has to be pursued, i.e., how many replicated WSs are used to solve a user query. Obviously, the degree of parallelism may change by enlarging the size of the pool of replicated servers. The approaches currently proposed in the literature to access a replicated Web service are substantially two: i) the client downloads data from one server only; ii) the client downloads data from all the available servers. In the former case the goal is to define an algorithm ensuring that each client (or browser) gets bound to the "most convenient" WS replica for the whole download time ([Say98], [Say99], [Car96], [Car97], [Dyk00], [Zeg00], [Con99], [Con01]). Hence, in this case, the main design issue is the algorithm chosen to select the ***best*** server that will be used for downloading data. This choice is fundamental to obtain the best (client-side) responsiveness of the Web service.

On the other hand, if all the servers are used to download data, the attention is on the way the replicas are used ([Bye99], [Rod00], [Ghi01]), i.e., on the portion of data requested to each replica.

In this work we analyze the client-side approaches according to the following criteria:

- ***URT minimization***
  User Response Time (URT) is the elapsed time between the generation of a request from a browser, for the retrieval of a Web page, and the rendering of that page at that browser's site ([Con99], [Con01]). The minimization of URT is the main goal to be pursued in every approach because it is strictly related to the QoS perceived by users.
- ***System scalability***
  It is an intrinsic requirement of Web Servers' Replication: we expect that the number of replicas increase with the Web users' increase.
- ***Robustness***
  Tolerance to fault conditions is fundamental to guarantee the availability of a Web service, This is a key issue in business transactions.
- ***Orthogonal***
  The client-side techniques have not to be in contrast with server-side solutions. That is, if we use both server-side and client-side solutions, the performance must not decrease with respect to the use of a server-side solution alone.
- ***Load-balancing***
  Solutions should be oriented to distribute the load among servers by avoiding to congest some of them and to have the others underutilized.
- ***Net-balancing***
  With Net-balancing we indicate the attempt to distribute network traffic on the net by trying to not increase the congestion.
- ***Adaptability***
  We consider this feature with respect to dynamic conditions. A dynamic technique fits well when the system condition changes during the transfer phase. This issue is fundamental in mobile-computing environments.
- ***Overhead***
  For each technique we have to evaluate the overhead it produces and the changes it may require in the client, in the network, and in the servers.

To make a comparative analysis among the different approaches, and to obtain a classification of the techniques already deployed, we divided the access to a replicated Web service in four principal phases. These phases are: i) the initial probing, ii) the server(s) choice, iii) the downloading method, and finally iv) the download monitoring. The initial probing aids to assess the responsiveness and availability of the contacted servers. The server choice has to define which servers (one or more server) to use in the download of data. The downloading method defines how data must be downloaded from server(s). The client in fact can download data as single entities like single objects of a web page (e.g. an entire gif file or an applet) or single-object fragments (note that, in this case, few scattered bytes of a picture are not renderable). The download monitoring is used to support fault tolerance and QoS improvement by continuously measuring the server availability and the current performance.

The claim of the comparison among the different techniques at client-side proposed in the literature, is to find the best choice in each phase. By gathering these choices we design an optimized technique whose target is to minimize the server response time and to increase the availability of the system.

3.1 Phases in the client-side techniques

i)The initial probing

The initial probing phase is fundamental to obtain information about the servers and the network links involved in the data transfer. By using this initial probing phase we can know which servers are on, e.g.

in the case in which DNS information is stale. Moreover, it is worth noting that a server can send information about its own state to the client, or the client itself can get these information by querying, for example, the server on a port different by HTTP port. This initial probing phase is very important especially for a dynamic selection algorithm whose choice is based on the information collected in this phase. During the initial probing phase we have to decide what kind of information[3] we want to collect and which servers are to be contacted. The information has to permit to identify the best choice among all the available WSs.

To perform the initial probing, we need to use a tool that satisfies the following requirements [Car96]:

- It runs at the application level and can be used by any program, e.g., the browser.
- It has to have a minimum impact on the net and on the server to which the probing is made.
- It doesn't need information coming from the network or the server.
- It has to be robust.
- It has to be accurate as much as possible and to give information in a time interval that is short with respect to the URT.

The simplest technique for the initial probing is based on the Round Trip Time (***RTT***) measurement. This measurement can be performed at different levels in the protocol stack. A simple RTT estimate is provided by the ***ping*** program that is based on an ICMP request [Ste96]. As the time latency has a role greater than network congestion for small size files an approach by using ping can be effective. Naturally when the file size increases the network congestion prevails on the link latency time [Car97]. In [Car97] authors assert that for small size files it is possible to obtain appreciable results. The use of more pings, up to 5, depending on the file size, is useful to estimate the bandwidth between the client and WSs. We can also use RTT measurements at HTTP level. The ***HTTP Request Latency*** is defined as the interval that starts from the time an http query is issued and ends when the first byte in the HTTP packet arrives to the client. This kind of request can give information about the HTTP/1.1 latency that has a high correlation (0.73) with the HTTP response time. Note that using an ICMP ping we have a lower (with respect to the HTTP Request Latency) correlation value (0.51), and this value is further reduced if we use as the initial-probing metric the number of hops between the server and the client [Say98]. The HTTP Request Latency can provide a good estimate of the HTTP response time because most of the files (90%) involved in the data transfer from a Web server have a size less than 10-20 KBytes ([Say98], [Arl00]). To measure the HTTP Request Latency**,** the client can make an HEAD request by exploiting this feature of the HTTP protocol ([Ghi01], [Fie97]). The server replies to this request sending the information on the content of the web page the HEAD request is referring to. By this approach the client can also have both an assessment of the server load, and a partial estimate of the bandwidth between client and server. Two tools are available to provide a precise estimate of the bandwidth: ***bprobe*** and ***cprobe*** [Car97]. By using these tools is possible to establish available bandwidth in presence of network congestion or not. Even if they do not require any additional software on the server, ***bprobe*** and ***cprobe*** has a drawback in term of network overheads, e.g. ***cprobe*** needs to send 30000 bytes to each server to assess the available bandwidth.

Delay and packet loss measurements can also be obtained by exploiting ***Poip*** [Bar99]**.** This tool injects UDP packets into the network following a Poisson process. This tool is based on the consideration that sampling at interval fixed by a Poisson process produces observations that match the time-averaged state of the system. However, ***Poip*** cannot predict TCP-connection performance, as these performance are highly affected the TCP congestion-avoidance and congestion-control mechanisms [Bar99].

Another issue that the client needs to solve (before performing the initial probing) is related to which server(s) to contact. Initial probing can be made to all replicated to select (see point ii) below) a subset of servers that will be used in data downloading. This approach provides information about all replicated WSs in term of servers' load and the network congestion in the links between the client and each server. However, it can introduce a large overhead on the client, the servers and the network. Client overheads occur when it injects measurements' data into the network, it receives packets that do not contain useful information, and finally when it processes the measurements. Obviously, the overhead is proportional to the amount of these data. From a network standpoint, the overheads are represented by the packets sent from by the server to the client that do not contain useful data. However, even if the client uses useful data to make estimations (e.g. by downloading the first page from all servers) there are still some overheads at client side to elaborate the measurements. A critical aspect in the initial probing is the overhead produced in contacted servers because this may increase the server load thus negatively affecting the overall replicated-service performance.

On the other hand, the client can probe only a subset of all available replicas. By exploiting a subset of replicated WSs we can reach performance close to optimum [Mye99]. In this case the problem is to define this subset. To define it several choices are possible. We can assign, in a simple way, equal probability to each server. We can use a static approach taking into account historical download state information in the choice of the servers, but these information are not always available. Other choices can be based on geographical proximity or the minimum number of hops. The most promising choice is

[3] We can request from servers many different information but hereafter this aspect will be not further analysed.

to define a subset randomly and to select the most responsive servers in this subset. The dimension of this subset must be investigated.

ii) Server(s) choice

One of the most important issue in the replication of a Web service is the choice of the server(s) from which the client downloads data. After initial probing, the client can choose the server(s) to download data or, if necessary, the server subset that may be queried to have more precise information (more precise probing). On the base of information obtained by the first phase, the client has to decide from which server(s) it downloads data: i) one, or ii) more servers.

When only one server is chosen system performance are subordinated to this choice. In addition we have a system with a single point of failure. By using more servers we avoid these drawbacks, and potentially we can decrease the URT ([Bye99], [Rod00], [Ghi01]). However, depending on the used technique, this approach can lead to worse performance due to an increase in the overheads experienced by the client, the network and in servers. The increase overhead Increasing the number of servers used increases the number of the requests per server and this may potentially determine a decreasing in servers' performance [Hu98]. Furthermore, with the multiple servers approach we need intelligent clients that are able to manage parallel downloading from several servers.

In both cases (i.e., one, or more servers) we need to adopt a server(s) selection criterion. We can distinguish the criteria in three main classes[4]: ***Static***, ***History based***, and ***Dynamic***.

Static. A static criteria is based on information that are supposed not changing, e.g., the hardware resources and their configuration, the number of hops between servers and clients, the bandwidth of the links, etc. Example of static criteria used in servers' selection are: the selection based on the topological proximity (minimum number of hops) or the random selection. Topological proximity can be obtained by routing protocols. In the random selection each server has a fixed probability to be contacted, eventually based on IP address of the server list returned from DNS. When this probability is the same for all the servers in the DNS list, we obtain the same behavior of the system that is implemented by the Round-Robin DNS mechanism. Giving equal probability to each server is an easy technique to perform at the client, but can lead to poor performance ([Con99], [Con01], [Zeg00]).

A static criteria is simple but it has many drawbacks as it doesn't consider the dynamic feature of the Internet environment. They are based on the presence of the resources and not on their use, i.e. their availability. In addition, the servers' load and the network congestion are neglected. Hereafter we don't deal with static criteria.

***Hystory based*.** They use information that is collected off-line and is not evaluated at the moment in which the selection takes place. By using this approach there is the assumption that measured quantities don't change significantly and quickly in time, consequently their value can be used to predict future behavior. This assumption is hardly verified in Internet because there is high variability [Dyk00]. It is clear that history-based techniques are not deployable during the initial phase by the client, because they need to have a previously collected knowledge. These techniques fit better to be adopted into a proxy or more generally by a provider. Moreover, maintaining information, e.g. daily, for several replicated WSs connections, means to store an amount of data that can growth dramatically if we consider the number of replicated Web sites. This approach is generally impracticable for its overhead unless the client contacts only a limited subset of sites. History-based techniques seem to be suitable when client-based policy is implemented into a proxy. In this case we have to consider its adaptability to changes of the environment.

History-based techniques are based on periodic off-line information updates. The use of this technique leads to poor performance when the environment status changes with a speed that is the same or higher than the updating period as in this case stale data is used to choose server(s). Furthermore, in dynamic environments these techniques have a great overhead as to keep updated information they must generate a large amount of traffic.

Dynamic. This class inluces all criteria that make (active or passive[5]) measurements to assess the system status and to predict its behavior in the reference interval (i.e. download interval). Quantities in Internet are highly variable and then dynamic techniques are suitable for short reference intervals. On the other side these techniques may have a heavy overhead related to the technique adopted in the probing phase. Among the quantities that can be measured there are: network (highly variable) and server (they can be almost stable in time [Mye99]) performance figures.
An important problem that we have to take into account by adopting a dynamic criteria is related to the instability of servers' performance. It is possible that from the initial probing phase a server seems to be

[4] A similar classification can be found in [Dyk00].
[5] Passive techniques (i.e. tcpdump) don't perturb network state (i.e. the client "listens" to the network). Active tecniques inject stimula into the network, then they measure network answer to these stimula. Active techniques can perturb network state [Bar99]. Both techniques produce overhead at the client that uses them.

unloaded. Each client that contacts this server during the same RTT interval get the same view of this server (i.e. potentially a good server to download data) and select it as the best server, This choice may produce congestion on the selected server while other servers may became unloaded, and so on. A possible solution to reduce this problem is based on partitioning the servers in equivalence classes from the performance standpoint (e.g., a class contains all servers with a response time below a threshold value [Zeg00]). Servers belonging to a class are selected in a random way.

The simplest dynamic criteria is based on the selection of all the available servers. When the client chooses to use all the servers it doesn't use any selection criteria. In this case we do not have any limitation to the service responsiveness and availability, but we may introduce an excessive overhead in the client (because it maintains the connection and the state information related to each server), in the network, and in servers because their load increases A more promising approach, in particular if the number of replicated WSs is high, is to consider a subset of all servers (in the extreme cases we return to one server or to all servers). We have already mentioned that we can obtain almost optimal performance if we use a subset of all available servers [Mye99]. In this way the overhead in the client, network and servers is less than when we use all replicated WSs. We have to note that the server selection, both in term of server choice and servers number, belonging to this subset is very important because there is the possibility to not include the fastest servers.

iii) Downloading method

This phase is relevant when several servers are used during the downloading. It specifies how the selected servers contribute to the data downloading. Several methods are used:

1) ***Fixed-size data blocks*** [Rod00]. The data to download are divided into fixed-size blocks. Each served is queried for a block. The block size is fixed either a-priori or at the beginning of the download, and hence no run-time computation is need. One of the most important problem with this technique is that data integrity is lost. Moreover it is important to define the size of these blocks: small-size block means many small requests, while large-size block may reduce the efficiency of the method if there are low-speed servers. In addition there is the problem to manage many requests in pipelining.
2) ***Variable-size data blocks*** [Ghi01]. Each server is queried for a block whose size depends on the server performance. This method is highly sensitive to vatiations in servers performance. Furthermore, a great overhead can be produced to define block size. We have to consider also that variable size block can be substituted by more fixed size ones requested in pipelining, in which size is little and well evaluated.
3) ***Single entities.*** An entity corresponds to single files, images, etc. Hence, the entities have variable dimensions. The great advantage is that we have not data fragmentation and data can be managed better at application layer.[6] Problems may arise when we have big files (even if this event has low probability to be on the Web). In this case the client is obliged to remain connected to the server selected for that entity (even though it is slow) for the entire duration of the entity download. This problem can be easily solved by fragmenting the entities whose size is greater than a pre-defined threshold.
4) ***File portions encoded by erasure codes*** [Byers99]. In this case erasure codes are used (better known as Forward Error Correction codes). These codes ensure that a file of k packets generates n packet encoding of the same file with the property that any k packets are enough to restitute the initial file. Based on this encoding property the client is able to recover data coming from different servers in parallel once it receives a subset of the transmitted packets, say ***k*** out of ***n***. Obviously, there is an overhead due to encoding/decoding operations and ad-hoc clients and servers must be adopted. This approach is mainly suitable for a reliable multicast transmission.

iv) Download monitoring

This choice is related to the selection of the state information that has to be maintained during the data downloading. These information may include:

1) The status of each client-sever connection, e.g., throughput, bandwidth, latency, etc. This approach gives to the client the information to adapt its choices to variable network/server(s) conditions.
2) Memorization of partially acquired data. By doing so it is possible to avoid requests of data that the client has already received.
3) History of the client-sever connections. This information is oriented to the use of a static selection criteria and may be useful if the network and server conditions are stationary. As explained before a static criteria is meaningful if the server(s) selection is implemented into a provider or if the client connects to a limited subset of sites.

[6] This approach decrease the probability to have received a large amount of data that cannot be sent to the browser for rendering because they are fragments of a greater entity that has not yet completely received

In all cases we have to take into consideration that download monitoring introduces overhead in the client, in the network and server(s). From the overhead standpoint, the best approach seems to be the technique adopted by Rodriguez et al. [Rod00] that is explain in detail in Section 3.2. In this technique the number of state information is small because the technique automatically adapts to system changes (e.g., the performance of client-server connections) and no explicit monitoring is required [Rod00].

Finally, when only a server or a subset of servers is chosen, we have to define the role for the server(s) not involved into the download phase. Potentially, we can continuously monitor these server(s) to dynamically redefine to which server(s) the client requires data, or to choose a new server(s) when the performance of one server (among those previously selected) degrades. Obviously, this approach produces overheads in the network, client and server(s). Moreover we have to define what kind of monitoring to do and the monitoring period.

From the overhead standpoint the lightest solution is to neglect server(s) not involved into download phase until there is the need to consider them.

3.2 Analysis of existing solutions

As previously stated, all of available approaches can be collected in two main categories, depending on how many servers are used to download data:

- all available servers, i.e. all WSs;
- one server.

The approach of using all available WSs is based on the assumption that the more resources you use the more you can improve the QoS perceived by users. Moreover choosing all WSs is equivalent to avoid a difficult choice. The choice of the ***best*** server from which the client downloads data is complex and produces overhead. When all servers are used, the data to download is partitioned and each server is queried for a part. There are many different approaches that use all WSs ([Rod00], [Ghi01], [Bye99]); hereafter, we sketch the simplest (yet efficient) technique proposed by Rodriguez et al. [Rod00]. According to this technique the client downloads fixed size blocks of data from all the WSs simultaneously. The great advantage of the Rodriguez's technique is related to its simplicity: when a server is faster than the others, it automatically receives more blocks' requests. However, we have a great increase of complexity when we want to implement improvements such as: the exploitation of pipelining (an HTTP/1.1 feature), the decrease of the amount of idle times[7] and the increase of the size block requested to the ***best*** responsive servers. Among the others techniques that download data in parallel from all servers we can remember the closest conceptually to [Rod00]. This is explained in [Ghi01]; in this case the data block size is proportional to the measured throughput for every client-server connection. This approach can lead to URT improvements and it is adaptive to the link conditions.

An alternative approach that uses all the WSs in parallel is presented in [Bye99]; fundamentally, the main difference with the two previous techniques is that, in this case, data are encoded by using erasure codes (see point iii) in the previous section).

The only one server approach has many advantages with respect to the all-server approach. In many situations the client has to maintain the communication with only one server. In general this need increases when a client not only download readable data, but also transmits data to server in an interactive way, such as in business transactions, shopping carts and search engine requests. In these application scenarios, the client must to be connected with only one server. This requirement is not valid only when a client begins the communication but it is valid for the whole user's session. Moreover, changing server when a session is already begun has bad effects on the performance of the transfer [Kan00]. Another great advantage in binding a client to the same server for consecutive transactions during the same session is related to a better exploitation of cache memory and to improve server performances [Nor-d]. Binding for all the session the client to only one server that have good responsiveness is justified by taking into consideration that the server performance varies smoothly during the session [Mye99].

4 The MIL-based approach

Taking into account the pro and cons of the one-server and all-server solutions, we propose our client-side solution, named ***MinimumLatency-based*** (***MIL-based***), for the replication of a Web service. In our approach the client is bounded with few WS replicas. To reach a good level of QoS and fault tolerance (i.e., responsiveness and availability) a small subset of servers, say. 2 or 3 servers, seems enough. Further increasing the number of servers is more related to the availability issues then the responsiveness ones (as stated also in [Mye99]). Increasing the number of server used increases the number of the requests per server and this may potentially determine a decreasing in servers' performance [Hu98].

[7] Idle times are the interval during which the server has not yet received a request and is not sending data.

The MIL-based approach is based on the choices done in each phase of the analysis presented in Section 3.1. Specifically, the MIL-based approach uses an initial probing phase to evaluate the HTTP RTT by sending a HEAD request on a subset of all servers. This subset is defined in a random and dynamic way. Dynamic means that, if performance of the selected servers decrease, other servers have to be contacted. The dimension of this subset should not be greater than 3 WSs, but a performance evaluation study is currently ongoing to evaluate the subset size. One the subset is identified, the client downloads data by querying single entities from each server. However, if the size of an entity is greater than 20 KB, the file that represents that entity is divided into blocks of fixed length to be downloaded from different servers. Finally, the client downloads in parallel from the replicated WSs that are in the chosen subset. By adopting a monitoring phase, the subset of servers used in the downloading is dynamically modified to remove servers that become unavailable or whose performance degrade below a given threshold.

The main qualitative characteristics of our approach are:

- ***Scalability***
  It is easy to obtain; no changes at the client-side software are requested if we add other WSs.
- ***Orthogonal***
  Our approach is not in contrast with server-side techniques and the orthogonal requirement is satisfied.
- ***Availability***
  Who doesn't respond to the probing phase is temporarily unavailable. In this way by initial probing phase the client can avoid to attempt to use an unavailable replica.
- ***Overhead***
  In the probing phase we introduce some overhead in the client to run the program, in the network because we increase the traffic and into the server to answer to the probing requests. It is worth noting that in this probing phase can be also used a lightweight protocol, i.e. UDP instead of using a HEAD request.
- ***Available DNS feature activated***
  To deploy this technique it is necessary to configure the DNS to respond to an URL resolution request with the whole list of the IP addresses of the WSs.
- ***Low cost and easy to use***
  To exploit it, the client has only to download an applet or it can be integrated into the browser package.

A simulative study is currently underway to tune the parameters of the ***MIL-based*** approach, and to evaluate its characteristics from a quantitative standpoint.

References

[Arl00] M. Arlitt, T. Jin, ***A Workload Characterization Study of the 1998 World Cup Web Site***, Hewlett-Packard Laboratories, IEEE Network, May-June 2000.

[Bar99] Paul Bardford, Mark Crovella, ***Measuring Web Performance in the Wide Area***, Performance Evaluation Review, Special Issue on Network Traffic Measurement and Workload Characterization, August 1999.

[Bye99] J. W. Byers, M. Luby, M. Mitzenmancher***, Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads***, Proceedings of IEEE INFOCOM'99, New York, March 1999.

[Car96] R. L. Carter, M. E. Crovella, ***Dynamic Server Selection using Bandwidth Probing in Wide-Area Networks.***

[Car97] R. L. Carter, M. E. Crovella, ***Server Selection using Dynamic Path Characterization in Wide-Area Networks,*** In Proceedings of IEEE Infocom'97, April 97, Kobe, Japan.

[Che01] Cherkasova L; DeSouza M and Ponnekanti S***., Performance Analysis of Content-Aware Load Balancing Strategy FLEX: Two Case Studies,*** Thirty-Fourth Hawaii International Conference on System Sciences, HICSS-34 2001:Maui, Hawaii, January 3rd –6th, 2001.

[Cis97] Cisco Systems. ***Scaling the Internet Web Servers: A white Paper***, http://www.cisco.com/warp/public/751/lodir/scale_wp.htm , 1997.

[Cis99] http://www.cisco.com/warp/public/cc/pd/cxsr/dd/tech/dd_wp.htm

[Con99] M. Conti, E. Gregori, F. Panzieri, ***Load Distribution among Replicated Web Servers: A QoS-based Approach***, Proc. Second ACM Workshop on Internet Server Performance (WISP'99), Atlanta, Georgia, May 1, 1999.

[Con01] M. Conti, E. Gregori, F. Panzieri, ***QoS-based Architecture for Geographically Replicated Web Servers***, Cluster Computing Journal, 4, 2001, pp. 105-116.

[Con01a] M. Conti, E. Gregori, W. Lapenna, "Replication Web Servers: Comparison of Client-based and Server-based solutions", Computer Measurement Group Conference, CMG-Italia, 4-6 giugno, 2001, Rome, Italy.

[Dyk00] S. D. Dykes, K. A. Robbins, C. L. Jeffery, ***An Empirical Evaluation of Client-side Server Selection Algorithms***, IEEE INFOCOM, VOL. 3, March, 2000, pp. 1361-1370.

[Ghi01] V. Ghini, F. Panzieri, M. Roccetti, ***Client-centered Load Distribution: A Mechanism for Constructing Responsive Web Sevices***, presented at the 34th Hawaii International Conference on System Sciences (HICSS-34), Maui (Hawaii),3-6 January 2001.

[Fie97] R. Fielding et al., ***RFC 2068 - Hypertext Transfer Protocol - HTTP/1.1***, http://www.ietf.org/rfc/rfc2068.txt

[HP-a] http://ovweb1.external.hp.com/nnminteract/dld_whitepaper.htm

[Hu98] J. Hu, S. Mungee, and D. C. Schmidt, ***Principles for Developing and Measuring High-performance Web Servers over ATM***, in Proceedings of INFOCOM '98, March/April 1998.

[Kan00] J. Kangasharju, J. W. Ross, J. W. Roberts***, Performance Evaluation of Redirection Schemes in Content Distribution Networks***, In Proceedings of 5th International Web Caching and Content Delivery Workshop, Lisbon, Portugal, 22-24 May 2000.

[Mye99] A. Myers, P. Dinda, H. Zhang, ***Performance Characteristics of Mirror Servers on the Internet***, IEEE Infocom 1999, NY, March 1999.

[Nor-a] http://www.alteonwebsystems.com/collateral/PCD_eng.pdf

[Nor-b] http://www.alteonwebsystems.com/collateral/acedirector.pdf

[Nor-c] http://www.alteonwebsystems.com/collateral/GSLB_wp.pdf

[Nor-d] http://www.alteonwebsystems.com/collateral/vma_white_paper.pdf

[Rod00] P. Rodriguez, A. Kirpal, E.W. Rod00, ***Parallel-Access for Mirror Sites in the Internet***, Proceedings of IEEE/Infocom 2000, Tel-Aviv, Israel, March 2000.

[Say98] M. Sayal, Y. Breitbart, P. Scheuermann, R. Vingralek, ***Selection Algorithms for Replicated Web Servers***, Workshop on Internet Server Performance, SIGMETRICS, Madison, USA, June 1998.

[Say99] M. Sayal, Y. Breitbart, P. Scheuermann, R. Vingralek***, Web++: A System For Fast and Reliable Web Service***, Proceedings of 1999 USENIX Annual Technical Conference, June 6-11, 1999, Monterey Conference Center, Monterey, CA, USA.

[Sch00] T. Schroeder, S. Goddard, B. Ramamurthy***, Scalable Web Server Clustering Technologies, IEEE Network***, May/June 2000.

[Sha01] A. Shaikh, R. Tewari, and M. Agrawal, ***On the Effectiveness of DNS-based Server Selection***, ***Proc. IEEE INFOCOM 2001***, April 2001.

[Ste96] W. Richard Stevens, TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesly , 1996.

[Zeg00] E. W. Zegura, M. H. Ammar, Z. Fei, S. Bhattacharjee, ***Application-Layer Anycasting: A Server Selection Architecture and Use in a Replicated Web Service***, IEEE/ACM Transactions on Networking, Vol. 8, NO. 4, August 2000.