Reliable Distribution of Data using Replicated Web Servers*

Moreno Marzolla†
Dipartimento di Informatica, Università Ca' Foscari di Venezia
via Torino 155, 30172 Mestre (VE), Italy
e-mail: marzolla@dsi.unive.it

Abstract

In this paper we describe an algorithm for accessing files over replicated Web Servers. The algorithm breaks the request for a document W, which is assumed to be replicated among N different servers, into N requests such that any K replies are sufficient to reconstruct the whole page. In this way, the document is downloaded from the K fastest servers. The algorithm is feedback-free and does not require any computation to be performed on client or server sides. Thus, it may be implemented on devices with limited computing capabilities. Analytical evaluation of the proposed algorithm shows that, under the model's assumptions, a correctly tuned value for K significantly improves the probability of completing earlier the transfer of W.

1. Introduction

Accessing large documents from geographically distributed servers can be difficult due to unreliable network connections or faulty servers. Servers may occasionally become overloaded, unresponsive, or they may crash. The network itself is subject to transient congestion, and individual links may break down.

The problem of reliable distribution of data has already been considered in the literature. Some of the proposed solutions [2, 3] use error-correcting codes to allow clients to reconstruct missing or damaged data. These solutions require that both the servers and the clients perform computations in order to encode the data to be transferred, and decode the received information. If the clients or servers have limited computational capabilities, this can be difficult to implement.

Other proposals [7, 8] consider the problem of accessing large documents by hosting them on on multiple Web servers and accessing the servers in parallel. Suppose we want to access a document W on N different Web Servers $S_0, S_1, \ldots, S_{N-1}$, where each server has a copy of W. If the servers are geographically distributed, then choosing the "best" server for fetching the page is a nontrivial problem. Not all the replicas could be up and running at the same time, and not all the replicas could provide the same bandwidth and/or the same latency. Moreover, the variable nature of the network may alter the parameters of each client-server connection, slowing down previously fast connections or speeding up congested ones.

In this paper we propose a feedback-free approach for accessing documents over multiple Web Server (WS) replicas. Our proposal is based on the *information dispersal* technique described in [11], and works as follows. Different subsets $R_0, R_1, \ldots R_{N-1}$ of the original page W are requested to the N WS replicas in such a way that any K replies (where K is a user-defined parameter, $1 \leq K \leq N$) are sufficient to reconstruct the page. The size of each request will be shown to be $\frac{|W|}{N}(N - K + 1)$. Setting $K = 1$ means requesting the whole page to all replicas, while setting $K = N$ is equivalent to distributing the requests among all N replicas in such a way that all of them must reply before it is possible to reconstruct the page. Tuning the parameter K allows the client to automatically select the K fastest replicas, at the price of requesting bigger subsets of the document W as K decreases. We evaluate the performance of the proposed algorithm by computing the impact of the parameter K on the probability of completing a transfer in time less than t.

Some techniques for ensuring high availability and responsiveness of Web services are described in [1, 4, 5, 7, 8]. The first techniques were based on constructing a Web service out of replicated servers, which are *locally* distributed in a cluster of workstations, and distributing client's requests among those servers. In [8] it is shown that these techniques are somehow limited in that they may be vulnerable to failures of the gateway interfacing the cluster with

*This work has been partially supported by MIUR Research Project FIRB "Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools".

†Author's current address: INFN Sezione di Padova, via Marzolo 8, 35100 Padova, Italy.

the external network, and they do not take into account issues such as client latency time over the network.

In [5] the authors discuss approaches for providing World Wide Web users with adequate Quality of Service; to do so, they propose a load distribution strategy among WS which are geographically replicated across the Internet. Responsiveness is guaranteed by binding the user to the "most convenient" replica, which they identify as the one providing the shortest user response time. Once the replica is selected the client is permanently bound to it until the document is downloaded.

In [7] the authors propose a mechanism for constructing responsive Web services based on fragmenting the request for Web documents into a number of sub-requests for separate parts of the document. Each sub-request is concurrently issued to a different available WS replica; the system monitors the WS connections and reacts to slowdowns by dynamically switching between the replicas. However, when the number of WS replicas is high, there is a growing probability that at least one of them becomes suddenly congested or unresponsive. Thus, requiring *all* transfers to complete in order to be able to reconstruct the whole page can be inefficient. Timeouts are used in [7] to reduce this problem, but tuning their value is nontrivial. Long timeouts imply that the algorithm adapts slowly, while short timeouts could lead to unnecessary retransmissions. Our approach differs from [7] as it is completely feedback-free; it also differs from [3] in that no computation is required on either side.

This paper is organized as follows. In Section 2 we define the algorithm to access a document W across N WS replicas. Section 3 contains an analytical performance evaluation of the algorithm, assuming a simple network connection model. We then derive the probability of completing the transfer of document W within a deadline t. The model is finally evaluated numerically, and the results are presented in Section 4.

2. The Algorithm

A Web document W is a sequence of $|W|$ bytes, $(b_0, b_1, b_2, \ldots b_{|W|-1})$. The document is assumed to be located at N different WSs, $S_0, S_1, \ldots S_{N-1}$. A *chunk* of W starting from position i and ending at position j, $0 \leq i \leq j \leq |W| - 1$, is denoted as $W[i, j]$ and is defined as $W[i, j] = (b_i, b_{i+1}, b_{i+2}, \ldots b_{j-1}, b_j)$.

A *request* R made to a WS consists of any number of non-overlapping chunks of W. The size of a request R is the total number of bytes it contains. In the following we will use the term "request" to indicate both those generated by the client, containing basically the list of starting and ending positions for the chunks, and the reply generated by a server. The size of a request is thus the number of bytes that are contained in the portion of the document sent from one server to the client.

Algorithm 1 Computation of $R_0, R_1, \ldots R_{N-1}$

Require: K, $1 \leq K \leq N$
Ensure: R_i is the request for server S_i, $0 \leq i \leq N-1$
  fragSize := $|W|/N$
  $t := 0$
  $R_0 := R_1 := \ldots := R_{N-1} := \emptyset$
  for $i = 0$ to $N-1$ **do**
    $W_i := W[i \times \text{fragSize}, (i+1) \times \text{fragSize} - 1]$
    for $j = 1$ to $N - K + 1$ **do**
      $R_t := R_t \cup W_i$
      $t := (t+1) \mod N$
    end for
  end for

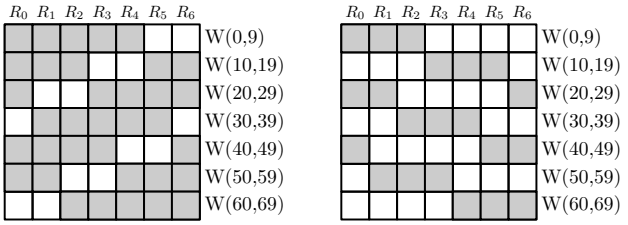
We define a set of N requests, $R_0, R_1, \ldots R_{N-1}$ for a given document W such that R_i will be sent to server S_i, with the following properties: (1) Any K replies are sufficient to reconstruct the whole document, for a fixed $K, 1 \leq K \leq N$; and (2) All the requests have (approximately) the same size.

The requests R_i, $i = 0, \ldots N-1$ can be computed using Algorithm 1. The algorithm divides the page W in N non overlapping chunks $W_0, W_1, \ldots W_{N-1}$, of size $|W|/N$ each. Each chunk is then cyclically inserted into $N-K+1$ different requests. It can be easily seen that this guarantees property (1) above: any chunk W_j will *not* appear in $N-(N-K+1) = K-1$ requests, so any K of them can be chosen with the guarantee that at least one will contain a copy of every chunk. To guarantee property (2), the chunks are evenly distributed among all the requests.

Since each chunk of size $|W|/N$ is present in $N-K+1$ different requests, and there are N chunks, the total size of all the requests is $|W|(N-K+1)$, the size of each request R_i being $|R_i| = |W|(N-K+1)/N$

Fig. 1 illustrates how requests are computed by Algorithm 1. For each $i = 0, \ldots N-1$, request R_i to server S_i is made of the chunks corresponding to the shaded boxes on the i-th column of the grid. The j-th copy of chunk W_i, $j = 0, \ldots N-K-1$, is assigned to request R_t, where $t = (i \times (N-K+1) + j) \mod N$. All the $N-K+1$ instances of the N chunks, are evenly distributed among the N requests in such a way that each request is made of exactly $N-K+1$ chunks, and all the requests have thus the same size.

Algorithm 2 can be used to retrieve the requests from all the available servers $S_0, S_1, \ldots S_{N-1}$. It can be easily implemented by using a feature of version 1.1 of the Hyper Text Transfer Protocol (HTTP/1.1) [13], namely the support for *Byte-Ranges* transfers. HTTP/1.1 compliant WSs accept a `Range` header which can be used to specify which byte ranges of the specified document are requested. The server



(a) $N = 7, K = 3$ (b) $N = 7, K = 5$

Figure 1. Requests generated by Alg. 1

Algorithm 2 Fault-tolerant retrieval of a Web page

Require: $S_0, S_1, \ldots S_{N-1}$ replicas
Require: $1 \leq K \leq N$ minimum replies required to reconstruct the page
Require: W Web document to retrieve
  Compute $R_0, R_1, \ldots R_{N-1}$ using algorithm 1
  $C := 0$ {Number of Requests completed}
  $A := \{S_0, S_1, \ldots S_{N-1}\}$ {Servers still active}
  for $i = 0$ to $N - 1$ **do**
    Asynchronously start request R_i on S_i
  end for
  while $C < K$ **do**
    Wait for any $S_j \in A$ to send part of its request R_j
    if Request R_j of S_j completed **then**
      $A := A - S_j$ {Remove S_j from active servers}
      $C := C + 1$ {Increment completed requests}
      $W := W \cup R_j$ {Update the reconstructed page}
    end if
  end while

encodes the requested fragments into the reply body, and returns a status code 206 (*"Partial Range"*) to the client.

As in [7], the program first interrogates the DNS (Domain Name Server) to get the list of IP addresses associated with the domain name of the requested document. After that, it contacts all the replicas using a HTTP HEAD request. This request is used to check the size of the Web page, whether the requested Web page has been relocated (in such case the new location is contacted), and whether the replica is down or unresponsive (in such case the replica is not used at all).

Given the number N of working replicas, the size $|W|$ of the Web page, and the user-supplied parameter K, the requests for the WSs are first computed using Algorithm 1. At this point, the client opens N asynchronous HTTP connections, one with each WS, and starts receiving the data. As soon as K replies have been completed, all the connections are closed and the page W is reconstructed.

3. Model of Web Server connections

In this section we present an analysis of Algorithm 2. We use a simple model of the behavior of client-server connections based on a Markov Reward Model. The analysis is aimed at calculating the Cumulative Distribution Function for the random variable $T_{N,K}(W)$, which denotes the time needed to complete the transfer of a Web document W in time at most t, given N WS replicas and K sufficient replies to reconstruct W.

We suppose that data transfers between a server and the client happen in bursts, that is, each transfer is made of *active* periods, during which bytes are transferred at a given (fixed) rate Bw, alternating with *idle* periods, where no data transfer takes place. This model is derived from the *packet train model* described in [9]. We assume that the duration of idle and active periods are independent and exponentially distributed random variables.

The connection between WS S_j, $j = 0, \ldots N-1$ and the client can be modeled using a two-state continuous time birth-death Markov Chain (MC). The underlying continuous-time Markov model $X_j = \{X_j(t),\ t \geq 0\}$ is defined over the discrete state space $\{0, 1\}$. When in state 1 (i.e., $X_j(t) = 1$), then the connection is active. When in state 0 (i.e., $X_j(t) = 0$), the connection is idle. The model of a single connection is characterized by three nonnegative parameters: the transition rate λ_j from state 0 to state 1, the transition rate μ_j from state 1 to state 0, and the transfer rate Bw_j when in state 1. We assume that the N client-server network connections are independent.

We compute $\Pr\{T_{N,K}(W) \leq t\}$, the probability of downloading the document W in time at most t from N servers using Algorithm 2 with parameter K, as follows:

$$\begin{aligned}
&\Pr\{T_{N,K}(W) \leq t\} = \\
&= \sum_{i=K}^{N} \Pr\{i \text{ servers replied by time } t\} \\
&= \sum_{i=K}^{N} \sum_{\substack{\Pi \subseteq \{0,1,\ldots,N-1\} \\ |\Pi| = i}} \Pr\left\{\begin{array}{l}\text{Only WSs } \{S_j\}_{j\in\Pi} \\ \text{completed by time } t\end{array}\right\}
\end{aligned} \tag{1}$$

We define $O_j(t)$ the total time spent by the network connection from server S_j to the client in state *active* (i.e., $X_j(t) = 1$) during the interval $[0, t]$:

$$O_j(t) = \int_0^t X_j(s)ds$$

We let D_j to be the time needed to transfer the whole request R_j from server S_j to the client:

$$D_j = \frac{|W|}{Bw_j} \times \frac{N - K + 1}{N}$$

We can now substitute the following in Eq. 1:

$$\Pr\{\text{Only WSs } \{S_j\}_{j\in\Pi} \text{ completed by time } t\} =$$
$$= \prod_{j=0}^{N-1} \left(\Pr\{O_j(t) \geq D_j\} I_{j\in\Pi} + \Pr\{O_j(t) < D_j\} I_{j\notin\Pi}\right)$$

where I_P is the indicator function for predicate P (i.e., $I_P = 1$ if and only if P is true, 0 otherwise).

In order to evaluate $\Pr\{T_{N,K}(W) \leq t\}$ it is necessary to compute $\Pr\{O_j(t) \geq s\} = 1 - \Pr\{O_j(t) < s\}$. This is the *operational time distribution* of the Markov process X_j over the interval $[0, t]$. A closed formula for this distribution has been derived in [6] through *uniformization*. We use one of the algorithms (called "Algorithm I") proposed in [12] to compute the distribution of $O_j(t)$, and thus $\Pr\{T_{N,K}(W) \leq t\}$ (see [10] for details).

4. Numerical Results

In this section we compute the probability $\Pr\{T_{N,K}(W) \leq t\}$ in different scenarios, as a function of t and for various values of K. The aim is to check the effectiveness of Algorithm 2 for different choices of K.

The parameters are the number of WS replicas N, the triple (λ_j, μ_j, Bw_j), $j = 0, \ldots N-1$ characterizing the network connection between server S_j and the client, and the size $|W|$ of the Web document to fetch. We consider a document of size $|W| = 2 \times 10^6 B$ which is fetched from $N = 5$ replicas. We consider four different scenarios, identified with the numbers 1–4. The parameters are set as shown in Table 1. The column *N. of WS* refers to the number of WSs having the parameters set as in the last three columns.

Scenario	**N. of WS**	$Bw\ (B/s)$	λ	μ
1	5	10^5	$1/0.25s$	$1/0.5s$
2	4	10^5	$1/0.25s$	$1/0.5s$
	1	2×10^4	$1/0.5s$	$1/0.5s$
3	2	10^5	$1/0.25s$	$1/0.5s$
	3	2×10^4	$1/0.5s$	$1/0.5s$
4	2	10^5	$1/0.5s$	$1/0.5s$
	2	2×10^4	$1/0.25s$	$1/0.5s$
	1	2×10^4	$1/1s$	$1/0.5s$

Table 1. Parameters used in scenarios 1–4, ($N = 5$, $|W| = 2 \times 10^6 B$)

In Fig. 2 we plot $\Pr\{T_{N,K}(W) \leq t\}$ as a function of t for different values of K. The optimal value of K is the one maximizing the probability $\Pr\{T_{N,K}(W) \leq t\}$ for a given t. We observe that if all the network connections have the same parameters, as in Fig. 2(a), then the best performance is obtained by choosing $K = N$; this is because the size of each request for $K = N$ is $|W|/N$, which is the minimum among all possible choices of K. Since in scenario 1 we are assuming that all connections are equivalent, the best strategy there is the one minimizing the number of bytes transferred over each connection.

Things change if we suppose that one connection has worst performances than the others, as in Fig. 2(b). Choosing $K = N$ in this case yields the worst performances, the best alternative being choosing the four fastest WS replicas. This is because if $K = N$ the performances are dominated by the *slowest* connection, which in this case vanishes the advantage of having smaller requests.

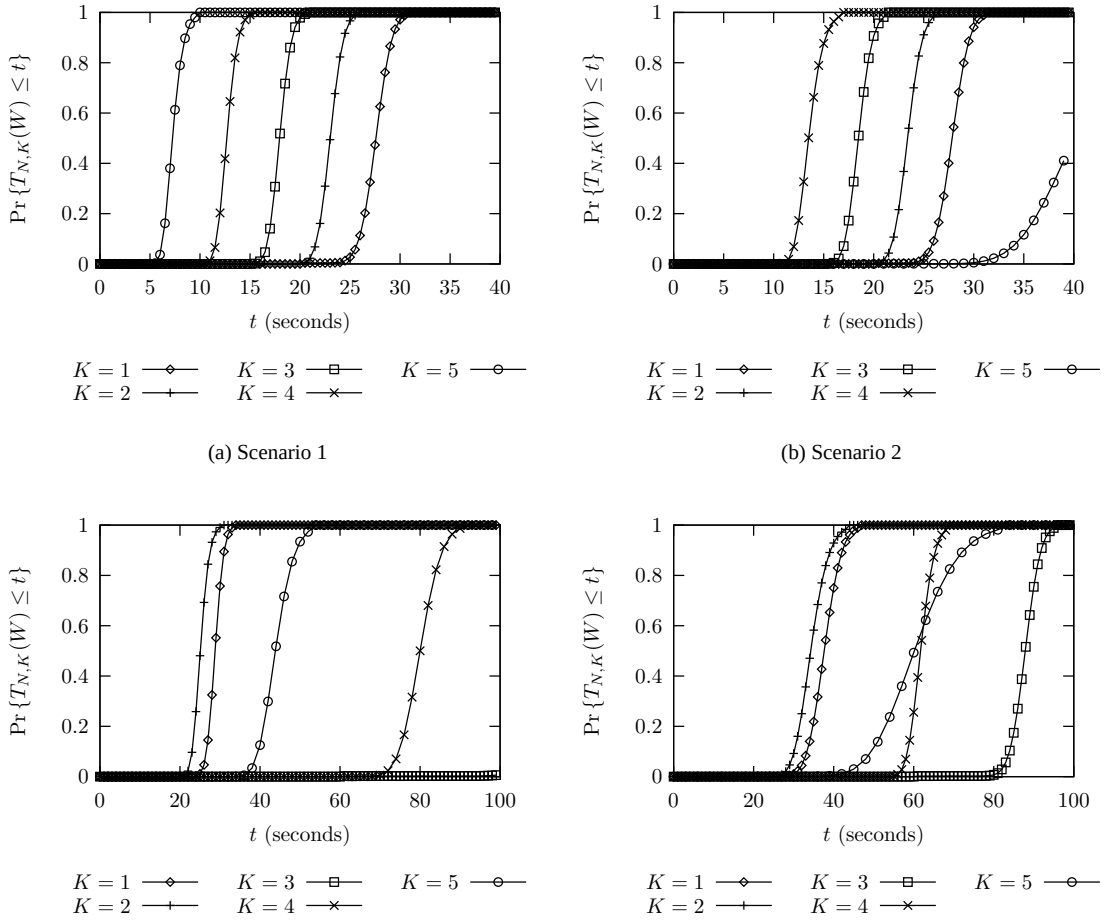
Figures 2(c) and 2(d) depict the behavior of the algorithm in the presence of heterogeneous network connections. Both figures show that the best performances are obtained by setting $K = 2$. Note also that the second best alternative is $K = 1$ (asking the whole document W to all WS replicas), and this approach is better than choosing $K = 3, \ldots, 5$ even if it incurs the greatest overhead on the sizes of the requests. In Fig. 2(d) we note that the curves for $K = 5$ and $K = 4$ cross each other. In this case, deciding which is the best alternative depends on the value of t.

5. Conclusions and future work

In this paper we presented an algorithm for fault-tolerant retrieval of a Web document W replicated among N different WS replicas $S_0, S_1, \ldots S_{N-1}$. The algorithm is evaluated analytically by using Markov model of network connection. The analysis showed that the proposed algorithm can be effective in reducing the expected time to complete the transfer of a document W, which is quite interesting as the algorithm does not require any feedback from servers or from the client, nor it does monitor the network performances. The value of the parameter K, the number of replies which are sufficient to reconstruct W, must be carefully chosen in order to obtain good performance. Higher values of K are recommended if most of the network connections offer high throughput. Lower values of K are recommended if the network connections are highly unbalanced, and only few of them offer high throughput. Past transfers history can be used for this purpose, where available. A fully automatic approach for determining K would be desirable, and is subject of ongoing research.

References

[1] R. B. Bunt, D. L. Eager, G. M. Oster, and C. L. Williamson. Achieving load balance and effective caching in clustered web servers. In *Proc. 4th Int. Web Caching Workshop*, San Diego, CA, Mar. 1999.



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

(d) Scenario 4

Figure 2. Results for scenarios 1–4 ($N = 5$, $|W| = 2 \times 10^6 B$)

[2] J. W. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *Proc. INFOCOM'99*, pages 275–283, 1999.

[3] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proc. SIGCOMM'98*, pages 56–67, 1998.

[4] M. Colajanni, P. S. Yu, and D. M. Dias. Analysis of task assignment policies in scalable distributed web server systems. *IEEE Trans. on Parallel and Distributed Systems*, 9(6):585–600, June 1998.

[5] M. Conti, E. Gregori, and F. Panzieri. QoS-based architectures for geographycally replicated web servers. *Cluster Computing*, 4(2):109–120, Apr. 2001.

[6] E. de Souza e Silva and H. R. Gail. Calculating cumulative operational time distribution of repairable computer systems. *IEEE Trans. on Computers*, 1(35):322–332, Apr. 1986.

[7] V. Ghini, F. Panzieri, and M. Roccetti. Client-centered load distribution: A mechanism for constructing responsive web services. In *Proc. HICSS'34*, Maui, Hawaii, Jan. 2001. IEEE Computer Press.

[8] D. Ingham, F. Panzieri, and S. K. Shrivastava. Constructing dependable web services. *IEEE Internet Computing*, 4(1):25–33, January/February 1999.

[9] R. Jain and S. Routhier. Packet trains – measurements and a new model for computer network traffic. *IEEE J. on Sel. Areas of Comm.*, 4(6):986–995, Sept. 1986.

[10] M. Marzolla. Design and analysis of a fault-tolerando web retrieval algorithm. Tech. Report CS-2002-5, Dip. di Informatica, Università Ca' Foscari, Venezia, Italy, Apr. 2002.

[11] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. of the ACM*, 36(2):335–348, Apr. 1989.

[12] G. Rubino and B. Sericola. Interval availability distribution computation. *IEEE Trans. on Computers*, pages 48–55, 1993.

[13] World Wide Web Consortium. Hypertext transfer protocol – HTTP/1.1. RFC 2616. Available online at http://www.w3.org/Protocols/rfc2616/rfc2616.txt.