# Client-centered Load Distribution: A Mechanism for Constructing Responsive Web Services

Vittorio Ghini, Fabio Panzieri, Marco Roccetti
*Dipartimento di Scienze dell'Informazione, Università di Bologna*
*Mura Anteo Zamboni, 7, 40127 Bologna (Italy)*
*{ghini, panzieri, roccetti}@cs.unibo.it*

## Abstract

*In this paper we describe the design, implementation and experimental evaluation of a software mechanism that supports responsive (i.e. highly available and timely) Web services, constructed out of replicated servers. Specifically, this mechanism operates by engaging all the available replicas in supplying a fragment of the Web document that a client requires. The size of the fragment a replica is requested to supply is dynamically evaluated on the basis of the response time that replica can provide its client with. In addition, the proposed mechanism can dynamically adapt to changes in both the network and the replica servers' status, thus tolerating possible replica or communication failures that may occur at run-time. The performance results we have obtained from our experimental evaluation illustrate the adequacy of the mechanism we propose.*

## 1. Introduction

Responsiveness, i.e. high availability and timeliness, is a crucial issue in the design of a Web service, as, from the service user perspective, a poorly responsive service can be virtually equivalent to an unavailable service.

A host of techniques, e.g. [1,2,3,6,8,9,10,11,12,14], has been proposed in the literature that addresses specifically the issue of providing highly available Web services. In general, these techniques are based on i) constructing a Web service out of replicated servers, locally distributed in a cluster of workstations, and ii) distributing the client request load among those servers. Thus, service availability is achieved through the redundancy inherent in the service implementation; in addition, the overall service throughput (i.e., the number of client requests per second that can be served) is optimized through careful distribution of the client request load among the clustered servers.

As discussed at length in [7], these techniques can only partially meet the high availability requirement mentioned above (e.g., they may be vulnerable to failures of the router/gateway that interfaces the service cluster to the rest of the network). In addition, these techniques cannot be deployed in order to meet timeliness requirements such as the client latency time over the network (that may notably affect the timeliness of a Web service) as these requirements are beyond the control of these techniques.

In order to overcome these limitations, an alternative approach has been proposed in [7], and explored further in [4,5]. According to this approach, a responsive Web service can be provided by replicating servers across the Internet (rather than in a cluster of workstations).

In the Internet context, a successful deployment of this approach will depend on the ability of achieving the following two principal goals: i) dynamically binding the client to the *most convenient* replica server, and ii) maintaining data consistency among the replica servers.

Unfortunately, neither of these two goals is easy to achieve. Firstly, the dynamic binding of clients to replica servers can turn out to be difficult to implement, owing to the location based naming scheme used in the Web. This scheme provides a one-to-one mapping (i.e., the Uniform Resource Locator - URL) between a name of a resource and a single physical copy of that resource; hence, dynamic binding of a client to distinct replica servers requires that the client-side software be adequately extended in order to be able to select an available replica, at run-time. Secondly, maintaining replica consistency on a large geographical scale can be hard to achieve, without affecting the overall service performance. In addition, the Internet environment is subject to (real or virtual) partitions, that can prevent communications between functioning nodes; hence, within this environment, both clients and replica servers may hold mutually inconsistent views of which replica server is available and which is unavailable.

Owing to these observations, we have developed a mechanism for constructing responsive web services that is implemented as an extension of the client-side software. Specifically, our mechanism provides the clients of a replicated Web service, constructed out of replica servers distributed over the Internet, with timely responses (issues of replica consistency fall outside the scope of this mechanism).

The principal goal of our mechanism is to minimize what we term the User Response Time (URT), i.e. the time elapsed between the generation of a browser request for the retrieval of a Web page, and the rendering of that page at the browser site.

In summary, rather than binding a client to its *most convenient* replica server, as proposed in [4,5,7], our mechanism intercepts each client browser request for a Web page, and fragments that request into a number of sub-requests for separate parts of that document. Each sub-request is issued to a different available replica server, concurrently. The replies received from the

replica servers are reassembled at the client end to reconstruct the requested page, and then delivered to the client browser.
Our mechanism is designed so as to adapt dynamically to state changes in both the network (e.g. route congestion, link failures), and the replica servers (e.g. replica overload, unavailability). To this end, our mechanism monitors periodically the available replica servers and selects, at run-time, those replicas to which the sub-requests can be sent, i.e. those replicas that can provide the requested page fragments within a time interval that allows our mechanism to minimize the URT. As our mechanism implements effectively load distribution of client requests among the available replicas, we have named it Client-Centered Load Distribution ($C^2$LD).
The design of $C^2$LD is based on an analytical model that we have developed. This model is essential in order to determine the size of the page fragment that can be requested to each replica, and the extent of the replica monitoring period $C^2$LD is to use in order to execute a client request. We have validated this model by implementing it over the Internet, using four replica servers located in Italy, in the UK, and in the USA.
In this paper we describe the $C^2$LD design, implementation and validation we have carried out. Specifically, in the following Section the $C^2$LD analytical model is introduced. Section 3 describes the $C^2$LD implementation we have developed. Section 4 discusses the experimental results we have obtained from that implementation; finally, Section 5 provides some concluding remarks.

## 2. Analytical model

The timeliness requirement that our $C^2$LD mechanism is to meet can be expressed by means of a User Specified Deadline (USD), i.e. a value that indicates the extent of time a user is willing to wait for a requested Web page to be rendered at his/her workstation.
It is worth observing that, firstly, as the USD value is user specified, it may differ from the actual response time a browser request may experience over the Internet (i.e. the URT previously introduced), at least in principle; thus, if a user sets an unrealistic USD, the $C^2$LD returns an appropriate exception. Secondly, note that, in order to introduce no modifications in the software of commercial browsers that can make use of $C^2$LD, in our implementation the USD is set by the user prior to the invocation of the browser, and then captured by the $C^2$LD mechanism.
Owing to this USD time constraint, each replica server that receives a sub-request for a page fragment must honor that sub-request within a time interval that allow the $C^2$LD mechanism to reconstruct the requested page, out of all the received fragments, before the USD deadline expires. Thus, it is crucial that the $C^2$LD mechanism assess accurately both the size of the fragment each replica is to supply, and the time intervals within which these fragments are to be received at the client site.
To this end, we have developed the analytical model summarized below, and discussed in detail in [15].

### 2.1. Model

Assume that exactly $N_{REP}$ replica servers be available, and that the size of the requested Web page be known, and be equal to *DS* bytes. Moreover, assume that the total amount of time needed to download an entire page is exactly equal to the sum of $N_{INT}$ subsequent time intervals, each of which has a duration of *S* seconds (i.e. $URT = N_{INT} \cdot S$). Finally, let us denote with *DR* the data rate that a given replica server *i* is able to provide during a given time interval *k*, $k \in \{1,.., N_{INT}\}$, and with *PS* the size of the document fragment requested to a certain replica server *i*. Note that the data rate *DR* may vary unpredictably in different time intervals. In addition, a sub-request, submitted to a certain replica server at the beginning of a given time interval *k*, may terminate during some later interval *h* (i.e., $h \geq k$). Hence, a realistic model must be able to represent both the time interval in which each sub-request is transmitted to each replica, and the exact sequence of all the sub-requests transmitted to each replica. Based on these assumptions, the analytical model we have developed is as follows.
We denote with the index *r*, ranging in the interval $\{1,.., N_{REQ_i}\}$, each single sub-request that can be issued to the replica *i*. Using the replica index *i* and the sub-request index *r*, we can define the following mapping $k_{i,r}$:

$$k_{i,r}(T) = (T \; div \; S) + 1, \tag{1}$$

where *T* denotes the exact time instant in which a given sub-request *r* is issued to the replica *i*, and $(T \; div \; S)+1$ denotes the time interval containing *T*.
In order to assess the data rate that will be provided by a given replica *i*, we adopt the following measurement-based strategy. Assume that the sub-request *r* must be issued to the replica *i* at the time *T*, in the interval $k_{i,r}$; then, the data rate for that sub-request can be estimated as:

$$DR_{i,r} = \frac{PS_{i,r-1}}{URT_{i,r-1}}. \tag{2}$$

In the above formula, $PS_{i,r-1}$ represents the size of the document fragment downloaded with the previous sub-request *r-1*, terminated by the time *T*. $URT_{i,r-1}$ is the elapsed time experienced for downloading the *r-1* document fragment of size $PS_{i,r-1}$. Note that the $URT_{i,r-1}$ value may be experimentally measured at the time *T*, when the previous sub-request *r-1* has been completed. Given that value, the document fragment to be requested to the replica *i* is:

$$PS_{i,r} = DR_{i,r} \cdot S_{i,r}^{*} \tag{3}$$

with

$$S_{i,r}^{*} = k_{i,r} \cdot S - T \tag{4}$$

and

$$(k_{i,r} - 1) \cdot S \le T < k_{i,r} \cdot S\,. \tag{5}$$

In Eq. (3) above, the term $S_{i,r}^{*}$ represents the URT expected from the execution of the sub-request *r* directed to the replica *i*; in Eq. (4), for the sake of simplicity, the term $k_{i,r}$ has been used in place of $k_{i,r}(T)$. Depending on the value of *T*, the following two possible events may occur:

1. the sub-request *r-1* terminates exactly at the beginning of the interval $k_{i,r}$, i.e. $T = (k_{i,r} - 1) \cdot S$ ;
2. the sub-request *r-1* terminates during the $k_{i,r}$ interval, i.e. $(k_{i,r} - 1) \cdot S < T < k_{i,r} \cdot S$ .

If event 1 occurs, the size of the requested page fragment is proportional to the total duration *S* of a complete interval. Instead, if event 2 occurs, the size of the requested fragment is proportional to the residual time $k_{i,r} \cdot S - T$ needed to reach the end of the $k_{i,r}$ interval. Finally, the following requirement must be met by all the replica servers, in order to ensure that a requested page be entirely downloaded within the USD deadline:

$$\frac{DS}{\sum_{i=1}^{N_{REP}} \frac{\sum_{r=1}^{N_{REQ_i}} DR_{i,r} \cdot URT_{i,r}}{N_{INT} \cdot S}} \le USD\,. \tag{6}$$

Eq. (6) states that the sum of the average data rates provided by all the replica servers during the downloading period $N_{INT} \cdot S$ must be such that the requested page (with size *DS*) is completely downloaded before the *USD* deadline expire. To conclude this Subsection, we wish to point out that our model provides a measurement-based strategy for assessing the actual data rate each replica can provide, as the size of a fragment to be requested in a sub-request *r* depends upon the measurement of the $URT_{i,r-1}$ value experienced at the time the (previous) sub-request *r-1* terminates.

## 3. Implementation

We have implemented our analytical model on top of the HTTP 1.1 interface, as suggested in [5], using the Java programming language. Our implementation supports the standard HTTP 1.1 interface; so as to operate transparently to the higher software layers (e.g. the browser software). For the purposes of this implementation, we have provided the users of our mechanism with a C$^2$LD configuration procedure that allows them to set the USD timeout, introduced earlier, before they request access to a Web service (a default USD value is used by our C$^2$LD implementation, if a user does not make use of that configuration procedure).

Typically, in order to access a Web service, a user starts a browser by providing it with the URL of that service. That browser invokes an HTTP GET method with that URL. The C$^2$LD mechanism intercepts that HTTP GET invocation, starts the USD timeout and, using the URL in the GET invocation, interrogates the DNS.

The DNS maintains the IP addresses of the $N_{\text{REP}}$ replica servers that implement a Web service. When C$^2$LD submits a request to the DNS for resolving a URL, the DNS returns the IP addresses of all the replica servers associated to that URL. As the replica servers addresses are available to the C$^2$LD mechanism, this mechanism interacts with each replica as illustrated in Figure 1, and summarized below.

C$^2$LD invokes an HTTP HEAD method on each replica *i*. The reply from replica *i* to the first HTTP HEAD invocation is used by C$^2$LD to: i) get the size of the requested page, ii) estimate the data rate that replica *i* can provide, and iii) assess the size of the first fragment that can be fetched from that replica.

C$^2$LD maintains a global variable (*download_done*, in Fig.1) that indicates whether or not all the fragments of a requested page have been delivered. Until a page is not fully downloaded, C$^2$LD uses the Eq. (2) and (3) in Section 2 to compute the size of the fragment that is to be requested to the replica *i*.

Once the requested fragment size has been calculated, C$^2$LD issues an HTTP GET request to the replica *i*, in order to retrieve the required fragment of that size. (Specifically, a fragment of Z bytes size is requested by invoking an HTTP GET method with the following option set: "*Range: bytes=Y-X*", where X and Y denote the bytes corresponding to the beginning and the end of the requested fragment of size Z, respectively.)

In order to adjust adaptively to possible fluctuations of the communication delays that may occur over the Internet, the fragment size is computed each time a fragment is to be requested, based on the value of the URT experienced in fetching the previous fragment. Thus, in essence, as the GET request *r-1* directed to a given replica *i* terminates, a new GET request *r* can be issued to the replica *i* with the fragment size value $PS_{i,r}$ computed on the basis of the response time $URT_{i,r-1}$.

```
/* C^2LD */
…
within USD do                          /* set USD timeout */
…
HEAD(…)                                /* send HEAD request to replica i */
URT(i) := …                            /* assess URT replica i can provide */
PS(i,1) := …                           /* compute 1st fragment size for replica i */
within S do                            /* set timeout of length S */
        if not download_done then      /* check if page download completed */
                GET (…)                /* get fragment from replica i */

                $DR_{i,r} = \frac{PS_{i,r-1}}{URT_{i,r-1}}$ ;  /* compute expected data rate, based on URT of previous request */
                $PS_{i,r} = DR_{i,r} \cdot S_{i,r}$ ; /* compute size of next fragment to be requested */
        else
                return;

od
        …
od
```

**Figure 1: Implementation of the C$^2$LD Service**

Note that some of the replica servers may not respond timely to the HTTP (HEAD and GET) invocations described above (e.g., they may be unavailable owing to network congestion). Thus, C$^2$LD associates a timeout to each HTTP request it issues to each server *i*. If that timeout expires before C$^2$LD receive a reply from a replica server *i*, it assumes that the server *i* is currently unavailable, and places it in a *stand_by* list. Replica servers in that list are periodically probed to assess whether they have become active again. Requests for replicas in the *stand_by* list are redirected to active replicas.
Finally, we wish to mention that, in order to increase the degree of parallelism in the fetching of document fragments, the C$^2$LD mechanism has been implemented using the thread programming model provided by the Java 2 Software Development Kit. A detailed discussion of our implementation of this mechanism can be found in [15].

## 4. Measurements

The effectiveness of our C$^2$LD implementation has been validated through a large number of experiments (4000, approximately). These experiments were carried out during a two-month period; namely, November and December 1999. These experiments consisted essentially of a client program (i.e. a browser) downloading Web documents of different size from up to 4 geographically distributed replica servers. These documents were downloaded by the same client program using both the C$^2$LD mechanism, and the standard HTTP GET downloading mechanism, for comparison purposes.
In this Section, we describe in detail the scenario within which these experiments have been carried out, introduce the metrics we have used to assess our implementation, and discuss the performance results we have obtained.

### 4.1. Scenario

The performance of our C$^2$LD implementation has been evaluated using the Internet to connect a client workstation (equipped with our C$^2$LD software) with four replica servers.
The client workstation was a SPARCstation 5 running the SunOS 5.5.1 operating system and the Sun Java Virtual Machine 1.2. This workstation was located at the Computer Science Department of the University of Bologna. The four different replica servers were running the Apache Web server over the Linux platform.
For the purposes of our evaluation, these servers were located in four distinct geographical areas. Specifically, a replica server was located close to the client workstation; namely, at the Computer Science Laboratory of the University of Bologna, in Cesena. This Laboratory is four network hops far from our Department in Bologna. The connection between our Department and this Laboratory has a limited bandwidth of 2 Mbps, and is characterized by a rather high packet loss rate (between 2% and 9%).
A second replica server was located in northern Italy; namely, at the International Center of Theoretical Physics in Trieste. This server was reachable through 9 network hops via a connection whose bandwidth ranged between 8 and 155 Mbps.
A third replica server was located at the Department of Computing Science of the University of Newcastle

upon Tyne (UK). This server was reachable through 15 network hops from our client workstation.
Finally, a fourth replica server was located at the Computer Science Department of the University of California at S. Diego. This server was reachable through a 19 network hops transatlantic connection. Figure 2, obtained with the utility [13], depicts the locations of both the client and the replica servers used in our experiments, and the routes between this client and those servers; this Figure shows that the different routes connecting our client with the four replica servers scarcely overlap (i.e., route overlapping occurs only up to Milan, when communicating with the replica servers in Trieste, Newcastle, and San Diego). Within this scenario, a replicated Web service can be configured so as to use one of the following 11 combinations of the four available replica servers. Namely, a service can be replicated across the two servers in Cesena and Newcastle (C+N, in the following), only, or those in Cesena and Trieste (C+T), or in Trieste and Newcastle (T+N), or in Cesena and S. Diego (C+S), or in Trieste and S. Diego (T+S), or Newcastle and S. Diego (N+S).
A more redundant service can be implemented across one of the following four combinations of three servers, instead: Cesena, Newcastle and Trieste (C+N+T); Cesena, Newcastle, and S. Diego (C+N+S); Cesena, Trieste, and S. Diego (C+T+S); Trieste, Newcastle, and S. Diego (T+N+S). Finally, a redundant Web service can be implemented across the four replica servers in Cesena, Newcastle, Trieste, and S. Diego (C+N+T+S). Our experiments have been carried out using all these 11 combinations of replica servers. It is worth noting that the four machines running the server replicas were moderately loaded during our experiments. In contrast, the routes to the European Web replica servers were heavily loaded during daytime; network traffic over these routes typically decreased during the night.
The average network traffic conditions on both the European and transatlantic routes, experienced during the evaluation of our $C^2$LD mechanism, are reported in the following Table 1. Specifically, in this Table, the first row indicates the 90% percentile of the Round Trip Time (RTT) obtained with the ping routine from our client workstation in Bologna to the four replica servers; the second row reports the minimum RTT experienced over the routes to those servers; the third row reports the average packet loss rate we measured over those routes, as obtained with the ping routine (ICMP).
Finally, note that the network traffic almost saturated the bandwidth of 8 Mbps, available at the University of Bologna routers, during working hours [15].

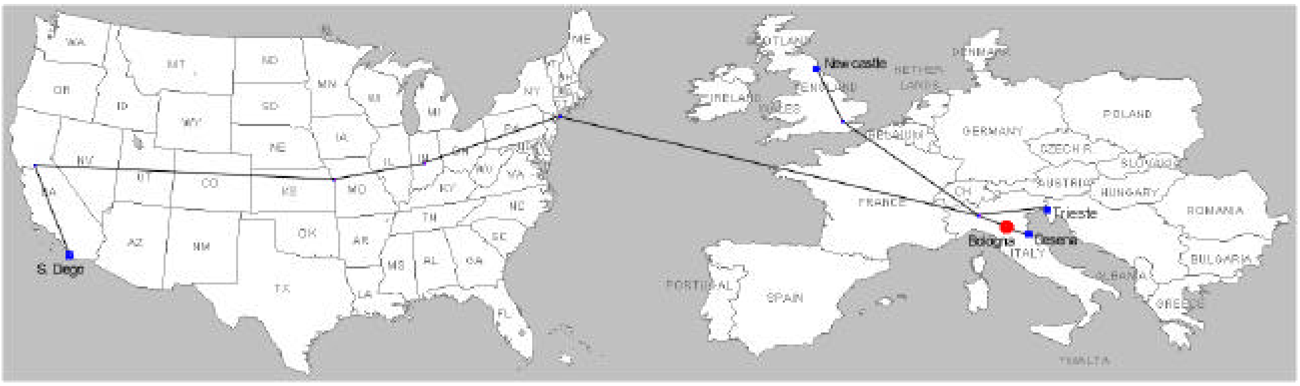

**Figure 2. Routes between the client and the Web replica servers**

| ping from Bologna to: | Cesena | Trieste | Newcastle | S. Diego |
|---|---|---|---|---|
| RTT 90% of arrived pkt (msec) | 107 | 95 | 160 | 450 |
| RTT min (msec) | 10 | 38 | 59 | 190 |
| Lost Packet | 2%-9% | 0% | 0% | 0%-3% |

**Table 1. The measured round trip time and lost packet percentage**

## 4.2. Measures

The following two metrics have been used to evaluate the effectiveness of our $C^2LD$ mechanism: i) the percentage of document retrieval requests successfully satisfied by our mechanism, and ii) the URT (as defined earlier) our mechanism provides. Both these metrics capture the principal user requirements; namely, that a requested document be effectively retrieved, and that it be done timely. Based on these metrics, the evaluation of our mechanism has been carried out using different values of the following four parameters: i) the number of server replicas that implement a given Web service, ii) the data rate that each different server replica can provide, iii) the download monitoring period adopted by the $C^2LD$ mechanism, and, finally, iv) the size of the document to be retrieved.

The performance of our $C^2LD$ mechanism has been compared and contrasted with that provided by the standard HTTP document retrieval mechanism, as this is implemented by the HTTP GET function. To this end, each replica server maintained a set of downloadable files of different size, ranging from 3 Kbytes to 1 Mbytes. These servers were requested to retrieve the same Web document, at the same time of the day (i.e. under the same network traffic conditions, approximately) using, alternatively, both the standard HTTP GET request, and our mechanism. The download monitoring period *S* used by our mechanism ranged from 50 milliseconds to 10 seconds.

A number of experiments were carried out for each given file size and download monitoring period *S*. Each experiment consisted of 15 consecutives download requests. 4 out of 15 of these requests were executed by invoking the HTTP GET function; the other 11 requests were executed by the $C^2LD$ mechanism. The experiments used files whose size was 3, 10, 30, 50, 100, 200, 500 and 1000 Kbytes. For each file size, the experiments with the $C^2LD$ mechanism were repeated using a different download monitoring period, ranging from 50 milliseconds to 10 seconds.

As mentioned earlier, our experiments were carried out on the 11 possible configurations of a replicated service, introduced in Subsection 4.1; however, the values reported in the Tables and Figures in this Subsection are the average of the results obtained in all our experiments.

As a first result, Table 2 summarizes the page loss percentage obtained with our mechanism, and that obtained with the standard HTTP GET downloading mechanism (performed with S. Diego, Newcastle, Trieste and Cesena, respectively). It can be seen from this Table that the $C^2LD$ mechanism provides a highly available service, since it always guaranties the downloading of the requested document. Instead, the standard HTTP mechanism is not always able to provide its client with that document, as shown by the page loss percentage experienced by the S. Diego and Cesena servers, in particular.

Figure 5 summarizes the results of the URT assessment we have obtained. The lowest curve in this Figure represents the URT provided by our $C^2LD$ mechanism, as it results from averaging its value over all the performed experiments. The two higher curves, instead, represent the URT values provided by the two fastest Web replica servers, when interrogated with the standard HTTP downloading mechanism. Specifically, the URT values provided by these two replica servers were obtained as follows. The four different replica servers were exercised with the standard HTTP mechanism; then, out of these four servers, the URT results obtained by the two fastest ones were selected to plot the graph in Figure 5.

| | $C^2LD$ | S. Diego (HTTP) | Newcastle (HTTP) | Trieste (HTTP) | Cesena (HTTP) |
|---|---|---|---|---|---|
| 50 Kbytes | 0 % | 0.3 % | 0 % | 0 % | 1.65 % |
| 100 Kbytes | 0 % | 0.25 % | 0 % | 0 % | 5 % |
| 200 Kbytes | 0 % | 0.2 % | 0 % | 0 % | 4 % |
| 500 Kbytes | 0 % | 0.3 % | 0 % | 0 % | 1 % |
| 1 Mbyte | 0 % | 0.45 % | 0 % | 0 % | 2.5 % |

**Table 2: Fault Percentage**

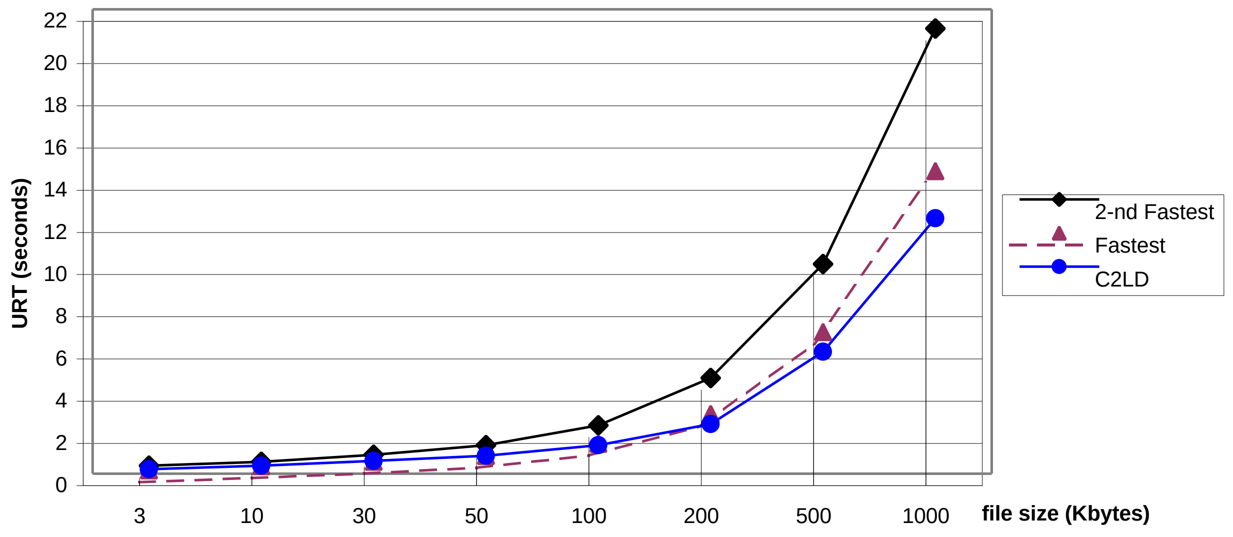

**Figure 5. C$^2$LD vs. HTTP**

As shown in that Figure, the performance of the C$^2$LD mechanism and the HTTP mechanism are equivalent when the document size less than 50 Kbytes. Instead, when the document size is larger than 50 Kbytes, the C$^2$LD mechanism outperforms the standard HTTP downloading mechanism. As already mentioned, the C$^2$LD URT values in Figure 5 represent an average URT value, as it results from all the experiments we have carried out. However, it may be interesting to assess the URT improvement that can be obtained by our mechanism as the number of replica servers, concurrently used, varies. To this end, we have carried out experiments in which the number of active replicas was varying from 1 to 4.

Figure 6 illustrates the results of our experiments, in these four cases. Specifically, in these cases, the URT is measured as a function of the file size. As shown in this Figure, the larger the number of replicas that are used, the better the URT values that can be obtained; in addition, as the file size increases, the advantage of using the C$^2$LD mechanism becomes more notable.

In addition, Table 3 shows the average percentage improvement of the URT values that has been experienced in the experiments depicted in Figure 6, as the number of replica servers grows.

We have experimented our C$^2$LD mechanism using different values of the download monitoring period parameter *S*, in order to assess the influence of that parameter on the URT provided by our mechanism. Specifically, we have measured the C$^2$LD URT using values of the monitoring period *S* ranging from 50 ms to 10 s. Figure 7 below illustrates the results our experiments. This Figure reports the different URT curves that were obtained using files of different size. Note that the monitoring period that provides the lowest URT values is 0.5 seconds, regardless of the file size.

Finally, for the sake of completeness, we report below two additional graphs. These graphs illustrate the performance results obtained by both requesting each replica to fetch a 500 Kbytes file, using a standard HTTP GET request, and exercising our mechanism with the fetching of the same file; the monitoring period used by our mechanism in these experiments ranged from 500 to 2000 milliseconds. Specifically, the histograms in Figure 8 represent the URT values obtained by the C$^2$LD mechanism (denoted as *parallel* in this Figure), and the URT values obtained by each single replica. Each histogram illustrating the performance of our mechanism relates to a different combination of the server replicas used during the experiments, as indicated in the Figure. Thus, for example, the leftmost *parallel* histogram reports the URT values obtained with our mechanism when only the replica servers in Cesena and Newcastle were used; the rightmost *parallel* histogram reports the URT values obtained with the C$^2$LD mechanism when all the four replica servers were used. The remaining histograms (other than the *parallel* histogram) relate to the individual replica servers, as indicated in the Figures.

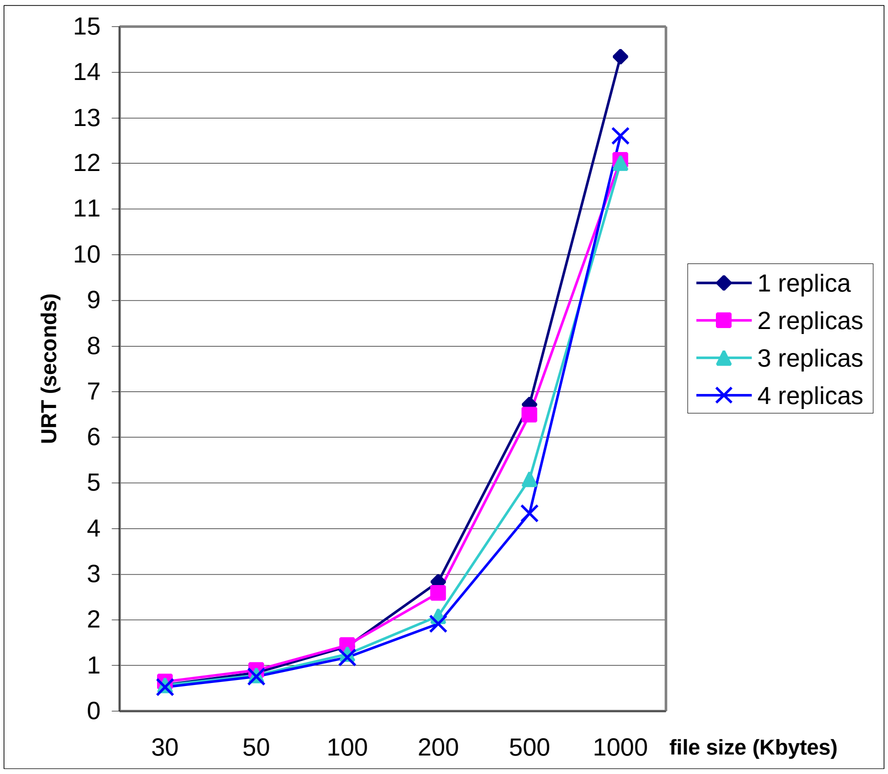

**Figure 6. URT improvement provided by the $C^2LD$ mechanism as the number of replicas grows**

| Number of Replica Servers | URT percentage improvement |
|---|---|
| 2 | 4% |
| 3 | 17.2% |
| 4 | 21.5% |

**Table 3. Average URT percentage improvement**

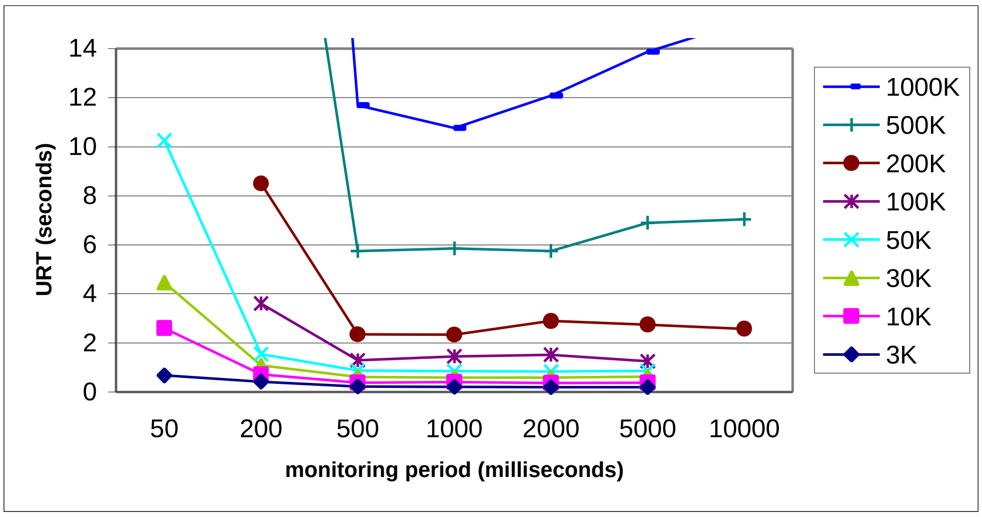

**Figure 7. URT depending on the monitoring period**

Figure 9 compares the percentage of URT improvement obtained by the $C^2$LD mechanism with that obtained by each single replica (interrogated using HTTP GET). Both Figure 8 and 9 show that, in general, the $C^2$LD mechanism outperforms each single replica in all cases; the only exception occurs when the $C^2$LD uses only two replica servers, and, out of these two servers, one is very fast (Trieste or Newcastle), and the other is very slow (i.e. San Diego).

## 5. Concluding remarks

In this paper, we have shown that the $C^2$LD mechanism we have developed in order to implement responsive Web services can outperform the standard Web page downloading mechanism (e.g. that provided by the HTTP implementation) as it can improve the URT by an average factor ranging between 4% and 21%, depending on the size of the downloaded page as indicated in Table 3 above. We have applied our $C^2$LD mechanism to the fetching of generic Web resources, such as files and documents; we wish to assess the adequacy of our mechanism when deployed for accessing digital video and audio resources. In addition, we are planning to extend the work described in this paper by addressing the following topics. Firstly, we intend to evaluate the overhead caused by our mechanism both at the network and at the server levels. Specifically, we wish to assess the effectiveness of our mechanism when multiple clients concurrently access the same replicated Web service. Secondly, we intend to evaluate the performance of our mechanism when implemented within a proxy server; in this context, we wish to explore the use of caching and prefetching techniques. Thirdly, we wish to compare and contrast the performance of our mechanism with that provided by other replicated services, such as those based on a locally distributed cluster of workstations. Fourthly, we wish to examine strategies that optimize the routing of requests to replica servers. Finally, we wish to investigate policies for maintaining data consistency among geographically distributed replica servers.

## Acknowledgments

This work has been partially funded by the Department of Computer Science of the University of Bologna, the Italian MURST and Microsoft Research Europe. A special debt of gratitude is due to our colleagues at the Computing Science Department of the University of Newcastle upon Tyne (UK), at the International Center for Theoretical Physics of Trieste (Italy), and at the Department of Computer Science of the University of California at San Diego (USA) for providing us with the resources required to carry out our experiments. Last, but by no means least, we wish to thank the anonymous referees who reviewed this paper for their valuable and constructive comments.

## References

[1] R.B. Bunt, D.L. Eager, G.M. Oster, C.L. Williamson, “Achieving Load Balance and Effective Caching in Clustered Web Servers”, Proc. 4th International Web Caching Workshop, San Diego, CA, March 1999.

[2] “Cisco Local Director”, CISCO System Inc., White paper, 1996.

[3] M. Colajanni, P. S. Yu, D. M. Dias, “Analysis of Task Assignment Policies in Scalable Distributed Web-Server Systems”, IEEE Trans. on Parallel and Distributed Systems, Vol. 9, N 6, pp. 585-600, June 1998.

[4] M. Conti, E. Gregori, F. Panzieri, “Load Distribution among Replicated Web Servers: A QoS-based approach”, Proc. 2nd ACM Workshop on Internet Server Performance (WISP'99), Atlanta, GA, , May 1999.

[5] M. Conti, E. Gregori, F. Panzieri, “QoS-based Architectures for Geographically Replicated Web Servers”, Cluster Computing (to appear).

[6] P. Damani, P.E. Chung, Y.Huang, C.Kintala, Y. M. Wang, “ONE-IP: Techniques for Hosting a Service on a Cluster of Machines”, Comp. Net. and ISDN Sys., 29, 1997, pp. 1019-1027.

[7] D. Ingham, S.K. Shrivastava, F. Panzieri, “Constructing Dependable Web Services”, IEEE Internet Computing, Vol. 4, N. 1, January/February 2000, pp. 25 - 33.

[8] A. Iyengar, J. Challenger, D. Dias, P. Dantzig, “High-Performance Web Site Design Techniques”, IEEE Internet Computing, Vol. 4., N. 2, March/April 2000, pp. 17-26.

[9] J. Li, H. Kameda, “Load Balancing Problems for Multiclass Jobs in Distributed/Parallel Computer Systems”, IEEE Trans. on Computers, Vol. 47, No. 3, March 1998, pp. 322-332.

[10] E.D. Katz, M. Butler, R. McGrath, “A Scalable HTTP Server: The NCSA Prototype”, Comp. Net. and ISDN Sys., 27 (2), pp.155-164, November 1994.

[11] J. Li, H. Kameda, “Load Balancing Problems for Multiclass Jobs in Distributed/Parallel Computer Systems”, IEEE Trans. on Computers, Vol. 47, No. 3, pp. 322-332, March 1998.

[12] V. Pai, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. Nahum, “Locality-aware Request Distribution in Cluster-based Network Servers”, Proc. *ASPLOS- VIII*, San Jose, CA, October 1998.

[13] Datametrics Systems Corp., “VisualRoute - Mapping the Internet”, http://www.visualroute.com

[14] J. Watts, S. Taylor, “A Practical Approach to Dynamic Load Balancing”, IEEE Trans. on Parallel and Distributed Systems, Vol. 9, NO. 3, March 1998, pp. 235-248.

[15] V. Ghini, F. Panzieri, M. Roccetti "Client-centered Load Distribution: A Mechanism for Constructing Responsive Web Services" Technical Report No. UBLCS-07, Laboratory for Computer Science, University of Bologna, June 2000.

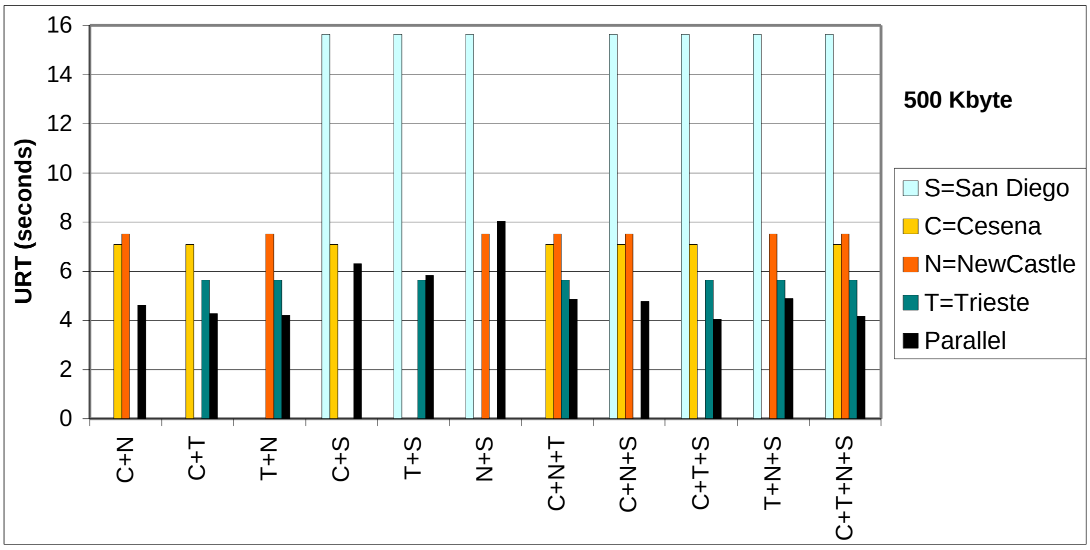

**Figure 8. C$^2$LD vs. single replicas performances**

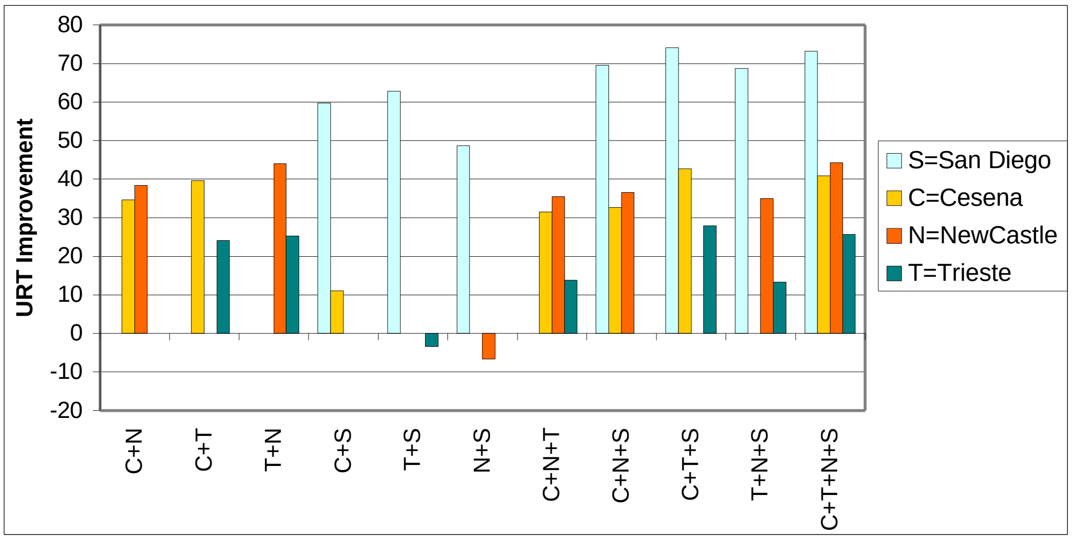

**Figure 9. Percentage improvement of the C$^2$LD mechanism**