Design and Analysis of a Fault-Tolerant Web Retrieval Algorithm

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 which allows users to access documents over replicated World Wide Web Servers in a fault-tolerant way. The algorithm breaks the request for a Web 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 algorithm downloads the document from the K fastest servers. The impact of the value of K on the performances of the retrieval algorithm is evaluated using a simple analytical model for the network connections between the client machine and the servers. Results show that, under the model's assumptions, a correctly tuned value for K can improve significantly the probability of completing earlier the transfer of W.

Keywords Web Architectures, Reliability Analysis, Fault-Tolerant Systems, Analytic Modeling.

1 Introduction

Accessing large documents over the World Wide Web can be problematic due to the unstable nature of the network. Web Servers (WSs) may become overloaded, unresponsive and occasionally crash. The network itself is subject to transient congestions, and individual links may break down. However, many Web documents are hosted on multiple WS replicas. Multiple replicas ensure scalability, and recently [6, 7] it has been shown that they may be used to improve the responsiveness when accessing Web documents. Multiple geographically-distributed replicas guarantee a certain level of fault-tolerance, since each WS usually fails independently from the others.

Suppose we want to access a document W residing on N different Web Servers $S_0, S_1, \ldots, S_{N-1}$, where each server has an identical copy of W. If the servers are geographically distributed, then choosing the "best" server for fetching the page is a non-trivial 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 situation different approaches can be followed. The whole page W can be requested to all the replicas at the same time; the fastest server is automatically chosen, in that it will be the first one to provide the page. Assuming that the client machine does not constitute a bottleneck, this solution has the obvious disadvantage of wasting network bandwidth, as all the servers are sending the whole page W, even if only the copy provided by the fastest server will actually be used. An alternative approach would be that of subdividing the page W in N equally sized, non overlapping fragments $W_0, W_1, \ldots W_{N-1}$. The size of each fragment is $|W|/N$. For each i, $0 \leq i \leq N-1$, fragment W_i is requested to server S_i. When all servers complete their request, it is possible to assemble the whole page from the fragments. This approach does not waste bandwidth since the total number of bytes sent over the network is exactly equal to the size of page W. Unfortunately, this algorithm is sensible to network congestions and servers availability, since it requires all the replies to reconstruct the whole page. In this situation, the time needed to fetch the page is dominated by the *slowest* WS replica, and the situation tends to get worse as the number N of replicas increases, as it will be more likely that at least one replica slows down its transfer for any reason.

In this paper a new strategy for accessing documents over multiple WS replicas is shown. This methodology is based on a simple variant of the *information dispersal* technique described in [14], 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.

In order to evaluate the algorithm's performances, we compute the impact of the choice of parameter K on the probability of completing a transfer in time less than t. To do so, we consider a simplified Markov model of network connection based on the concept of *packet trains* proposed in [9] with additional assumptions explained in section 4. The computation of the probability of completion in time less than t is evaluated by computing the Operational Time Distribution for the Markov model, using the algorithm described in [15]. Analytic modeling allows us to obtain an early performance characterization of the algorithm. In particular, we observe that a correct choice of K can dramatically improve the probability of completing the transfer prior to a given deadline. We observe also a strong dependence of the performances from the value of K, which means that an incorrect choice for K can impact negatively the time needed to transfer the document.

This paper is organized as follows. In Section 3 the algorithm will be defined. Section 4 contains a mathematical analysis of the performances of the algorithm, under a simplified model for the behavior of network connections. The model is used to derive the probability of completing the transfer of a Web document W within a deadline t. The model is then evaluated numerically, and the results will be presented in Section 5.

2 Previous work

There exists a vast literature on techniques for ensuring high availability and responsiveness of Web services [2, 3, 4, 6, 7]. The first techniques considered 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 [7] it is shown that these techniques are somehow limited in that they may be vulnerable to failures of the gateway interfacing the cluster with the external network, and they do not take into account issues such as client latency time over the network.

In [4] the authors discuss strategies for providing World Wide Web users with adequate Quality of Service; to do so, they propose a strategy implementing load distribution among WS replicated across the Internet (rather than in a cluster of workstations). Responsiveness is guaranteed by binding the user to the "most convenient" replica, which they identify as the one providing the shortest user response time.

In [6] a mechanism for constructing responsive Web services is developed and implemented. The mechanism is called "Client-Centered Load Distribution" (C^{2}LD). Responsiveness is achieved by intercepting each client's browser request for a Web page, and fragmenting the request 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 replies received from the replicas are assembled at the client end, and the page is reconstructed and delivered to the client's browser. C^{2}LD is capable of dynamically reacting to changes in the network and the WS replicas; to do so, a monitor mechanism periodically checks all replicas and determines for each one the optimal fragment size so that the whole transfer can be completed within a user-defined deadline. This monitoring mechanism guarantees that replicas which are temporarily unavailable, or providing poor performances, are recognized and placed in a separate "stand-by" list and not used until they become responsive again.

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, engaging a data transfer with each available replica, requiring all transfers to complete in order to be able to reconstruct the whole page, can be inefficient and lead to poor performances. Timeouts can be used to reduce this problem, but their tuning is nontrivial. Long timeouts imply that the algorithm adapts slowly, while short timeouts could lead to unnecessary retransmissions.

Here are mainly concerned with the issue of providing a Web document W to the user while tolerating a certain degree of network or server failures. Rather than assuming that all servers will eventually reply, and treating server/network failures as unexpected events, our approach assumes that a subset of the available WS replicas could provide poor performances or become completely stuck. To overcome these problems, the request for the Web document W is divided in a number of partially overlapping sub-requests, so that any K out of N of them must complete before the page can be reconstructed, where N is the number of the available WS replicas. The parameter K, $1 \leq K \leq N$ is chosen by the user. This mechanism is in fact equivalent to selecting the K fastest WSs to download the page, even when the notion of "fastest WS" changes over the time and is thus impossible to know *a priori*. We don't make use of any timeout mechanism, yet the proposed algorithm, once K is defined, is completely adaptive in that it tolerates up to $N - K$ failures (or $N - K$ poorly-performing connections). However, to achieve this the algorithm introduces some redundancy in the requests sent to the WS replicas, and the amount of redundancy depends on K as will be described in detail in the next section.

The analysis presented in Section 4 is aimed at identifying the impact of the choice of the parameter K under a simplified analytical model of network connection. Different

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

```
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
```

scenarios are illustrated in Section 5.

3 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 chunks of W; overlapping portions of different chunks only appear once in a request. 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.

We want to 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 $K, 1 \leq K \leq N$;

2. All the requests have the same size.

The requests R_i, $0 \leq i \leq N-1$ 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 using the loop of lines 6–9. 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

$$\sum_{i=0}^{N-1} |R_i| = N \times \frac{|W|}{N}(N-K+1) = |W|(N-K+1) \tag{1}$$

The size of each request R_i being

$$|R_i| = \frac{|W|}{N}(N-K+1) \tag{2}$$

Fig. 1 illustrates an example of how requests are computed by Algorithm 1. The algorithm assigns each chunk to $N-K+1$ different requests in turn, starting from R_0 through R_{N-1}, wrapping back to R_0 and starting again. The j-th copy of chunk W_i, $0 \leq i \leq N-1$, $0 \leq j \leq N-K$, is assigned to request R_t, where

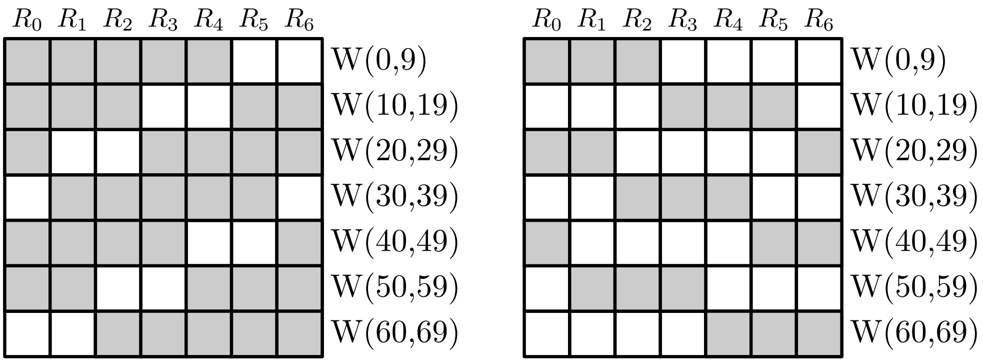
$$t = i \times (N-K+1) + j \mod N \tag{3}$$

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 now 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) [17], 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 encodes the requested fragments into the reply body, and returns a status code 206 (*"Partial Range"*) to the client.

As in C^{2}LD [6], 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;



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

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

Figure 1: Example of requests generated by Algorithm 1 applied to a page W of size 70, replicated among $N = 7$ Web Servers. On the left we set $K = 3$; on the right we set $K = 5$. 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, $0 \leq i \leq N - 1$.

- whether the requested Web page has been relocated (in such case the new location is contacted);
- whether the replica is down or unresponsive (in such case the replica is not used at all by the algorithm).

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 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.

4 Model of Web Server connections

In this section an analysis of Algorithm 2 is presented. The analysis is performed using a simple analytical 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

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
    Block waiting 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 server $S_j$ from active servers
        $C := C + 1$                    Increment counter of completed requests
        $W := W \cup R_j$                      Update the reconstructed page $W$
    end if
end while
```

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]. In the original packet train model, the network traffic consists of a number of packet streams between various pairs of nodes on the network. Each stream is made by a number of trains, and a train is a sequence of packets. The gap between trains is large with respect to the gap between packets in a train. We simplify the model as follows: we assume that the gaps between trains and train lengths are independent and exponentially distributed random variables. Moreover, we neglect the internal structure of each train, considering a train as a continuous stream of bytes arriving at a constant rate Bw. With these assumptions, the model reduces to a Markov-Modulated process with two states, corresponding to active and idle periods [8].

The bursty behavior of network traffic is well known [10, 13]. Here we make the additional assumption that the duration of active and idle periods are independent and exponentially distributed random variables. The exponentiality hypothesis is quite strong,

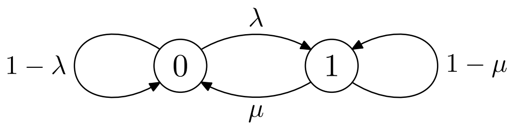


Figure 2: Birth-death MC modeling the active/idle behavior of a network connection.

and it has been shown not to hold in the case of packet interarrivals during Telnet connections, and in the case of FTPDATA chunks interarrival times during FTP connections [13]. However, our aim is not to develop a realistic model of network connection; we want only to represent its bursty behavior, since this is one of the most important factors impacting the performances of Web retrieval algorithms.

Under these assumptions, the behavior of the connection between a WS and the client can be modeled using the continuous-time birth-death Markov Chain (MC) depicted in Fig. 2. The underlying continuous-time Markov model $X = \{X(t),\ t \geq 0\}$ is defined over the discrete state space $\{0, 1\}$. When the system is in state 1, then the connection is active. When the system is in state 0, the connection is idle. The model of a single connection is characterized by three nonnegative parameters:

λ	Transition rate from state 0 (*Idle*) to state 1 (*Active*)
μ	Transition rate from state 1 to state 0
Bw	Transfer rate of the connection when in state *Active*

We assume that the N network connections between the servers and the client are independent, so the connection between replica S_j and the client is characterized by its own parameters (λ_j, μ_j, Bw_j), $0 \leq j \leq N-1$, which may be different from those of other connections. Moreover, we suppose that each WS replica has been running enough to reach a steady-state condition when the client's request arrives. This means that, at time $t = 0$, the system is in equilibrium. In order to 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, we have:

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

The probability that a WS completes its transfer before a given deadline t can be computed as follows. We denote with Q the infinitesimal generator matrix for the MC of Fig. 2, which is defined as:

$$Q = [Q_{ij}] = \left(\begin{array}{cc} -\lambda & \lambda \\ \\ \mu & -\mu \end{array} \right) \qquad (4)$$

The initial condition $\pi = (\pi_0, \pi_1)$ is set to the steady-state probability vector for the MC , and is [1]:

$$\pi_0 = \frac{\mu}{\lambda + \mu} \qquad \pi_1 = \frac{\lambda}{\lambda + \mu} \qquad (5)$$

W	The Web document to fetch
N	Number of WS replicas
K	Minimum number of replies necessary to reconstruct the page W, $1 \leq K \leq N$
λ	Transition rate from state *Idle* to state *Active*
μ	Transition rate from state *Active* to state *Idle*
Bw	Bandwidth available when a connection is in state *Active*
$T_{N,K}(W)$	Time needed to transfer the page W from at least K out of N WS replicas.
π_0, π_1	Probability that, at the steady state, the Markov process of Fig. 2 is in state 0 and 1 respectively.
$O(t)$	Operational time during the interval $(0, t)$; it is the time spent by the Markov process of Fig. 2 in state 1 during the interval $(0, t)$.
Q	Transition Matrix for the Markov process of Fig. 2.
q	Uniformization rate, which verifies $q \geq \max_{ij} \lvert Q_{ij} \rvert$
P	Uniformized transition matrix.

Table 1: Symbols used in the paper

We consider the Markov process $\{X(t),\ t \geq 0\}$ over the finite time interval $(0, t),\ t \geq 0$. We compute the Cumulative Distribution Function for the time a network connection remained in state *Active*. This is exactly the *cumulative operational time* as defined in [12, 15]. Denoting with $O(t)$ the total time spent in state 1 by the MC of Fig. 2, over the time interval $(0, t)$:

$$O(t) = \int_0^t I(s)ds \tag{6}$$

where

$$I(s) = \begin{cases} 1 & \text{if } X(s) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

This defines a *Markov Reward Model*: $I(s)$ represents the reward rate associated with the states of the MC . $O(t)$ is the total reward accumulated over the time interval $(0, t)$.

A closed formula for the distribution $\Pr\{O(t) \leq s\}$, $0 \leq s < t$, has been derived in [5] through *uniformization*. Efficient algorithms to evaluate the distribution are described in [11, 12, 15, 16]. In particular, we use one of the algorithms (called "Algorithm I") proposed in [15] to compute the distribution of $O(t)$. We recall briefly how that algorithm works.

Let $P = [P_{ij}]$ denote the uniformized probability matrix for the infinitesimal generator matrix Q. Let q be the *uniformization rate*, which verifies $q \geq \max_{ij} |Q_{ij}|$. P is related to Q by

$$P = I + \frac{Q}{q} \tag{8}$$

I being the identity matrix.

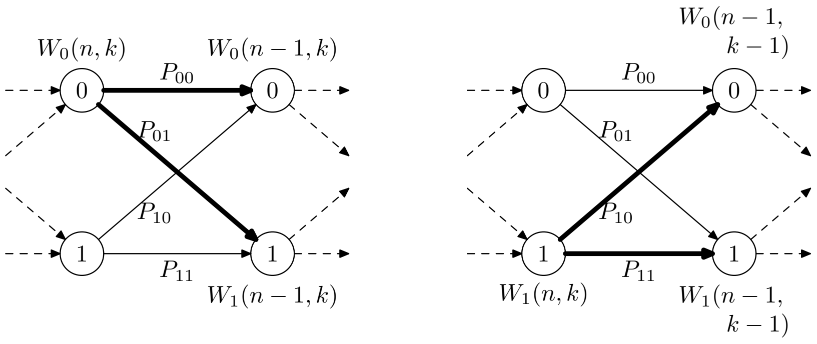
In [15] it is shown that the following equation holds, for $s < t$:

$$\Pr\{O(t) \leq s\} = \sum_{n=0}^{+\infty} e^{-qt} \frac{(qt)^n}{n!} \sum_{k=0}^{n} \binom{n}{k} \left(\frac{s}{t}\right)^k \left(1 - \frac{s}{t}\right)^{n-k} W(n, k) \tag{9}$$

where $W(n, k)$ is the probability that the uniformized Markov process visits at most k *Active* states during the first n transitions. $W(n, k)$ can be evaluated recursively in the following way. Let $W_0(n, k)$ and $W_1(n, k)$ be the probabilities that the uniformized Markov process visits at most k *Active* states during the first n transitions, given that the initial state is 0 and 1 respectively. We have that

$$W(n, k) = \pi_0 W_0(n, k) + \pi_1 W_1(n, k)$$

(remember that π_0 and π_1 are the initial probabilities associated with states 0 and 1 respectively, and were defined in Eq. (5)).



(a)

(b)

Figure 3: Computation of the function $W_0(n,k)$ 3(a) and $W_1(n,k)$ 3(b)

$W_0(\cdot,\cdot)$ and $W_1(\cdot,\cdot)$ are defined as:

$$W_0(n,k) = \begin{cases} P_{01}W_1(n-1,k) + P_{00}W_0(n-1,k) & \text{if } n>0 \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

$$W_1(n,k) = \begin{cases} P_{11}W_1(n-1,k-1)+ & \\ \quad P_{10}W_0(n-1,k-1) & \text{if } n>0 \text{ and } k>0 \\ 0 & \text{if } n=0 \text{ and } k=0 \\ 1 & \text{if } n=0 \text{ and } k=1 \end{cases} \tag{11}$$

(see Fig. 3). Also we have that $W_0(n,n+1) = W_1(n,n+1) = 1$, for each $n \geq 0$.

From the computational point of view, it is necessary to truncate the infinite sum in Eq. (9):

$$\begin{aligned} \Pr\{O(t) \leq s\} &= e(N) + e'(N,C) + \\ &+ \sum_{k=0}^{C}\sum_{n=k}^{N} e^{-qt}\frac{(qt)^n}{n!}\binom{n}{k}\left(\frac{s}{t}\right)^{n-k}\left(1-\frac{s}{t}\right)^k W(n,n-k) \end{aligned}$$

In [15] it is proved that $e(N)$ and $e'(N,C)$ can be bounded as follows:

$$e(N) \leq 1 - \sum_{n=0}^{N} e^{-qt}\frac{(qt)^n}{n!} \tag{12}$$

$$e'(N,C) \leq W(N,N-C)\left(1 - \sum_{k=0}^{C} e^{-q(t-s)}\frac{(q(t-s))^k}{k!}\right) \tag{13}$$

Given a tolerance error ϵ specified by the user, the first truncation on N can be computed from Eq. (12) as

$$N = \min \left\{ n \in \mathbf{N} \;\middle|\; \sum_{j=0}^{n} e^{-qt} \frac{(qt)^j}{j!} \geq 1 - \frac{\epsilon}{2} \right\} \tag{14}$$

Once N is known, the second truncation value C can be computed using Eq. (13) as:

$$C = \min \left\{ c \leq N \;\middle|\; W(N, N-C) \left(1 - \sum_{k=0}^{c} e^{-q(t-s)} \frac{(q(t-s))^k}{k!} \right) \leq \frac{\epsilon}{2} \right\} \tag{15}$$

The probability that S_j, $0 \leq j \leq N-1$, completes the transfer of its request R_j in time at most t, can be expressed as $\Pr\{O_j(t) \geq D_j(W)\}$, where $D_j(W)$ is the cumulative operational time needed to transfer the reply from server S_j to the client, and is:

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

The probability $\Pr\{O_j(t) \geq D_j(W)\}$ can be evaluated using Eq. (9), replacing λ, μ, Bw with λ_j, μ_j, Bw_j respectively. $O_j(t)$ is the cumulative operational time of the connection between server S_j and the client over the time interval $(0, t)$.

$\Pr\{T_{N,K}(W) \leq t\}$ can now be written as:

$$\Pr\{T_{N,K}(W) \leq t\} = \\ \sum_{i=K}^{N} \sum_{\substack{\Pi \subseteq \{0,1,\ldots,N-1\} \\ |\Pi| = i}} \prod_{j=0}^{N-1} \left(\Pr\{O_j(t) \geq D_j(W)\} I_{j \in \Pi} + \Pr\{O_j(t) < D_j(W)\} I_{j \notin \Pi} \right)$$

where I_P is the indicator function for predicate P.

5 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), $0 \leq j \leq N-1$ characterizing the network connection between server S_j and the client, and the size $|W|$ of the Web document to fetch. We will consider two main scenarios, one were a document of size $|W| = 2 \times 10^6 B$ is fetched from $N = 5$ replicas, and the other where the document size is $|W| = 10^7 B$ and the number of replicas is $N = 10$. We define two more factors characterizing the various scenarios. They are named *Network Speed* and *Connection quality*. The network speed is simply the value for the bandwidths Bw_j, while connection quality refers to the values for λ_j, μ_j. The settings are summarized in the following table:

Name	**Types**	**Values**
Network speed	Fast	$Bw = 10^5 B/s$
	Slow	$Bw = 2 \times 10^4 B/s$
Connection quality	Good	$\lambda = 1/0.25s$, $\mu = 1/0.5s$
	Poor	$\lambda = 1/0.5s$, $\mu = 1/0.5s$
	Very Poor	$\lambda = 1/1s$, $\mu = 1/0.5s$

Seven different scenarios, identified with the numbers 1–7, will be analyzed. Scenarios 1–4 are characterized by $N = 5$ and $|W| = 2 \times 10^6 B$. Scenarios 5–7 have $N = 10$ and $|W| = 10^7 B$. The remaining factors are set as shown in Tables 2 and 3.

Scenario	**N. of WS**	**Network Speed**	**Connection quality**
1	5	Fast	Good
2	4	Fast	Good
	1	Slow	Poor
3	2	Fast	Good
	3	Slow	Poor
4	2	Fast	Poor
	2	Slow	Good
	1	Slow	Very Poor

Table 2: Scenarios 1–4, ($N = 5$, $|W| = 2 \times 10^6 B$)

The column *N. of WSs* refers to the number of WSs having the factors set as in the last two columns. For example, in scenario 4 we have $N = 5$ replicas, two of which have a fast network speed but a poor connection quality, two have a slow speed and good connection quality, and the other one has a slow speed and a very poor connection quality.

In Figures 4 and 5 we plot $\Pr\{T_{N,K}(W) \leq t\}$ as a function of t, for different values of K. In general, the best value of K is the one maximizing the probability $\Pr\{T_{N,K}(W) \leq t\}$ for a given t.

The results for scenarios 1–4 are shown in Fig. 4. We observe that if all the network connections have the same parameters, as in Fig. 4(a), then the better performances of

Scenario	**N. of WSs**	**Network Speed**	**Connection quality**
5	10	Fast	Good
6	3	Fast	Good
	7	Slow	Very Poor
7	1	Fast	Good
	4	Fast	Poor
	5	Slow	Very Poor

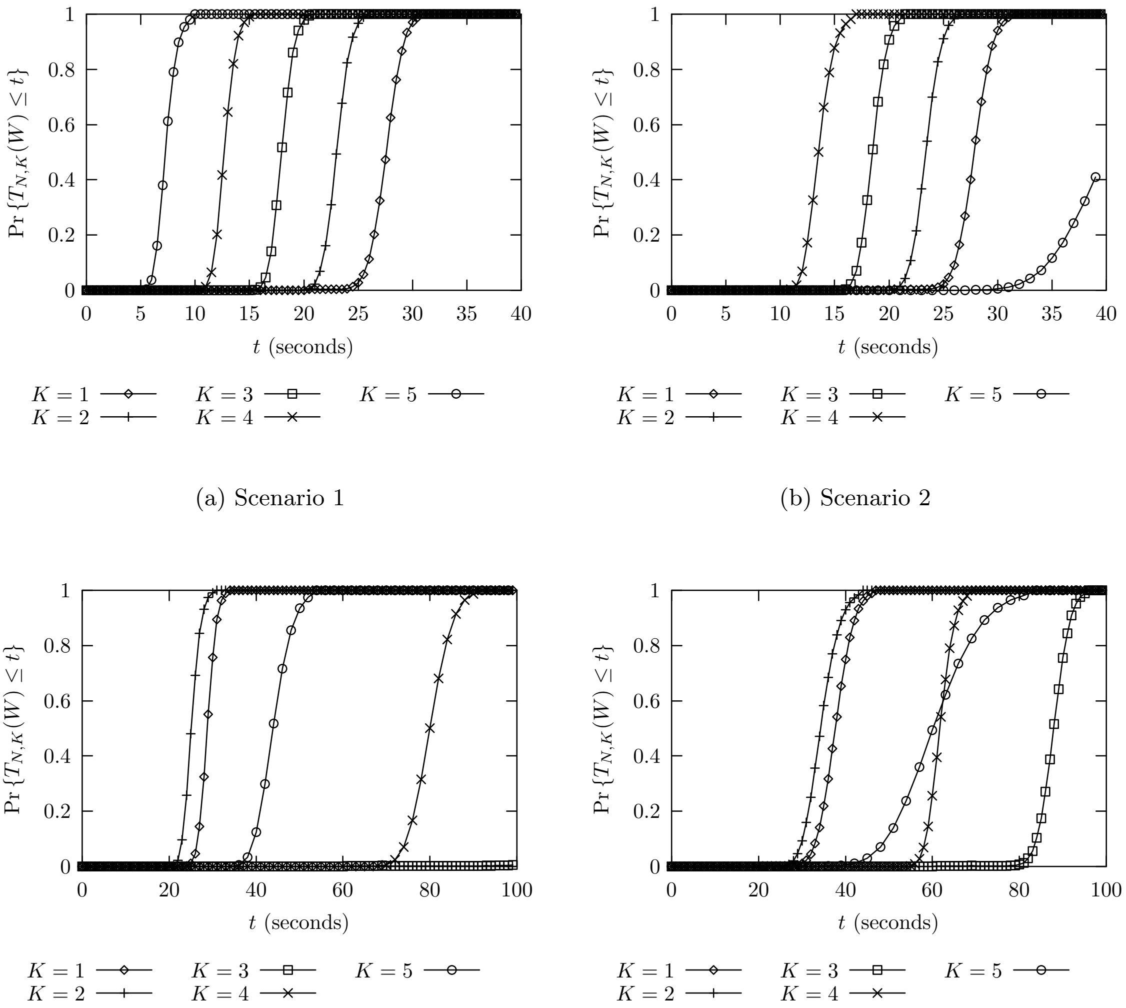
Table 3: Scenarios 5–7 ($N = 10$, $|W| = 10^7 B$)

Algorithm 2 are 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 better strategy is obviously 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. 4(b). We observe that choosing $K = N$ in this case yields the worst performances, the the best alternative in this case 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 4(c) and 4(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 $K = 2$, that is, selecting the 2 fastest replicas (it's not by chance that in scenarios 3 and 4 we have exactly 2 fast network connections). Note however 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 imposes the greatest overhead on the sizes of the fragments requested to the servers. In Fig. 4(d) we note that the curves corresponding to $K = 5$ and $K = 4$ cross each other. In this case, deciding which alternative should be considered better than the other depends on the value of t.

The results for scenarios 5–7 are shown in Fig. 5. Again, when all the network connections have the same parameters as in Fig. 5(a) then the best approach is to download the document W from all the WSs, setting $K = N$. In Fig. 5(b) we can see that the best performances are obtained by setting $K = 3$, that is, downloading the page from the 3



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

(d) Scenario 4

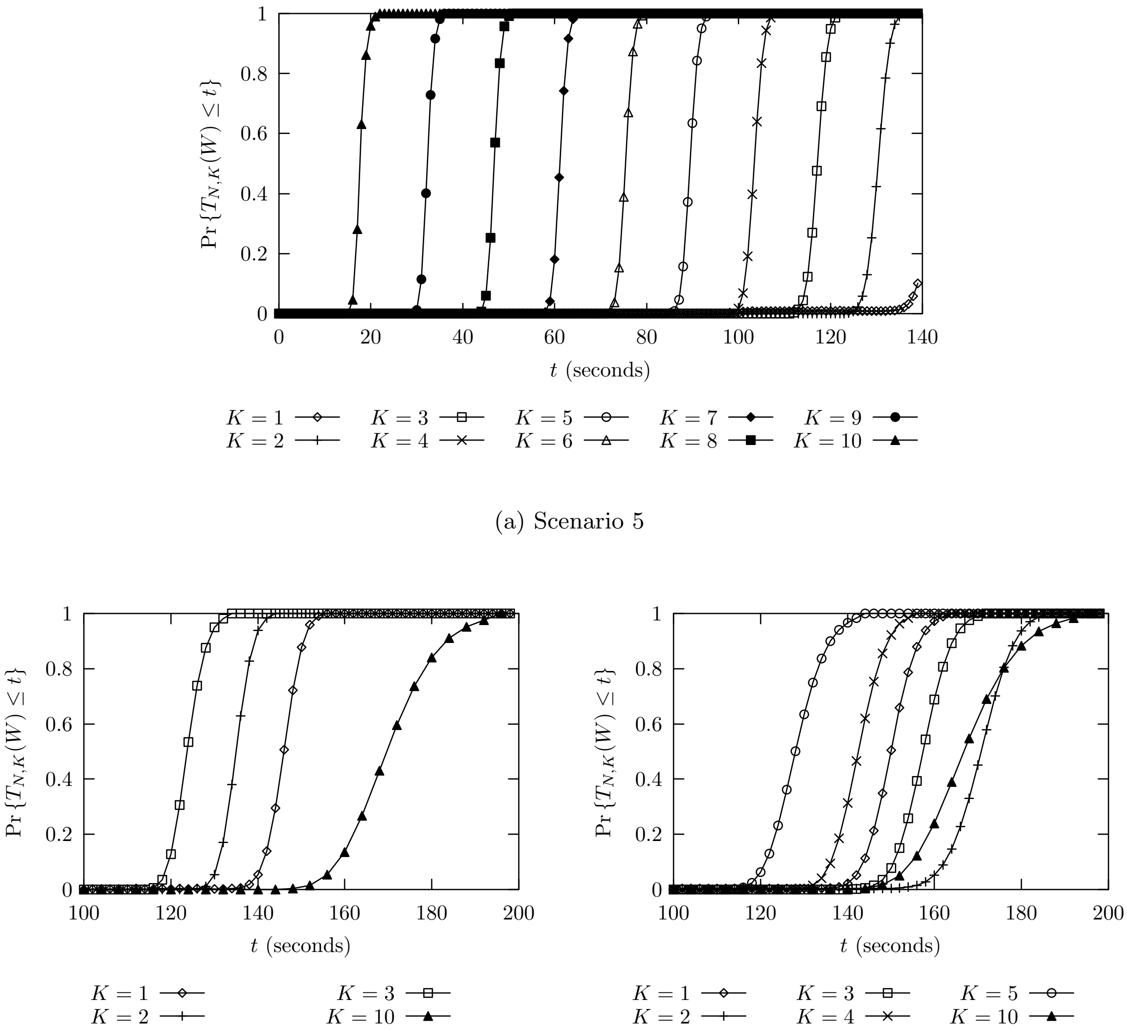
Figure 4: Results for scenarios 1–4 ($N = 5$, $|W| = 2 \times 10^6 B$)

fastest WSs. For $K = 3$, the size of the request made to each WS replica is (see Eq. (2)):

$$\frac{|W|}{N} \times (N - K + 1) = 0.8 \times |W|$$

This is eight times $0.1 \times |W|$, which is the request size if $K = 10$. However, since in Scenario 6 there are seven network connections which are considerably slower than the three fast ones, setting $K = 10$ yields worst performances with respect to $K = 3, 2, 1$, as the faster connections can transfer bigger requests faster than slow connections can transfer smaller ones.

In Fig. 5(b) we note that the curves corresponding to $K = 4$ up to $K = 9$ are not drawn. This is because the corresponding value of Pr $\{T_{N,K}(W) \leq t\}$ is very near to zero



(a) Scenario 5

(b) Scenario 6

(c) Scenario 7

Figure 5: Results for scenarios 5–7 ($N = 10$, $|W| = 10^7 B$). In 5(b) and 5(c), all omitted curves were zero in the plotted interval.

in the plot's range. This means that, for example, $K = 10$ is preferred to $K = 4, \ldots, 9$ in this case.

The situation illustrated by Fig. 5(c) is more complicated. We can see that the alternative yielding the best performances is in this case $K = 5$, followed by $K = 4$, $K = 1$ and $K = 3$. It is particularly interesting to note that setting $K = 1$ improves the probability of getting earlier the page W with respect to setting $K = 3$. Once again, this is caused by the difference of performances of the fastest network connection with respect to the others. Setting $K = 3$ causes smaller requests than $K = 1$, but at the price of requiring three replies from three different hosts to build the page, and the performances are dominated by those of the slowest network connections.

6 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 $S_0, S_1, \dots S_{N-1}$. The algorithm computes N different requests $R_0, R_1, \dots R_{N-1}$ according to a user-defined parameter K, each request being a subset of the original document W. All the requests have the same size, and request R_j is sent to server S_j, $0 \leq j \leq N-1$. The set $\{R_j\}_{0 \leq j \leq N-1}$ has the property that any K elements are sufficient to reconstruct W. Thus the algorithm is able to download the page from the K fastest WS replicas, without probing their speed.

The algorithm is evaluated analytically by using a simple model of network connection. Each connection is assumed to be, at any time, in one of two states: *active* and *idle*. During active periods, data is transferred at a constant rate, specific for each connection, while during idle periods no data transfer occurs. The lengths of active and idle periods are independent and exponentially distributed random variables. The model is evaluated numerically using the algorithm proposed in [15] to obtain $\Pr\{T_{N,K}(W) \leq t\}$, the probability of completing the transfer of page W from at least K out of N WS replicas, in time at most t. The model showed that the proposed algorithm can be effective in reducing the expected time to complete the transfer of a document W, which is quite remarkable since the algorithm does not employ any mechanism to poll the hosts to verify their responsiveness, nor it checks the network connections to assess their performances. The critical problem is deciding the value for K. Choosing a too conservative (that is, low) value can result in bad performances as larger sub-pages are requested to the WSs. Choosing a too optimistic value results in smaller requests, but this requires a greater number of replies to reconstruct the page, which raises the probability that at least one network connection performs badly during the transfer period, limiting the overall performances.

Some automatic ways to compute in advance the parameter K are clearly desirable, and are the subject of ongoing research. Moreover, we are working on a more realistic model of WSs and network connections, which would allow early evaluations of variants of Algorithm 2 or other similar Web retrieval algorithms. A C++ program implementing Algorithm 2 is being developed, and will be used to assess the performances on the real Internet. Such test program could be very useful to compare the performances of the algorithm proposed in this paper with respect to C^{2}LD mechanism described in [6].

References

[1] Bolch, G., Greiner, S., Trivedi, K. S., and de Meer, H. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation With Computer Science Applications*. Wiley-Interscience, 1998.

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

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

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

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

[6] Ghini, V., Panzieri, F., and Roccetti, M. Client-centered load distribution: A mechanism for constructing responsive web services. In *Proc. of 34th IEEE Hawaii International Conference on System Sciences (HICSS'34)* (Maui, Hawaii, Jan. 2001), IEEE Computer Press.

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

[8] Jagerman, D., Melamed, B., and Willinger, W. Stochastic modeling of traffic processes. In *Frontiers in Queuing: Models, Methods and Problems*, J. Dshalalow, Ed. CRC Press, 1996.

[9] Jain, R., and Routhier, S. Packet trains – measurements and a new model for computer network traffic. *IEEE Journal on Selected Areas in Communications 4*, 6 (Sept. 1986), 986–995.

[10] Leland, W., Taqqu, M., Willinger, W., and Wilson, D. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transactions on Networking 2*, 1 (Feb. 1994), 1–15.

[11] Nabli, H., and Sericola, B. Performability analysis of fault-tolerant computer systems. Technical Report 2254, Unité de Recherche Inria Rennes, IRISA, Campus Unitersitaire de Beaulieu, 35042 RENNES Cedex (France), May 1994.

[12] Nabli, H., and Sericola, B. Performability analysis: A new algorithm. *IEEE Transactions on Computers 45*, 4 (Apr. 1996).

[13] Paxson, V., and Floyd, S. Wide-area traffic: The failure of Poisson modelling. *IEEE/ACM Transactions on Networking 3*, 3 (June 1995), 226–244.

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

[15] Rubino, G., and Sericola, B. Interval availability distribution computation. *IEEE Transactions on Computers* (1993), 48–55.

[16] Sericola, B. Interval-availability distribution of 2-state systems with exponential failures and phase-type repairs. *IEEE Transactions on Reliability 43*, 2 (June 1994), 335–343.

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