Optimizing Download Time of Embedded Multimedia Objects for Web Browsing

Thanasis Loukopoulos and Ishfaq Ahmad, *Senior Member*, *IEEE*

Abstract—Notoriously high delays in accessing Web pages loaded with massive multimedia objects are highly undesirable. Inspired by the requirements of news agencies and other information providers to include multimedia content in their pages, this paper proposes a new solution to the problem of minimizing the Web response time. We consider an environment that consists of a central multimedia repository and various sites physically dispersed. Our approach is based on simultaneous downloading of some of the embedded multimedia objects from the repository, and the rest from the regional servers. We propose a cost model to formalize the relative benefits of the proposed scheme, and design an algorithm that replicates multimedia objects so as to take advantage of concurrency in data transferring. An extensive simulation study evaluates the performance of the proposed replication policy under storage and processing capacity constraints, as well as with various network transfer rates. Comparisons are carried out with alternative schemes.

Index Terms—Parallel downloads, replica placement, multimedia repository, Web, Internet.

1 Introduction

The World Wide Web is the de facto source of information for common users and is responsible for the majority of Internet traffic. Web pages usually consist of a text part and various embedded objects (e.g., images, applets, audio files, and video clips.). Typically, all components of a page are stored and downloaded by the same server. With a proper distribution scheme, a request to a page and the subsequent requests for the embedded objects can be redirected appropriately toward multiple servers that can concurrently transmit the data for the whole page. The current replication/redirection schemes aim at distributing the requests to identical mirror servers so as to balance their loads. Thus, any concurrency exhibited in transmitting a page is an ad hoc result of the load balancing process.

In this paper (parts of which appeared in [15]), we exploit the ability to explicitly distribute the HTTP requests for embedded objects in order to decrease the retrieval time. The distribution is made offline by rewriting the HTML part of a page, altering the URLs of the embedded objects so that they point to multiple servers. Such a technique avoids some of the overheads related to redirection since the client addresses the requests directly to the additional servers. The method is applicable to objects that require a complete download before being presented at the client's browser and not for streaming media. Throughout the paper, we refer to such objects with the term multimedia objects (MOs), even though, in certain cases (e.g., images), they involve only one medium.

We target sites containing pages heavily loaded with MOs, e.g., pages with many images and/or animation files. Typically, such sites are of high commercial value, e.g., news agencies, magazines, comic archives, and reducing their download times is critical to their success. Applying our scheme to these cases requires the following steps:

1. accumulation of servers statistics over a large time period,
2. offline redirection decisions and HTML rewriting,
3. periodic online statistics accumulation, and
4. decision (depending on the online statistics) whether to send the client the original HTML file or the rewritten one.

In other cases, our scheme can be a preprocessing step and should depend on the online conditions (e.g., whether the redirection spot is overloaded or not).

The contributions of this paper are as follows: First, we show that downloading a page in parallel from multiple servers (using URL rewriting) leads to increased performance in certain cases. Based on the observation, we incorporate the possibility of parallel downloads into a classic model used for replica placement. The resulting problem thus becomes: find the optimal placement of MOs and the optimal redirection decisions that minimize the overall download time. We tackle a special case of interest where there exists a central repository storing and indexing all the MOs. The MOs contained in a page are split into two sets, one to be downloaded from the original server and one from the repository. To achieve this, a model quantifying the expected download time is developed. Both the storage and processing capacity constraints influence redirection decisions. We then proceed by presenting an algorithm called REPD (Replication Enabled Parallel Downloads) to solve the combined replication/redirection problem. The algorithm is distributed in nature and is experimentally shown to outperform various alternatives.

The rest of the paper is organized as follows: Section 2 elaborates the system model used in our study. Section 3 formalizes the problem of minimizing the response time exhibited by a client as a constraint optimization problem.

- *T. Loukopoulos is with the Department of Computer Science, Hong Kong University of Science and Technology, Clearwater Bay, Hong Kong. E-mail: luke@cs.ust.hk.*
- *I. Ahmad is with the Department of Computer Science and Engineering, The University of Texas at Arlington, Box 19015, 416 Yates Street, Nedderman Hall, Arlington, TX 76019-0015. E-mail: iahmad@cse.uta.edu.*

Manuscript received 14 May 2002; revised 16 May 2003; accepted 16 Feb. 2004.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 116539.

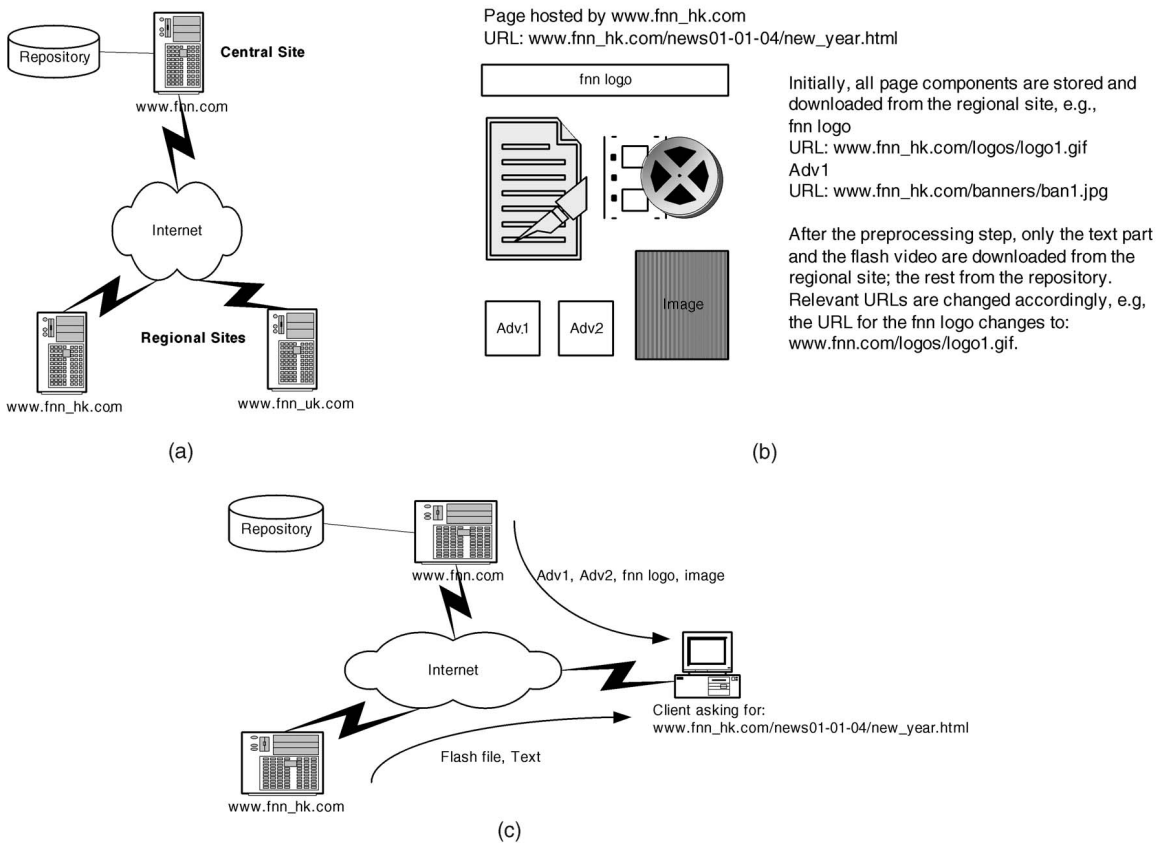


Fig. 1. The system model. (a) System organization, (b) preprocessing, and (c) parallel downloads.

Section 4 presents the proposed replication algorithms. Section 5 describes experimental results. Section 6 presents the related work reported in the literature. Finally, Section 7 provides some concluding observations.

2 System Model

The environment of interest consists of one central multimedia repository located at a company's headquarters and worldwide dispersed sites/servers hosting pages from the company's regional departments. Fig. 1a shows an example of a news agency where the regional sites provide storylines of local interest. We will use the term regional server to denote all the servers of the system other than the central one that supports the repository. The case of multiple repositories, as well as organizational schemes to support parallel downloads from multiple regional servers is out of the scope of the paper. However, the cost model and the algorithms proposed in Section 3 and 4, respectively, can be modified to support such extensions.

Pages hosted by a regional server contain MOs that are also stored at the repository. Therefore, it is possible to split the downloads of the embedded MOs between the regional and the central server. Fig. 1b depicts an example where all the MOs of a page are initially embedded using URLs referring to the regional site. In the preprocessing step, it is decided that some MOs should be served by the repository. Their URLs change accordingly, and a new copy of the HTML file is stored. The result of this process is shown at Fig. 1c where a client requesting the page receives its contents from both the regional and the central server in parallel.

Always downloading a page in parallel might not be the best solution since the workload of the servers could vary in a dynamic environment such as the Web. In order for our policy to be applicable in practice, regional servers should monitor the server dynamics of the repository and decide whether to satisfy requests for all MOs or use redirection. Examples of metrics that can be used is server workload (e.g., CPU utilization or opened connections), average service times, weighted past service times, etc. Such schemes were extensively studied in the context of server selection [11] where the problem is deciding which mirror server should satisfy an HTTP request. In this paper, we turn our focus on the implications parallel downloads have in replica placement decisions [14].

Typical replica placement formulations assume the objects initially undistributed and target in creating a distribution scheme (subject to various constraints) that optimizes certain criteria (e.g., average response time, network traffic). We consider the generic case where regional servers might not be able to store all MOs (due to storage capacity limitations) and/or might not be able to satisfy all requests without exceeding their processing capacity limitation. The target of the developed problem formulation is to minimize the average response time. Furthermore, the model tackles the case where MOs can be downloaded by following a link. We call such MOs optional to differentiate them from the embedded ones that are always presented to the user. Since optional MOs account for single object transfers, they cannot benefit from our scheme. However, including optional MOs

in the replica placement formulation is essential in order to tackle the case where an MO is embedded in some pages and optional in others.

3 Problem Formulation

In this section, we first discuss the response time parameters. We then use the resulting formulas to develop a cost model and define the problem as a two-objective optimization problem.

3.1 Response Time Parameters

We define the response time perceived by a user as the time difference between sending a request and receiving all the data required to present the Web page or, in the case of optional MOs, the data to present the single object. Let $S_1 \dots S_s$ be the s regional servers of the company and R be the server of the central repository. We assume that clients download all page components over a single persistent HTTP connection [17] and use the following simple formula to calculate the response time: Initial Overhead + Transfer Rate · Size of Objects.

A client's request for a page first results in opening a TCP connection to the server. Afterward, the actual HTTP request for the HTML file is sent, the server processes the request and responds with the file. We model the elapsed time between the client initiating a TCP connection and the server having finished the process of the HTTP request with $d_{S_i} \cdot RTT_{S_i}$, where RTT_{S_i} stands for the Round Trip Time between the client and the regional server S_i and d_{S_i} is a constant. A typical value for d_{S_i} is 2 (3/2 for establishing the TCP connection and 1/2 for sending the HTTP request). As the client's browser starts receiving the HTML file, it retrieves the URLs of the embedded objects and sends additional requests for them. We assume that this happens with the first data packet that arrives to the client (in practice, Web servers try to send the additional URLs needed as early as possible). With the use of persistent HTTP, these additional requests account for no latency since the client uses the previously opened TCP connection to route them.

In the general case, part of the MOs will be downloaded from the regional server and part from the repository. For the latter, a new TCP connection needs to be established, accounting for an additional latency of $d_{R,S_i} \cdot RTT_{R,S_i}$ (where d_{R,S_i} is a constant and RTT_{R,S_i} stands for the Round Trip Time between the client requesting the page from S_i and the repository). Thus, by denoting with $Ovhd(S_i)$ and $Ovhd(R, S_i)$ the initial overheads between the client issuing the Web page request and the first data packet being sent from the regional server and the repository, respectively, we can write:

$$Ovhd(S_i) = d_{S_i} RTT_{S_i} \quad (1)$$

$$Ovhd(R, S_i) = (d_{S_i} + 1/2) RTT_{S_i} + d_{R,S_i} RTT_{R,S_i}. \quad (2)$$

Notice that the 1/2 factor in (2) stands for the latency of sending to the client the first packet of the HTML file that includes the URLs of the embedded MOs. After gaining some insight about the cause of delays affecting response time, in the following section, we define a cost model to formulate the problem of selecting which MOs to download from the regional servers and which from the repository.

3.2 The Cost Model

Let $W_1 \dots W_n$ denote the company's pages and $M_1 \dots M_m$ the multimedia objects. Let $H_1 \dots H_n$ denote the HTML files, where H_j is related to only the W_j Web page. In case a page includes more than one HTML file, we treat the HTML parts of the page as one composite HTML file. Let $f(W_j)$ be the access frequency of W_j, measured in requests/sec.

Let U be an $n \times m(0,1)$ matrix such that $U_{jk} = 1$ iff M_k is embedded at W_j. A is an $s \times n(0,1)$ matrix such that $A_{ij} = 1$ iff W_j is hosted at S_i and 0 otherwise (page allocation matrix). We assume that a Web page is allocated to exactly one server and, if multiple copies of it exist, we treat each copy as a different page. We define X to be an $n \times m(0,1)$ matrix (download matrix) such that $X_{jk} = 1$ iff M_k should be downloaded from the regional server (obviously, either M_k is embedded at W_j or is an optional MO). Since each page is allocated to only one regional server, there is no need to define separate download matrices for each regional server. Let $B(S_i)$ be the average data transfer rate at which a request to S_i is satisfied and $B(R, S_i)$ the rate at which requests from the clients in the region of S_i are satisfied by the repository. Let $C(S_i), C(R)$, denote the S_i's and R's processing capacity, measured in HTTP requests/sec. We also refer to the storage capacity of S_i by using $Size(S_i)$. We use the same function to denote the size of H_j and M_k, all measured in bytes.

We can define the time it takes for Web page retrieval as the sum of the initial latency and the actual transfer time of MOs. Let $Time(S_i, W_j)$ and $Time(R, W_j)$ denote the time required to transfer the contents of W_j stored at S_i and R, respectively. We have:

$$Time(S_i, W_j) = Ovhd(S_i) + B(S_i) Size(H_j) + \sum_{k=1}^{m} X_{jk} U_{jk} B(S_i) Size(M_k) \quad (3)$$

$$Time(R, W_j) = Ovhd(R, S_i) + \sum_{k=1}^{m} (1 - X_{jk}) U_{jk} B(R, S_i) Size(M_k) \quad (4)$$

and, hence, the response time a user experiences, denoted as $Time(W_j)$, is given by:

$$Time(W_j) = \max\{Time(S_i, W_j), Time(R, W_j)\} \quad (5)$$

provided that at least one MO is downloaded from the repository.

Having retrieved page W_j, the user can request optional MOs. Let $N(W_j, M_k)$ be the average number of per user requests for M_k once W_j was downloaded. We define $Time(W_j, M)$ to be the total response time a user experiences when following the links of W_j to optional MOs. Computing $Time(W_j, M)$ is inherently difficult as it involves characterizing user behavior, e.g., a user can request all optional MOs at once (by opening multiple browser windows) or retrieve the MOs sequentially. We will use the aggregate response time of the latter case as our metric. For each optional MO download, a new TCP connection needs to be established, accounting for latency of either $Ovhd(S_i)$ (download from the regional server) or $Ovhd(R, S_i)$ (download from the repository). Summing up the above remarks, we end up with the following equation:

```
PARTITION (W_j)
    MOarr[] = {M_k|(U_jk = 1)}; /*MOarr stores the embedded objects of */
    Sort_by_Decreasing_Size (MOarr);
    LocalDownload[W_j] = Ovhd(S_i) + B(S_i)Size(H_j);
/*The time it takes for the local downloads. Initially, only the HTML file is downloaded*/
    RemoteDownload[W_j] = Ovhd(R, S_i);
/*The time it takes for the repository downloads. Initially, no objects are to be downloaded*/
    WHILE MOarr  NULL DO
        obj = Take Next Object from MOarr;
/*Add the time to download the object in both the repository and the local downloads*/
        RemoteDownload[W_j] += B(R, S_i)Size(obj);
        LocalDownload[W_j] += B(S_i)Size(obj);
        IF (RemoteDownload[W_j] < LocalDownload[W_j]) THEN
/*Downloading the object from the repository is more beneficial. Restore the time for local
downloads. */
            LocalDownload[W_j] -= B(S_i)Size(obj);
            Xjk = 0;
/*X is the (0,1) allocation matrix as defined in the cost model (Sec. 3)*/
        ELSE
/*Downloading the object from the regional server is more beneficial. Mark the object to be
stored locally.*/
            RemoteDownload[W_j] -= B(R, S_i)Size(obj);
            Xjk = 1;
        Delete_from_MOarr(obj);
    ENDWHILE
    Store the M_k's that have at least one non-zero entry in X matrix.
    Store all optional objects if local download is faster than the remote one.
```

Fig. 2. Pseudocode for MO allocation.

$$Time(W_j, M) = \sum_{k=1}^{m} \{N(W_j, M_k)[X_{jk}(Ovhd(S_i) + B(S_i) Size(M_k))+ + (1 - X_{jk})(Ovhd(R, S_i) + B(R, S_i) Size(M_k))]\}. \quad (6)$$

By taking into account the capacity of the servers as well as the storage space available, we can formulate the problem of minimizing the total response time users experience to be a two objective constrained optimization problem:

Assign $0, 1$ values to matrix X so as to minimize:

$$D_1 = \sum_{j=1}^{n} f(W_j) Time(W_j) \quad \wedge \quad D_2 = \sum_{j=1}^{n} f(W_j) Time(W_j, M) \quad (7)$$

subject to the following constraints:

$$\sum_{j=1}^{n} A_{ij} f(W_j)(1 + \sum_{k=1}^{m} X_{jk}(U_{jk} + N(W_j, M_k))) \leq C(S_i) \quad \forall (1 \leq i \leq s) \quad (8)$$

$$\sum_{j=1}^{n} f(W_j) \sum_{k=1}^{m} (1 - X_{jk})(U_{jk} + N(W_j, M_k)) \leq C(R) \quad (9)$$

$$\sum_{j=1}^{n} A_{ij} Size(H_j) + \sum_{k=1}^{m} Size(M_k) | (\exists W_j | A_{ij} X_{jk} = 1) \leq Size(S_i) \quad \forall (1 \leq i \leq s). \quad (10)$$

Equation (10) represents the storage capacity constraint for each regional server, while (8) and (9) represent the processing capacity constraint for the regional servers and the repository. By assigning α_1, α_2 positive weights to the target functions D_1 and D_2 of (7), respectively, we can restate the problem as a single target function constrained optimization one. Hence, we refer to the composite weighted target function by D. Reducing the relevant decision problem to the classical file allocation problem, which is known to be NP-complete [10], is straightforward. Therefore, the optimization problem cannot be solved to optimality and requires heuristics.

4 The REPD Algorithm

A centralized approach to solving the replication problem using mathematical programming would be inefficient for a widely distributed system. In our scheme, we let the regional servers decide which MOs should be stored and serviced locally, given the storage and processing capacity constraints. These distributed decisions may result in overloading the repository. In this case, an offloading negotiation mechanism between the repository and the regional servers takes place.

4.1 Description

Each regional server decides for each page which of the MOs to store locally. This is done by first sorting the MOs according to their size and then testing for each one in decreasing size order whether local downloading would result in smaller response time than downloading it from the repository. If the local download is more beneficial, then a copy of the object is kept and the expected response time of the Web page is updated accordingly. A description of the algorithm is given in Fig. 2.

```
RESTORE_STORAGE( )
    CandidateMOs = {Mk| (∃ Wj| Xjk =1)}; /*List of MOs marked for local storage*/
    UsedSpace = 0;
    ∀ (Mk∈ CandidateMOs)
        UsedSpace += Size(Mk);
     WHILE (UsedSpace > Size(Si)) DO
/*PART I: Find the most beneficial MO to deallocate*/
        ImpactMin =∞;   /*Stores the least impact found so far*/
        BestMO = -1;       /*Stores the index of the best candidate for deallocation*/
/*L1*/   ∀ (Mk∈ CandidateMOs)
            ImpactMO = 0;  /*The total impact of Mk*/
/*L2*/      ∀ (Wj| Xjk =1)
                ImpactPage = 0;          /*The impact Mk's deallocation has on Wj*/
                NewLocalDownload[Wj] = LocalDownload[Wj] - B(Si)Size(Mk);
                NewRemoteDownload[Wj] = RemoteDownload[Wj] + B(R, Si)Size(Mk);
                    /*Impact if the object is embedded*/
                ImpactPage += XjkUjk(max{NewRemoteDownload[Wj],
        NewLocalDownload[Wj]} -
                                                -
                                                max{RemoteDownload[Wj],LocalDownload[Wj]})
                                                ;
              /*Impact if the object is optional*/
                ImpactPage += N(Wj,Mk)[(Ovhd(R, Si)+B(R, Si)Size(Mk)) -
        (Ovhd(Si)+B(Si)Size(Mk))];
                ImpactPage *= f(Wj)/Size(Mk);
                ImpactMO += ImpactPage;
            IF (ImpactMO < ImpactMin) THEN
                BestMO = k;
/*PART II: Reoptimize page downloads*/
       /*For all pages where MBestMO was embedded and marked for local download*/
/*L3*/  ∀ (Wj | XjBestMOUjBestMO = 1)
       /*For all embedded MOs that are stored locally but were marked for remote download*/
/*L4*/      ∀ (Mk | Ujk = 1 ∧ Xjk = 0)
                IF (downloading Mk locally improves response time) THEN
                    Xjk = 1;
        Deallocate MBestMO;   /*Making necessary updates*/
        UsedSpace -= Size(MBestMO);
     ENDWHILE
```

Fig. 3. Pseudocode for restoring the storage constraint.

Let z denote the average number of embedded MOs per page. Page partition begins by sorting the embedded MOs which can be done in $O(zlogz)$. The WHILE loop removes one MO from MOarr in every iteration, thus it accounts for $O(z)$ complexity. Therefore, assuming w pages hosted on average at each regional server, the total complexity of the partitioning phase is $O(wzlogz)$.

Storing all the MOs the algorithm outputs may not be feasible due to storage or processing capacity constraints. To deal with this problem, we restore the storage capacity constraint by using a greedy method in which we evaluate the negative impact each MO deallocation causes in the target function D and remove the MO with the least negative effect. After each deallocation, we check whether we can further reduce the download time for pages previously marking the deallocated MO for a local download. This is performed by taking advantage of the fact that some MOs, although stored in the regional server, are not marked for local download in these pages since they would increase the response time. However, with the deallocation of an MO marked for a local download, it can be beneficiary to let previously unmarked MOs taking its place. If this is the case, we alter the object partitioning for these pages and continue to iterate the whole process until the storage constraint is no longer violated. The pseudocode of Fig. 3 summarizes the process.

The worst case of this phase comes when each page has m embedded objects and $Size(S_i) = 0$ which forces the while-loop to iterate until all MOs are deallocated. Part I of the algorithm runs in $O(mw)$ since it iterates for all MOs and all pages. The same holds true for Part II of the algorithm if all the embedded objects are marked for local downloads. Since Parts I and II are executed in every iteration of the while-loop and the number of iterations is m, we conclude that in the worst case, the deallocation phase runs in $O(m^2w)$.

Next, we analyze the uniform case where all MOs are of equal size, in each page there are, on average, z embedded MOs and each MO has the same probability of being embedded in a page. The above means that, on average, each MO is embedded in $l = wz/m$ pages. By assuming that $Ovhd(S_i)$, $Ovhd(R, S_i)$, and $Size(H_j)$ are comparatively small, we can approximate the ratio of MOs marked for local downloads using the transfer rate ratios: $r = B(S_i)/(B(S_i) + B(R, S_i))$. Since all MOs are of equal size, r also denotes the probability that, inside a page, an embedded MO is marked for local download. Therefore, the probability that an MO is marked for local download in at least

```
OFF_LOADING_REPOSITORY( )
    Collect_Status_Messages( );
    P(R) = ∑ P(Si, R);
    WHILE (P(R) > C(R)) DO
        L1 = {Si|(Space(Si)>0) ∧ (P(Si)>0)};
        L2 = {Si|(Space(Si)=0) ∧ (P(Si)>0)};
        L3 = {Si|(Si∉ L1) ∧ (Si∉ L2)};
        IF (L1 and L2 are empty) THEN
            BREAK;          /*CONSTRAINT CAN NOT BE RESTORED*/
        P(L1) = ∑ {P(Si)|(Si∈ L1)};
        P(L2) = ∑ {P(Si)|(Si∈ L2)};
        IF ((P(R)-C(R)) ≤ P(L1)) THEN
            ∀ (Si∈ L1)
                NewReq(Si) = P(Si)(P(R)-C(R))/P(L1);
                Send_Message(Si, NewReq(Si));
        ELSE
            ∀ (Si∈ L1)
                NewReq(Si) = P(Si);
                Send_Message(Si, NewReq(Si));
            ∀ (Si∈ L2)
                NewReq(Si) = P(Si)(P(R)-C(R)-P(L1))/P(L2);
                Send_Message(Si, NewReq(Si));
        Collect_Answers( );
        P(R) = ∑ P(Si,R);
    ENDWHILE
    ∀ Si
        Send_Message(Off_Loading_END);
```

Fig. 4. Pseudocode for the offloading phase.

one page is: $1-(1-r)l$. Thus, the storage capacity needed to store all necessary MOs is $m(1-(1-r)l)$. Let the available storage space be $c\,m(1-(1-r)l)$ with $c \in (0,1]$. The while-loop clearly performs $(1-c)m(1-(1-r)l)$ iterations. Let $l \geq 1$. L1 loop performs m iterations. L2 checks $rl = O(l)$ pages on average (since r is constant). Therefore, Part I runs in $O(ml)$. L3 is the same as L2 and iterates $O(l)$ times. L4 checks all MOs marked for remote download. The number of such objects inside a page is $z(1-r)$ on average. Thus, Part II runs in $O(zl)$ time. The total running time of the deallocation phase is $O(m(1-(1-r)\ l)(ml+zl))$. We distinguish two cases of interest. The first is the worse case where all pages embed all MOs (i.e., $z=m$). It follows from the definition of l that $l=w$ and the running time becomes $O(m(1-(1-r)w)\ (mw+mw) = O(m^2w)$ since $(1-r)w$ can be considered small. The worst-case result is identical to the one shown for the generic case, where MOs are not equal in size and are embedded with different probabilities. The second case of interest is when each MO is embedded in k pages (for a constant k). It follows that $l=k$ and that the running time becomes $O(m^2+mz)$ or $O(m^2)$.

For the processing capacity constraint restoration, we follow the same guideline, i.e., we check which (page, local MO) download pair would have the least decrease in performance if done from the repository and mark it accordingly. Again, we continue iterating until the constraint is met. If, through this process, an object is marked in all the pages to be retrieved from the repository, we deallocate it (further reducing the required storage space). The pseudocode of the algorithm resembles the one presented for restoring the storage constraint and is not included here.

Upon completion of the replication protocol, each S_i sends a status message to the repository. This message contains its free storage space $Space(S_i)$, the local processing capacity left $P(S_i)$ and an estimation for the workload that the current local assignment will impose to the repository $P(S_i, R)$. Having collected all the status messages, the repository checks if its estimated workload $P(R)$ will exceed its processing capacity $C(R)$. If this is the case, an offloading algorithm allocates the excess workload back to the regional servers. Servers that have both free storage and processing capacity available are considered first for allocating the extra workload, while those who have free processing power but have no storage space left are considered afterward. A description of the algorithm as pseudocode is presented in Fig. 4.

Upon receiving from the repository the extra workload to be added, every S_i assigns (W_j, M_k) more requests to be serviced locally. The criterion is the same as in the restoration of local processing capacity constraint, i.e., the (W_j, M_k) local downloads that result in the minimum increase of response time. As long as $S_i \in L_1$, all objects that are either already stored or whose storage would not result in capacity violation are considered until the new workload requirement is achieved or the storage capacity limit is reached. When $S_i \in L_2$, we explore the fact that some stored MOs might not be marked for local downloads in every page in which they are embedded (an MO is stored if it is marked for local download in at least one page). If the required workload level can still not be achieved, we check

if deallocating stored objects and allocating others can increase the workload of the regional server to the required level. This is done by first sorting all MOs in order of their total access frequency (i.e., the summation of the accesses in all the pages where they are referred). We afterward scan the array from left to right until we find an MO currently stored (this MO will have the least frequency among the ones stored). We then scan the array from right to left until we find an MO that is not stored. If the storage constraint is not violated, we replace the stored MO with the candidate one and proceed the scanning until either the required workload level is reached or the two scans cross each other.

Finally, if S_i still cannot serve $NewReq(S_i)$ additional requests, it sends a message to the repository with the number of additional requests it could satisfy and the fact that it now belongs to L3, i.e., not to be considered in the protocol. This can result in another round of message exchanges. The worst case for the number of message rounds occurs when servers are added to L3 one by one and is $O(s)$.

4.2 Alternatives

Here, we describe some additional algorithms that support parallel downloads. These algorithms are intuitive and serve as a yardstick in order to evaluate the performance of REPD in Section 5. The first algorithm stores the Most Frequently (MF) accessed MOs until the storage capacity is reached. Requests for MOs not hosted locally are satisfied from the repository, while the rest are served from the regional server. The regional server processing capacity constraint is removed by deallocating MOs, starting from the least frequent one until the required workload is reached. Note that, at the end of this process, the processing capacity constraint of the repository if violated cannot be restored. The reason is that the regional servers already store the most frequent MOs (up to their capacity) and, therefore, their workload cannot be further increased. MF exploits some parallelism, but not in a direct manner, since requests to the repository can only occur as a result of storage and processing capacity constraints. In case the constraints are not restrictive, the MF algorithm will store and download all MOs from the regional servers.

The second alternative is called MFPD (MF with Parallel Downloads). As implied by its name, MFPD is an extension of MF. It too allocates the most frequently accessed MOs at the regional servers with respect to the storage capacity constraint. Unlike MF though, MFPD splits the MOs in two sets: one to be transferred from the repository and one from the regional server, in a way similar to REPD. Note the difference between MFPD and REPD. REPD first optimizes downloads and then stores the MOs (using the deallocation phase), while MFPD first stores the most promising MOs and then optimizes the downloads (no deallocation phase is needed). In order to restore the processing capacity constraints, MFPD follows the same method with MF (i.e., it deallocates the least frequent MOs). As a consequence, MFPD is simpler compared to REPD and runs considerably faster. However, its performance is reduced, especially when the constraints are restrictive.

The final alternative is a local policy whereby everything is stored and retrieved from the regional servers (i.e., no parallelism occurs). Note that this policy assumes that regional servers have enough storage and processing capacity to satisfy all the requests.

TABLE 1
REPD Parameters

	RTT_A	RTT_B	d_A	$d_{B,A}$	B(A)	B(B,A)
LAN	110msec	480msec	2	4	200Kbps	150Kbps
Dial-up	240msec	640msec	2	2	8Kbps	7Kbps

5 Performance Evaluation

We performed two sets of experiments: The first was a practical scenario where we had two servers and one of them was acting as a repository. The second was a simulation setup where we evaluated the performance of REPD under various scenarios concerning the constraints and variables of the cost model.

5.1 Practical Scenario

We used two servers: one located at a technical institute in Greece (www.di.teilam.gr, designated from now on as A) and the other one at the Hong Kong University of Science and Technology (www.cs.ust.hk, hereafter referred to as B). Server A acted as a regional server, while B was the repository. We used two clients as our test cases. The first was located at an Internet Cafe with a 100 Mbps Ethernet LAN, while the second was behind a 52 Kbps dial-up. Both clients were geographically close to server A. The pages used in the experiment consisted of a 10 KB HTML part and images of 70 KB and 200 KB size (roughly). Pages were retrieved using Internet Explorer 6. We compared REPD against the Local alternative. For this reason, we had to define both the parameters of (1) and (2) and bandwidth values. The RTTs were derived using the ping utility. For the rest of the parameters, we used the following approach: we set $d_A = 2$ and recorded the average bandwidth when downloading a page with two 70 KB images from server A. We proceeded by assuming $d_{B,A} = 2$ and defining the bandwidth when downloading from B, using the same page. Afterward, we set one image to be downloaded from A and one from B, and recorded the completion time. Using the previously derived variables we updated the value of $d_{B,A}$. Table 1 summarizes the derived parameters (rounded).

We proceeded by testing the download time of a page containing 70 KB images as the number of the embedded objects increases. Figs. 5a and 5b show the results for the fast and slow clients, respectively (each case depicts 30 downloads).

For the fast client, we can observe that REPD achieves better performance in the majority of the test cases. The only case where REPD fails is when the number of images is five. This is due to the fact that the initial overhead of connecting to the repository out weights the benefits from parallelism. Fig. 5b illustrates that for a slow client, there is virtually no difference between REPD and the Local alternative. This is expected since the client's bandwidth acts as a bottleneck.

Next, we performed our tests using 200KB images. Fig. 6 demonstrates the results for the fast client. The download

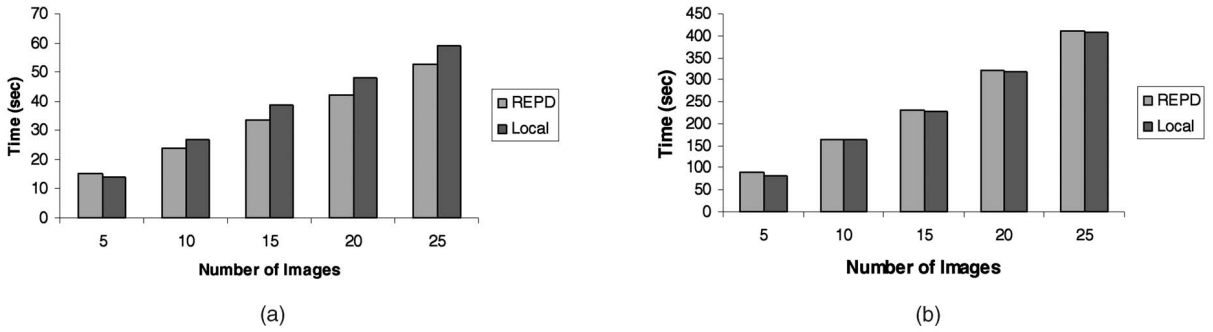


Fig. 5. (a) Download times for the fast client and (b) download times for the slow client.

times for the slow client showed no real difference between REPD and Local and are therefore not included.

Notice that gains from REPD are now more obvious. Even in the case of five images, REPD outperforms (by a narrow margin) the local alternative. Summing up the results of the first set of experiments, we can conclude that, provided the client is fast enough, REPD speeds up page download. In future work, we will evaluate its performance in the replica placement framework.

5.2 Simulation Results

We performed our simulation runs using a synthetic workload in the following manner. We generated pages with popularities following the Zipf distribution and HTML sizes following a heavy-tail one [3] with the average value being 30K. This is in accordance with previous work on characterizing the workload of Web servers (see [1], [5]). In our workload, roughly 10 percent of pages accounted for 60 percent of the requests. MOs were split into three categories according to their size (small 5K-50K, medium 50K-400K, large 400K-4M). Embedded MOs were assumed to be of small/medium size, while optional MOs were set as medium/large files.

The number of embedded MOs for a page was varied uniformly between 5 and 30, with 80 percent of MOs of small size. Ten percent of the pages in our experiments had links to optional MOs. The number of links was varied uniformly between 10 and 50 with 50 percent of MOs being of medium size, the rest of large. Unless otherwise stated, only 10 percent of users accessing a page with links to optional MOs request any of them. The number of optional MOs requested by an interested user was varied uniformly with the average value being 20 percent of the optional MOs referenced by the page. Page frequencies were derived with respect to the regional server processing capacity constraint and their popularity. Regional servers were assumed to have a processing power of 150 HTTP requests/sec. Unless otherwise stated, the repository was assumed to have enough processing capacity to satisfy all requests directed to it.

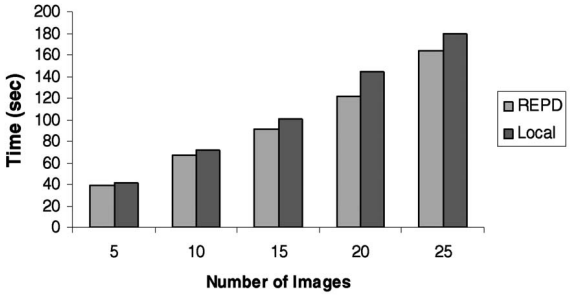


Fig. 6. Download times for the fast client.

RTTs between clients and regional servers followed a Gaussian distribution with a mean value of 50 msec and, between clients and the repository, a Gaussian distribution with mean value of 200 msec (requests arriving to a regional server mostly originate from clients with high proximity to it). All HTTP requests arriving to the repository were served using fixed data transfer rates taking values from a Gaussian distribution with minimum value 3Kbytes/sec, maximum 8Kbytes/sec, and mean 5KBytes/sec. The equivalent rates for the regional servers were decided proportionally depending on the experiment, e.g., if the proportion was fixed to 2:1, then the transfer rates were obtained from a Gaussian distribution with minimum value 6Kbytes/sec, maximum 16Kbytes/sec, and mean 10Kbytes/sec. We fixed the number of regional servers to 10. Each server hosted between 400-800 pages referring to 1,500-3,000 distinct MOs (both values varying uniformly). The total number of MOs in the network was 10,000. Each regional server received a total of 10,000 requests. We compared REPD with the three different alternatives presented in Section 4.2.

We conducted four experiments and recorded the average response time. Unless otherwise stated, the transfer rate proportion was 2:1 (regional servers:repository) and the weight parameters of the target function (7) were $(\alpha_1, \alpha_2) = (5, 1)$. The Local alternative was used as the basis for performance evaluation. Figs. 7, 8, 9, 10, 11, and 12 show the percentage of response time reduction each algorithm achieved compared to the Local alternative. In the first experiment (shown in Fig. 7), we tested how REPD performs as the storage capacities of the regional servers vary (100 percent storage capacity in the figures means that all MOs can be stored locally).

Fig. 7 shows that the performance of REPD and MFPD tends to remain constant when the available storage capacity is enough to hold the most beneficial (for REPD) or the most frequently accessed MOs (for MFPD) and drops significantly when the available storage space is reduced below a certain threshold (between 60 percent to 70 percent). It also demonstrates clearly the main merits of our approach

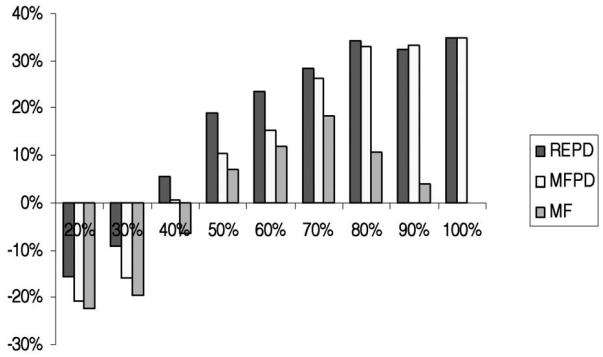


Fig. 7. Storage space ($C(S_i) = 100$ percent).

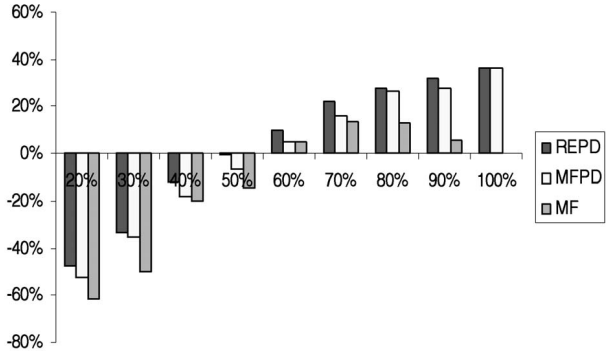


Fig. 8. Processing capacity at regional servers ($Size(S_i) = 100$ percent).

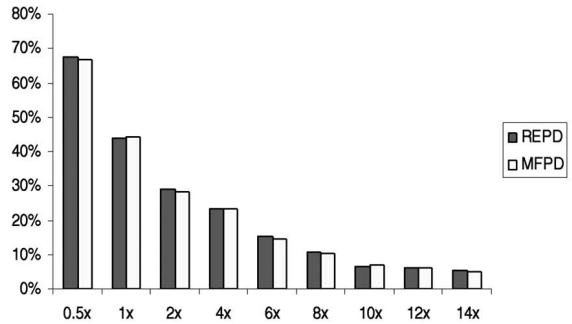
since REPD decreases the response time compared to the local alternative, even when only 40 percent of the storage space is available, while, at the same time, it maintains higher performance compared to the simpler MF/MFPD alternatives. The gains themselves are significant, e.g., with an available storage capacity of 50 percent, REPD results to an almost 20 percent decrease in the average response time. The same figure also shows that for low storage space values the response time increases compared to the Local alternative (negative values in the figures). This is expected since the Local alternative assumes no constraints.

Concerning the performance of MF, Fig. 7 illustrates that it initially increases as the storage space drops and then decreases to negative values. This somewhat unexpected result can be explained by considering that, at 100 percent storage space, MF behaves exactly as the Local alternative (i.e., stores and downloads everything from the regional server). As the available storage drops, MF redirects requests for the less frequent objects to the repository. This introduces some concurrency and results in gains in response time. As in REPD/MFPD, further reduction in the storage capacity results in performance decline, which is more acute in MF. This occurs because REPD and MFPD explicitly optimize the concurrent page download while MF accounts only for indirect parallelism.

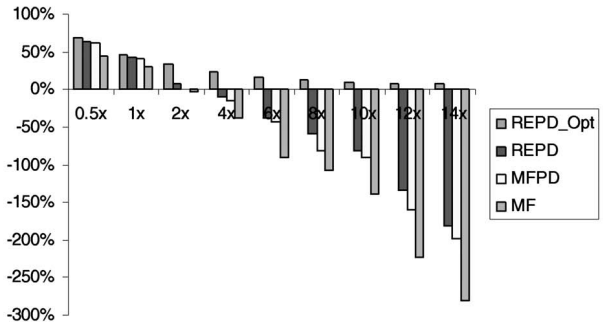
In the second experiment (Fig. 8), we study the effect of the processing capacity constraint at the regional servers. Fig. 8 shows REPD outperforming both MFPD and MF. For MF, the performance difference is not surprising, while, for MFPD, it deserves further explanation. Although MFPD optimizes the MO partitioning and downloading in the same way REPD does, when the processing capacity is not enough for all the selected MOs to be downloaded from the regional servers, MFPD assigns randomly a portion of the requests locally, shifting the rest to the repository, while REPD optimizes this assignment. Fig. 8 also shows that the performance of MF follows a trend similar to that of Fig. 7, i.e., increases initially and drops afterward. This is again due to the indirect download parallelism introduced when some of the MO requests are redirected to the repository.

Overall, the results of Fig. 7 and Fig. 8 are promising since the performance of REPD seriously reduces only in extremely adverse cases where storage and processing capacities are less than 60 percent of the required ones. Viewing these results from another perspective, we can say that REPD is expected to achieve comparable performance to the Local alternative using 40 percent less resources.

In the third experiment, we varied the transfer rate ratio up to 14:1, so as to test REPD as the margin of potential benefits from parallel downloads is lowered. Figs. 9a and 9b show the results when processing and storage capacities are fixed to (100 percent, 100 percent), and (60 percent, 60 percent). In the unconstrained case, shown in Fig. 9a, the performance of MF is identical to the Local alternative and is not included. In the other case, shown in Fig. 9b, we include, for comparison



(a)



(b)

Fig. 9. (a) Proportional transfer rate of regional servers ($C(S_i)$, $Size(S_i) = 100$ percent, 100 percent). (b) Proportional transfer rate of regional servers ($C(S_i)$, $Size(S_i) = 60$ percent, 60 percent).

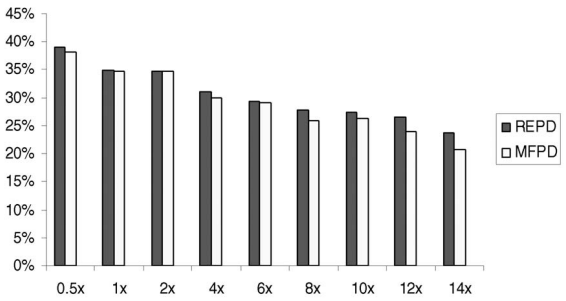


Fig. 10. Increasing transfer rate for fixed proportion of 2:1 (regional versus repository).

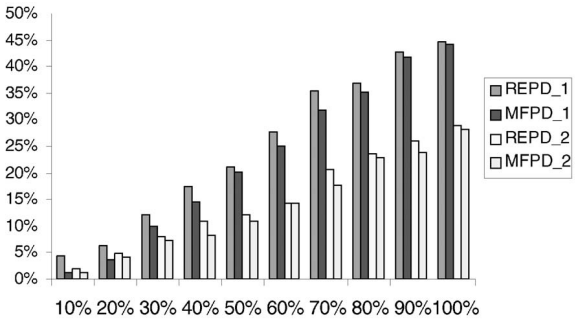


Fig. 12. Processing capacity of the repository (proportional transfer rate 1:1 and 2:1).

reasons, the unconstrained performance of REPD (designated with REPD_Opt).

Fig. 9a illustrates that, as the proportional transfer rate increases (i.e., downloading from the repository becomes more expensive), the performance of REPD and MFPD drops sharply. Both REPD and MFPD manage to achieve significant gains even when the ratio is large (more than 10 percent in the 8:1 case of Fig. 9a), while, in the most realistic spectra of rates between 0.5-2:1, REPD managed to exploit potential parallelization even in the adverse cases of Fig. 9b, achieving a reduction between 30 percent and 70 percent. Another observation is that the merits of REPD versus MFPD are only apparent when the storage or processing capacities are restricted. As a matter of fact, the slight performance difference in Fig. 9a is mostly due to fluctuations on the transfer rate during different runs.

Fig. 10 shows the performance of REPD/MFPD when the transfer rates of the regional servers and the repository increase, while their relative proportion remains fixed at 2:1. The figure shows a small, almost linear decrease in performance because, with the increase in actual transfer rates, the initial overhead becomes more important. Fig. 10 demonstrates that increasing the transfer rate to 70Kbytes/sec (and 35Kbytes/sec for the repository) results in the performance of REDP dropping from around 35 percent to 25 percent (the base transfer rate for the regional servers has a mean value of 5Kbytes/sec.).

In the fourth experiment, we tested two additional aspects that can affect REPD's performance. First, we examined the sensitivity of REPD toward requests for optional MOs. Fig. 11 shows the performance as the percentage of optional MO requests increase (a 100 percent value indicates that the 10 percent of the users that request optional MOs will ask for all MOs referenced by a page). We also tested different weight assignments of (7). The performance of REPD remains either constant or drops marginally as requests to optional MOs rise (depending on the weight assignment). We can observe that low weight ratios of 1-10:1 are less sensitive to the workload compared to larger ones (50-100:1). Nevertheless, the largest difference exhibited was no more than 5 percent (between REPD_1 and REPD_100 in the 80 percent case), which indicates that weight selection does not significantly affect the algorithm's performance.

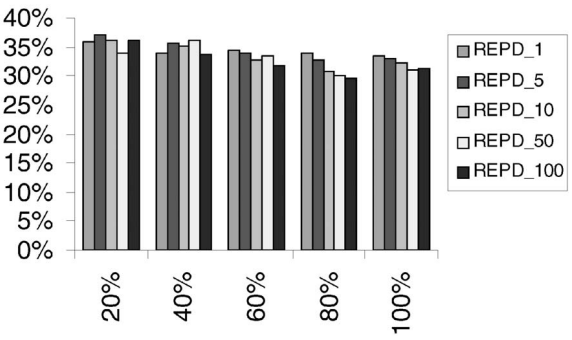


Fig. 11. Percentage of users requesting optional MOs ($\alpha_1 = 1, 5, 10, 50, 100, \alpha_2 = 1$).

Finally, we measured the performance of the algorithms by varying the processing capacity of the repository. We did this by fixing the processing capacity at the regional servers to 120 percent (i.e., each server could satisfy 20 percent more requests than the ones assigned by REPD) and by computing the number of requests REPD redirects to the repository. We assumed a percentage of them being satisfied by the repository and reassigned the rest back to the regional servers using the negotiation mechanism described in Section 4. In case the regional servers reach their processing capacities, we accordingly increase them in order to account for the remaining requests. For the MFPD, we assumed infinite processing capacity at the regional servers. Fig. 12 illustrates the results for transfer rate ratios of 1:1 and 2:1. We observe that the performance of the algorithms increases almost linearly to the processing capacity of the repository in both the cases. This, in turn, implies that small alterations of the repository's processing power (originated possibly by excessive workload) will not have a serious effect on the performance of the algorithms (e.g., the performance difference between the 100 percent and 80 percent case is around 5 percent).

Summarizing the experiments, the proposed scheme for concurrent downloads (REPD) was shown to significantly decrease the average page download time, compared to the naive approach of downloading all MOs from the regional servers. Furthermore, compared to a simpler policy (MFPD) that takes into account potential parallelism, but follows simpler allocation decisions, REPD was found to perform better, especially when resources are restricted.

6 Related Work and Comparison

There are two main issues in the replication of Web content. The first is deciding what to replicate where, also called the allocation or replica placement problem. The second is the design of a redirection method that allows a request to be satisfied by a server other than the one originally addressed to.

Various redirection schemes have been proposed in the relevant literature, some also being available as commercial products. IBM's Network Dispatcher [18] and CISCO's Local Director [8] map the domain name of the Web site to the IP of a multiplexing router that is placed in front of a server farm. Both schemes perform better when the servers of the farm are not geographically distributed. In [4] and [6], the DNS server returns the IPs of the servers holding the requested object. Server selection is performed by the client's DNS resolver after probing the candidate servers. Various methods for selecting among replicated servers were extensively discussed, for example, in [7] and [23], while scheduling algorithms varying from blind round-robin to more sophisticated feedback methods were developed by the research community [9]. Overall, the described redirection schemes account for at least the additional latency of forming an initial network connection before connecting to the required server. In our framework, redirection is performed by regional servers when sending the HTML file. Related to redirection is the proposal in [12] to build a global infrastructure that can return the distance between any pair of hosts.

With regard to allocation, much work has been done in the past in the context of file allocation. An extensive survey can be found in [10]. For a recent comprehensive summary, the interested reader is referred to [14]. Most of the existing models formulate this as an optimization problem, with the target function being either the overall network traffic, the client-replica distance, replica availability, or the response time. Solutions range from greedy approaches [20] to genetic algorithms [16]. The dynamic variation of the problem where replicas can be potentially created and deleted upon every request was studied, for instance, in [2], [21], and [24]. Here, we take advantage of the existing file allocation formulations in order to build a cost model that describes parallel downloads.

Prepartitioning of objects and URL rewriting are important aspects of our work. The idea of analyzing the page structure is also used in the HTTP_DRP [25] protocol, where, for each page, an index file is created and Uniform Resource Identifiers (URIs) are assigned to the embedded objects. Related to URL rewriting is the way media files are handled in the solutions offered by enterprises specializing in content management, e.g., NetApp [19]. A metafile (ASX) that includes the possible sources for downloading the requested media file is created. The client, instead of accessing the media file itself, accesses the metafile and is appropriately redirected. Incorporating such metafiles to our scheme is both straightforward and viable.

Perhaps, the works reported in [13] and [22] are the closest to our approach. In [22], the authors propose to transmit a file simultaneously from multiple mirror servers. Concurrency occurs on a per block basis. Our approach is complementary in that, while [22] aims at accelerating FTP transfers, we focus on HTTP transfers. Embedded objects are usually of small-medium size and, therefore, the benefits of block-parallel downloads are diminished. Furthermore, our scope differs since we do not assume mirror servers, but rather try to keep the stored (at the regional servers) objects to the minimum necessary. In [13], the authors investigate whether it is more beneficial to download all the page components from one server by opening multiple connections to it or from multiple servers. They conclude that the best strategy is to download everything from a single server. At first, their conclusion seems to contradict our policy. However, the strategy for parallel downloads considered in [13] was to transfer the HTML part of the page from one server and all the embedded objects from another.

7 Conclusions

In this paper, we proposed a replica placement/redirection scheme that benefits from parallel downloading the various multimedia objects contained in a page from a regional server and a central repository. To the best of our knowledge, this potential is not yet adequately addressed in the relevant literature. Performance measurements and simulation results indicate that our policy outperforms other alternatives under a wide range of varying parameters. The benefits of the proposed scheme are limited when the client's bandwidth acts as a bottleneck. However, this is not a major drawback, especially since client bandwidth continues to increase. Future work includes further performance measurements using different server pairs, as well as exploring parallel download techniques in a distributed server environment where each server can act as a potential repository. Another research direction is to integrate this work with some of the server selection schemes proposed in the literature.

References

[1] M.F. Arlitt and C.L. Williamson, "Internet Web Servers: Workload Characterization and Performance implications," *IEEE/ACM Trans. Networking*, vol. 5, no. 5, pp. 631-645, Oct. 1997.

[2] B. Awerbuch, Y. Bartal, and A. Fiat, "Optimally-Competitive Distributed File Allocation," *Proc. 25th ACM Symp. Theory of Computing (STOC)*, pp. 164-173, 1993.

[3] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *Proc. ACM SIGMETRICS '98*, pp. 151-160, July 1998.

[4] M. Beck and T. Moore, "The Internet-2 Distributed Storage Infrastructure Project: An Architecture for Internet Content Channels," *Proc. Third Int'l WWW Caching Workshop*, June 1998.

[5] A. Bestavros, "WWW Traffic Reduction and Load Balancing through Server-Based Caching," *IEEE Concurrency*, special issue on parallel and distributed technology, vol. 5, pp. 56-67, Jan.-Mar. 1997.

[6] S. Bhattacharjee, M. Ammar, E. Zegura, V. Shah, and Z. Fei, "Application-Layer Anycasting," *Proc. IEEE INFOCOM*, 1997.

[7] R. Carter and M. Crovella, "Server Selection Using Dynamic Path Characterization in Wide-Area Networks," *Proc. IEEE INFOCOM*, 1997.

[8] "CISCO Scaling the Internet Web Servers,"white paper at: http://www.cisco.com/warp/public/cc/cisco/mkt/scale/locald/tech/scale_wp.htm, 2004.

[9] M. Colajanni, P.S. Yu, and D.M. Dias, "Scheduling Algorithms for Distributed Web Servers," *Proc. 17th IEEE Int'l Conf. Distributed Computing Systems (ICDCS '97)*, May 1997.

[10] L.W. Dowdy and D.V. Foster, "Comparative Models of the File Assignment Problem," *ACM Computing Surveys*, vol. 14, no. 2, June 1982.

[11] M. Gullickson, C. Eichholz, A. Chervenak, and E. Zegura, "Using Experience to Guide Web Server Selection," *Multimedia Computing and Networking,* Jan. 1999.
[12] S. Jamin, C. Jin, Y. Jin, D. Riaz, Y. Shavitt, and L. Zhang, "On the Placement of Internet Instrumentation," *Proc. IEEE INFOCOM 2000,* Mar. 2000.
[13] J. Kangasharju, K.W. Ross, and J.W. Roberts, "Performance Evaluation of Redirection Schemes in Content Distribution Networks," *Proc. Fifth Int'l Web Caching and Content Delivery Workshop,* 2000.
[14] M. Karlsson, M. Mahalingam, and C. Karamanolis, "A Framework for Evaluating Replica Placement Algorithms," HPL Technical Report, HPL-2002-219, year?
[15] T. Loukopoulos and I. Ahmad, "Replicating the Contents of a WWW Multimedia Repository to Minimize Download Time," *Proc. Int'l Parallel and Distributed Processing Symp. '00,* May 2000.
[16] T. Loukopoulos and I. Ahmad, "Static and Adaptive Data Replication Algorithms for Fast Information Access in Large Distributed Systems," *Proc. 20th IEEE Int'l Conf. Distributed Computing Systems (ICDCS '00),* 2000.
[17] J. Mogul, "The Case for Persistent Connection HTTP," *Proc. ACM SIGCOMM '95,* pp. 299-313, Aug. 1995.
[18] "NetDispatcher: A TCP Connection Router,"IBM Research Technical Report RC 20853, July 1997.
[19] "Network Appliances' NetCache,"white paper at: http://www.netapp.com/products/level3/netcache/Webcache.html, 2004.
[20] L. Qiu, V. Padmanabhan, and G. Voelker, "On the Placement of Web Server Replicas," *Proc. IEEE INFOCOM 2001,* Apr. 2001.
[21] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal, "A Dynamic Object Replication and Migration Protocol for an Internet Hosting Service," *Proc. 19th IEEE International Conf. Distributed Computing Systems (ICDCS '99),* May 1999.
[22] P. Rodriguez, A. Kirpal, and E. Biersack, "Parallel-Access for Mirror Sites in the Internet," *Proc. IEEE INFOCOM 2000,* Mar. 2000.
[23] M. Sayal, Y. Breitbart, P. Scheuermann, and R. Vingralek, "Selection Algorithms for Replicated Web Servers," *Proc. Workshop Internet Server Performance,* June 1998.
[24] O. Wolfson, S. Jajodia, and Y. Huang, "An Adaptive Data Replication Algorithm," *ACM Trans. Database Systems (TODS),* vol. 22, no. 4, pp. 255-314, June 1997.
[25] "The HTTP Distribution and Replication Protocol," *Proc. World-Wide Web Consortium,* http://www.w3.org/TR/NOTE-drp, 2004.

Thanasis Loukopoulos received the Diploma in computer engineering and informatics from the University of Patras, Greece, in 1997. He was awarded the PhD degree in computer science by the Hong Kong University of Science and Technology (HKUST) in 2002. After receiving the PhD degree, he worked as a visiting scholar at HKUST. Currently, he is doing military service. His research areas of interest include content distribution networks, P2P systems, and Web services.

Ishfaq Ahmad received BSc degree in electrical engineering from the Univeristy of Engineering and Technology, Lahore, Pakistan, in 1985, and the MS degree in computer engineering and the PhD degree in computer science from Syracuse University, New York, in 1987 and 1992, respectively. His recent research focus has been on developing distributed multimedia systems, video compression techniques, scheduling and mapping algorithms for scalable architectures, heterogeneous computing systems, and Web management. His research work in these areas has been published in more than 140 technical papers in refereed journals and conferences, with best paper awards at Supercomputing '90, Supercomputing '91, and the 2001 International Conference on Parallel Processing. He is currently a full professor of computer science and engineering in the Computer Science and Engineering Department at the University of Texas at Arlington. Prior to joining UT Arlington, he was an associate professor in the Computer Science Department at HKUST in Hong Kong. At HKUST, he was also the director of the Multimedia Technology Research Center, an officially recognized research center that he conceived and built from scratch. The center commercialized several of its technologies to its industrial partners worldwide. He has been on the program committee of more than 50 international conferences and is an associate editor of *Cluster Computing*, *Journal of Parallel and Distributed Computing*, *IEEE Transactions on Circuits and Systems for Video Technology*, *IEEE Concurrency*, and *IEEE Distributed Systems Online*. At the University of Texas Arlington, he is the director of IRIS (Institute for Research in Security), where he and his colleagues are engaged in research for security and safety using sensors, multimedia, parallel computing, and other technologies. He is a senior member of the IEEE.