Replicating the Contents of a WWW Multimedia Repository to Minimize Download Time

Thanasis Loukopoulos and Ishfaq Ahmad

Department of Computer Science
The Hong Kong University of Science and Technology, Hong Kong

Abstract

Dynamic replication algorithms aim at allocating, migrating and deleting copies of an object over various Internet hosts, according to the access patterns exhibited on-line, so as to improve object proximity for the end-users and/or load-balance the servers. Most of the existing algorithms try to disseminate the objects of an entire Internet Service Provider (ISP), without taking into account the needs and characteristics of specific web sites with large commercial value. In this paper we tackle the replication problem in an Internet environment, inspired by the need of news agencies and other information providers to include in their pages multimedia content without incurring high access delays. We consider an environment that consists of a central multimedia repository and various physically dispersed sites. We propose a cost model to formalize the replication of multimedia objects located at the repository which can result in decreasing the download time. Taking into account implementation issues, such as the storage and processing capacity constraints, the proposed replication policy is evaluated and compared with alternatives including an ideal LRU caching scheme. Qualitative comparisons with the other replication schemes are reported as well.

1 Introduction

During page creation users can refer to distant sites holding large multimedia objects (MOs), without necessarily copying them locally. When a client accesses such a page, separate HTTP requests are issued towards the distant servers in order to download the MOs. Intuitively, we can exploit this ability of concurrent downloads in order to: i) save storage space by not copying all MOs locally, ii) achieve load balancing among the servers, iii) decrease the retrieval time. In this paper we propose an organizational scheme for a company running sites across the world when significant portion of multimedia data is shared among them. Shared MOs are stored in a central repository. Web pages containing objects from the repository are parsed locally and the MOs referenced by them are split into two sets, one to be downloaded from the local server and one from the repository. The target is to minimize the maximum retrieval time of these two parallel downloads with respect to the servers storage and processing capacity limitations.

The rest of the paper is organized as follows: in Section 2 we present the system model in more details, Section 3 formalizes the problem of minimizing the response time exhibited by a client as a constraint optimization one, while Section 4 presents the replication algorithms. Section 5 describes our experimental studies and Section 6 discusses the related work. Section 7 concludes the paper with a summary and future extensions of this research.

2 System Model

We consider an environment that consists of one central multimedia repository located at the company's headquarters and worldwide dispersed sites (local sites), hosting pages from the company's local departments. When a client requests a web page, the HTTP request is received by its local server which responds with the HTML document. The client's browser parses the document and new requests are made using the URLs of the objects contained in the page. These URLs may refer either to MOs stored in the local site or in the repository. Clearly, storing locally all MOs needed by the web pages can be infeasible due to storage capacity constraints. Caching of multimedia material contained in the most popular webpages seems a more promising solution but has its own shortcomings.

To understand why the current caching schemes are not able to efficiently tackle with a central multimedia repository environment, we should note that these policies consider the case of a central site whose pages are heavily accessed and try to minimize the network traffic and the latency experienced by users, by keeping the most popular set of pages as close to the clients as possible. Initial HTTP requests addressed to the central site are redirected towards servers with better proximity to the clients. Regardless of the part of the network where it occurs, see related work section, any redirection strategy accounts for additional latency. Since in our case clients send their requests for HTML documents to the local servers and not to the central site, the above latency can be avoided.

In contrast, replication policies do not suffer from redirection latencies while improving client-pages proximity. Dynamic replication algorithms recently proposed by the research community (see related work section), focus on either minimizing network traffic, or load-balancing the web servers. Our approach differs from the above by targeting at minimizing the average response time users perceive. To do this, we exploit the fact that a user would experience faster response time if one part of the objects was downloaded from the local server, while another one from the repository, in parallel. To our best knowledge this technique is not yet adequately addressed in the relevant literature of caching/replication over the Internet.

Upon creation or update of an HTML file in a local web site, the server parses the document and retrieves the URLs of multimedia content. Based on statistics collected, such as page access frequency, each local server decides for his web pages which MOs should be downloaded locally and which ones from the central repository. The above information is included in a reference database together with the position of the URLs in the HTML document. Upon the arrival of a request for an HTML document to a local server, the local server queries the reference database and replaces on the fly the remote URLs with the local ones. This is necessary to avoid having a client requesting from the repository an MO that should be downloaded from the local server. Assuming a fast indexing scheme for the reference database, the computational latency occurred due to querying and changing URLs on the fly is minimal compared to the network latency due to request redirection from the repository to the local site.

3 The Cost Model

Let $S_1 \ldots S_S$ be the *s* local servers of the company, *R* be the server of the central repository, $W_1 \ldots W_n$ the company's pages and $M_1 \ldots M_m$ the multimedia objects. Let $H_1 \ldots H_n$ denote the HTML documents, where H_j is related to W_j. In case a page includes more than one HTML document, we treat the HTML parts of the page as one composite HTML file. Let $f(W_j)$ be the access frequency of W_j during peak hours, measured in requests/sec. A page can explicitly require the download of a multimedia object (compulsory MO) or include a link to it. It is up to the client to decide whether to request any optional objects, (e.g., a music page with links to wav files).

Let *U* be an $n \times m$ (0,1) matrix such that $U_{jk} = 1$ iff M_k is compulsory for W_j and 0 otherwise. U' is an $n \times m$ matrix such that U'_{jk} denotes the probability that having downloaded W_j, will afterwards issue a request for the optional object M_k; in case $U_{jk} = 1$ (compulsory object), $U'_{jk} = 0$. *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 such that $X_{jk} = 1$ iff $U_{jk} = 1$ and once W_j is requested, M_k should be downloaded from the local server. X' is an extension of matrix *X* where an element X'_{jk} is 1 iff $X_{jk} = 1$ or M_k is optional for W_j and if requested (with probability $U'_{jk} \in (0, 1)$), it should be downloaded locally. Clearly in case $X'_{jk} = 1$, M_k must be stored at the S_i server for which $A_{ij} = 1$.

Let $B(S_i)$ be the average data transfer rate at which a request to S_i, is satisfied during peak hours 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 processing capacities of S_i and *R*, 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. Finally, we denote by $Ovhd(S_i)$ and $Ovhd(R, S_i)$ the average values of the two latencies, clients of S_i experience when sending an HTTP request to S_i and *R* respectively. Without going in many details these latencies are the summations of setting up a TCP/IP connection plus the time spent by S_i and R in order to process an HTTP request. We are now able to define the time it takes for a web page retrieval as the sum of the initial latency and the actual transfer time for both the objects requested locally and from the repository. 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. By using persistent connections [22] HTTP requests are pipelined over the same TCP/IP connection thus resulting to the following expressions:

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

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

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

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

Having retrieved a page W_j, the user can request any of the optional objects referenced by it. Let $f(W_j, M)$ be the average number of optional objects a user requests from page W_j over a time period of one second. We define $Time(W_j, M)$ to be the total response time a user experiences in the $N(W_j, M)$ requests. For each optional object download, a new TCP/IP connection needs to be established†, accounting for latency of either $Ovhd(S_i)$ (download from the local server) or $Ovhd(R, S_i)$ (download from the repository). Summing up the above remarks we end up with the following equation:

$$Time(W_j, M) = f(W_j, M) \sum_{k=1}^{m} \{U'_{jk}[X'_{jk}(Ovhd(S_i) + B(S_i)Size(M_k)) + \\ + (1 - X'_{jk})(Ovhd(R, S_i) + B(R, S_i)Size(M_k))]\} \quad (6)$$

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

Assign 0, 1 values at 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 main constraints:

$$\sum_{j=1}^{n} A_{ij} f(W_j)\left(1 + \sum_{k=1}^{m} X_{jk} + f(W_j, M) \sum_{k=1}^{m} U'_{jk}X'_{jk}\right) \le C(S_i) \quad \forall(1 \le i \le s) \quad (8)$$

$$\sum_{j=1}^{n} f(W_j)\left(\sum_{k=1}^{m} U_{jk}(1 - X_{jk}) + \sum_{k=1}^{m} U'_{jk}(1 - X'_{jk})\right) \le C(R) \quad (9)$$

$$\sum_{j=1}^{n} A_{ij}Size(H_j) + \sum_{k=1}^{m} \left\{Size(M_k) \middle| \exists W_j \;\; with \;\; (A_{ij} = 1) \wedge (X'_{jk} = 1)\right\} \le \\ \le Size(S_i) \quad \forall(1 \le i \le s) \quad (10)$$

Eq. 10 represents the storage capacity constraint for each site, while Eq. 8 and 9 the processing capacity constraint for the local sites and the repository respectively. By assigning α_1, α_2 positive weights to the target functions in Eq. 7 we can restate the problem as a single target function constrained optimization one. Hence, we refer to the composite weighted target function by *D*. We claim without providing detailed proof that the relevant decision problem is NP-complete (proof be reduction to the (0,1) Knapsack problem). We should note here that the above weight assignment has well defined natural meaning, since the retrieval time for a web page is more important than the time for downloading optional objects.

We assumed that local servers download objects using a constant average transfer rate. This assumption is closer to reality if the available bandwidth at the server's side acts as a bottleneck during peak hours. On the other hand, if the bottleneck is at another part of the network, e.g., clients using an 28.8 KBps modem, then the above assumption may not hold true. For this reason at the experimental evaluation section the transfer rates at which requests are serviced, vary significantly from the estimations used when deciding about replica creation and are distinct for each HTTP request. Another assumption made, is that the processing time for an HTTP request is constant. Since we assumed peak hours, i.e., almost fixed server utilization, the above approximation is realistic. In general, the introduced model aims at highlighting the efficiency of any replication policy that takes into account the properties of Eq. 5 (i.e., concurrent downloads) and provides a framework for constructing algorithms that balance the downloading times, without overloading any site or violating storage constraints. In the

†. We consider the general case where users won't request all the optional MOs at the same time so already opened TCP connections may be timed out. For this reason we also did not consider potential parallelism in downloading optional objects (Eq. 6).

following section we present such a policy.

4 The Algorithm

4.1 Motivation

A centralized approach to solving the replication problem using linear programming would be inefficient for a widely distributed system, since it incurs additional traffic for statistic collection and would also be computationally intensive. In our scheme, we let the local servers decide which MOs should be kept and downloaded by them, given the storage and processing capacity constraints. These distributed decisions may result in overloading the repository. In that case, an off-loading negotiation mechanism between the repository and the local servers takes place. The algorithm may be executed during off-peak hours and may be coupled with any of the dynamic scheduling policies proposed in the www literature (see related work). The reason is that allocation decisions made off-line using the past access patterns, may be inaccurate due to the dynamic nature of the Web, e.g., breaking news.

4.2 Description

Each local 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 as follows:

```
PARTITION ( $W_j$ )
    MOarr[] = $\{M_k | (U_{jk} = 1)\}$
/*MOarr stores the compulsory objects of $W_j$ */
    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 document should be downloaded*/
    RemoteDownload[ $W_j$ ] = $Ovhd(R, S_i)$ ;
/*The time it takes for the repository downloads. Initially, no objects are to be downloaded*/
    WHILE $MOarr \neq NULL$ 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[ $W_j$ ] -= $B(S_i)Size(obj)$ ;
            $X_{jk} = 0$ ;
/*X is the (0,1) allocation matrix as defined in the cost model (Sec. 3)*/
        ELSE
            RemoteDownload[ $W_j$ ] -= $B(R, S_i)Size(obj)$ ;
            $X_{jk} = 1$
        Delete_from_MOarr(obj);
    Store the $M_k$ 's that have at least one non-zero entry in X matrix.
    Store all optional objects.
```

Storing all the outputed by the above algorithm objects may not be feasible due to storage or processing capacity constraints. We restore the storage capacity constraint by using a greedy method, i.e., evaluating the negative impact each MO deallocation causes in the target function D and removing the MO with the least negative effect. After each deallocation we check whether we can 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 server may not be marked for a local download, since they may increase the retrieval time. After deallocating one object, the retrieval time of these pages may increase and marking the above MOs for local downloads can now reduce it. In such a case we alter the object partitioning for these pages and iterate the process until the storage constraint is no longer violated.

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 performed 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 as not to be downloaded locally, we deallocate it, further reducing the storage space required. A formal description of the above algorithms is not included here due to space limitations. We should note though, that the difference in D, which is our deallocation criterion, is amortized over the size of an object when we restore the storage constraint and over the difference between the new workload and the required one, when we restore the processing capacity constraint. This is done in order to make our criterion more judicious over large and frequently accessed objects.

Upon completion of the replication algorithm, 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 $C(R)$. In such case an off-loading algorithm allocates the excess workload back to the local servers. Servers that have both free storage and processing capacity available, are considered first for allocating the extra workload. A description of the algorithm in pseudocode follows:

```
OFF_LOADING_REPOSITORY()
  Collect_Status_Messages();
  $P(R) = \sum P(S_i, R)$;
  WHILE ($P(R) > C(R)$) DO
      $L_1 = \{S_i | (Space(S_i) > 0) \wedge (P(S_i) > 0)\}$;
      $L_2 = \{S_i | (Space(S_i) = 0) \wedge (P(S_i) > 0)\}$;
      $L_3 = \{S_i | (S_i \notin L_1) \wedge (S_i \notin L_2)\}$;
      IF (($L_1 = \emptyset$) $\wedge$ ($L_2 = \emptyset$) ) THEN
          BREAK;    /*CONSTRAINT CAN NOT BE RESTORED*/
      $P(L_1) = \sum \{P(S_i) | (S_i \in L_1)\}$ ;
      $P(L_2) = \sum \{P(S_i) | (S_i \in L_2)\}$ ;
      IF (($P(R) - C(R)) \le P(L_1)$) THEN
          $\forall (S_i \in L_1)$
              $NewReq(S_i) = P(S_i)(P(R) - C(R))/P(L_1)$ ;
              Send_Message($S_i$ , $NewReq(S_i)$);
      ELSE
          $\forall (S_i \in L_1)$
              $NewReq(S_i) = P(S_i)$ ;
              Send_Message($S_i$ , $NewReq(S_i)$);
          $\forall (S_i \in L_2)$
              $NewReq(S_i) = P(S_i)(P(R) - C(R) - P(L_1))/P(L_2)$;
              Send_Message($S_i$ , $NewReq(S_i)$);
      Collect_Answers();
      $P(R) = \sum P(S_i, R)$ ;
  ENDWHILE
  $\forall S_i$
      Send_Message(Off_Loading_END);
```

Upon receiving from the repository the extra workload to be added, every S_i assigns (W_j, M_k) more downloads to be serviced locally. The criterion to use, is the same as in the restoration of local processing capacity constraint, i.e., the (W_j, M_k) local downloads that result to the minimum increase of response time. We should note here that storing optional objects will have a positive effect, if as expected

$B(R, S_i) < B(S_i)$. As long as $S_i \in L_1$, all objects that are either stored already, or their 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 for some (W_j, M_k), even if M_k is already stored in the local server it is marked to be downloaded from the repository. If the required workload level still can not be achieved, we check if deallocating stored objects and allocating others can increase the workload of the server to the required level. Finally, if S_i still can't serve all the additional requests, it sends a message to the repository with the number it could satisfy and the fact that it now belongs to L_3, i.e., not to be considered in the algorithm. This can result in another phase of exchanging messages.

5 Experimental Evaluation

5.1 Workload

We performed our experiments using a synthetic workload with the below summarized attributes:

Table 1: Parameters used in experiments.

Parameter	Value
Number of Local Sites (LS)	10
Number of Web Pages per LS	400-800
Hot Pages (accounting for 60% of traffic)	10%
Number of Compulsory MOs per Page	5-45
Number of Optional MOs per Page (10% of pages have optional objects)	10-85
Number of MOs in the Network	15,000
Number of MOs in an LS	1,500-4,500
Small HTML size (35% of pages)	1K-6K
Medium HTML size (60% of pages)	6K-20K
Large HTML size (5% of pages)	20K-50K
Small MO size (30% of MOs)	40K-300K
Medium MO size (60% of MOs)	300K-800K
Large MO size (10% of MOs)	800K-4M
Number of Optional MOs requested per page.	30% of the total links in the page
Probability that a user will request one or more optional MOs.	10%
Processing Capacity of LS	150 HTTPreq./sec.
Processing Capacity of Repository	Infinite
Overhead at LS	1.275 -1.775 sec.
Overhead at Repository	1.975-2.475 sec.
Number of Page Requests per Server	10,000
(α_1, α_2)	(2, 1)

A number of previous papers on characterizing the workload of web servers [16], [23] report that a small percentage of pages accounted for a disproportionally large number of requests (hot pages). In our workload 10% of pages account for 60% of the requests. For the simulation we partitioned the HTML documents in small, medium and large sizes. We also did the same with MOs. Large size MOs represent small video clips while medium and small size objects represent audio and gif images. Since optional MOs are viewed only when the client explicitly requests them, we introduced a 10% probability that a user would be interested in viewing one or more of them. We also set the number of optional MOs an interested user requests, to 30% of the total optional objects referred in the page.

Concerning the server and communication link characteristics, we fixed the number of servers to 10 and their processing capacity to 150 HTTP requests/sec. The base value of the two overheads $Ovhd(S_i)$ and $Ovhd(R, S_i)$ was set to vary between 1.275 and 1.775 sec. for each local server and 1.975 - 2.475 sec. for the repository. These values represent estimations with the average processing time for HTTP requests at the local servers and the repository being 200 msec, the processing time overhead of our algorithm varying between 1 and 1.5 secs. and average RTTs (Round Trip Times) between clients and local servers/repository being set to 50/200 msec., respectively.

Every arriving HTTP request is served using a fixed data transfer rate which is defined for local servers from a uniform distribution taking values between 3 Kbytes/sec. and 10 Kbytes/sec. Requests arriving to the repository are satisfied at a fixed transfer rate taking values from a uniform distribution with minimum 0.3 Kbytes/sec. and maximum 2 Kbytes/sec. Clients geographically belonging to the same local server, experience the same transfer rate at their connections with the repository.

We generated 10,000 requests at each server. In order to simulate real life situations where the actual transfer rates and initial overheads differ from the estimations used when deciding about the object placement, 60% of the requests were satisfied from the local servers at a transfer rate ± 10 % within the estimation used; 30% were satisfied at a rate between 1/2 and 1/3rd of the initial estimation and the rest at a rate varying from 1/4th to 1/6th. The last 10% represents cases of network congestion. Since the repository's average transfer rate is much lower compared to the local servers, we decided to change it by only ± 20 % compared to the estimations used when deciding about the allocation. ± 20 % change was also used for the initial connection overheads of the repository. Finally, the overhead of a request to the local servers was varied by a -10% to a +50% factor. Overall, the rational behind decreasing significantly the performance when downloading objects from the local servers, while keeping the network parameters of the repository within a good range of the initial estimations is to test our policy when initial estimations lead to intensive replication, while the actual network attributes would require a more conservative approach.

5.2 Performance Evaluation

We compared our policy with 3 different policies. The first is a "download all from the repository policy" (Remote policy), the second is a "download all from the local servers" (Local policy) and the third is an ideal LRU caching/ redirection scheme with 0 redirection overhead. Constraints of Eq. 8, 9, 10 were not applied to both the Remote and the Local policy, while the LRU policy was subjected to only the constraint of Eq. 8.

In the first experiment, we relaxed the local site's processing capacity constraint and varied the available storage. We measured the average response times exhibited under all policies and reported their relative values compared to the results obtained by our policy when no constraints were imposed. Figure 1 presents the average results for 20 runs. We only plotted the LRU and our policy since only these are affected by the storage space of local sites. The remote policy resulted in 335% increased response time while the Local policy in 23.8%. It is clear from the figure that our policy outperforms the alternative LRU, with the performance differences being more significant when 100% storage space is available (the last tick-mark in the plot). At that point LRU's performance is comparable to the local policy and results in approximately 24% more retrieval time compared to our policy, which is optimized since no

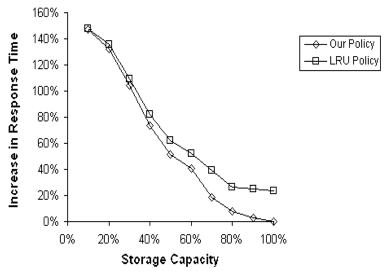

Figure 1: Plot of respose time versus local storage capacity.

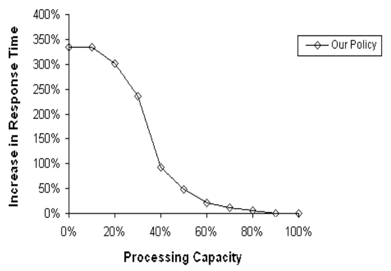

Figure 2: Plots of response time versus local processing capacities.

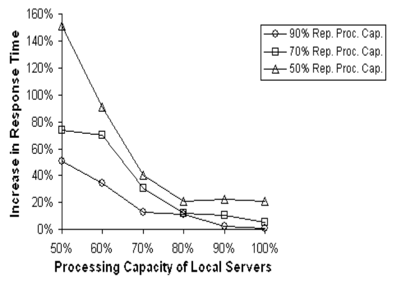

Figure 3: Plots of response time versus local processing capacities.

constraints are imposed. 100% storage with the synthetic workload used, would require 1,8 GB cache size on average, which is a reasonable demand. Another interpretation of Figure 1 is that our policy achieves the same response time with the Local and LRU, using around 65% of the capacity the other strategies need (the performance of our policy with 65% storage is almost the same as LRU with 100% and the local one). As the available storage decreases response times from our algorithm and LRU are comparable but are still much smaller compared to the Remote policy.

Figure 2 shows how our policy performs when the processing power of local sites varies while the storage capacity is 100%. The result is a double exponential curve. Response time towards the end of the curve is only marginally increased, since even with sites being able to support only 60% of the arriving requests (the rest going to the repository), the more traffic consuming objects were still able to be downloaded locally. However, reducing the processing capacity by less than 60% seems to have an ever increasing impact on response time, until the later becomes equal to the value of the remote policy (for 0% processing capacity).

Figure 3 shows the performance of our policy when central processing capacities are fixed to 90%, 70% and 50%. With local processing capacities of 70% and more, even in the case when the repository can only serve 50% of the requests, the response time of our policy is acceptable (around 40% more than the unconstrained one). On the other hand, when local capacities drop to 50%-60%, even in the less demanding case of 90% central capacity, the rise in response time is significant. This, in terms, means that local processing capacities affect the performance of our algorithm more than the repository's processing power.

Even though we set zero redirection overhead for LRU and Local policies, the gains from the proposed policy were substantial (outperforming both policies in most cases). The proposed policy performed well in all three experiments even when the network attributes (latency, transfer rate) significantly vary from the estimations used during allocation decisions.

6 Related Work and Comparison

There are two issues in replication of Web content. The first is deciding what to replicate where, also called the file allocation problem. The second is the design of the redirection method that allows a request to be satisfied by a server other than the one originally addressed to. Various redirection schemes were proposed in the relevant literature, some of which being also available as commercial products. IBM's Network Dispatcher [3] and CISCO's Local Director [2] 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 are better applied when the servers of the farm are not geographically distributed. Authors in [5] propose a policy that propagates information about the replication scheme in HTTP headers and thus, requires changes to both servers and clients. In the scheme proposed in [4] URLs refer actually to Java applets that incorporate knowledge about the redirection procedure and the current replica set. Although, this solution scales well and requires no changes to clients, it incurs the additional overhead of downloading and running the applet. In [6] and [7] the ISP's 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. The DNS infrastructure is also used for redirection in [1], only that this time server selection takes place in the site's DNS server. Various methods for selecting among replicated servers were extensively discussed for example in [9] and [10], while scheduling algorithms varying from blind round-robin to more sophisticated feedback methods, were developed by the research community [8]. Unfortunately, all DNS-based redirection methods possess the drawback that caching of DNS responses can degrade their performance by either making them to select a stale replica, [6], [7] or an overloaded server [1].

The need for an efficient redirection mechanism has been recognized by the WWW Consortium resulting in proposing the HTTP_DRP (Distribution and Replication Protocol) [11]. Among others, it introduces new functionalities with which a server is able to redirect a client elsewhere. HTTP_DRP can achieve per object replication, while in most of the above strategies the whole site contents should be kept by each server. The overhead though of an additional HTTP request can not be neglected. DistributedDirector by CISCO [1] and JetStream by WindDance [12] implement similar functionalities. Overall, the redirection schemes described, account for at least the additional latency of forming an initial network connection before connecting to the required server. Obviously, most strategies cause much more overhead than the establishment of one TCP connection, while some of them involve coarse grain replication and thus, are inappropriate for our purpose. In our policy redirection is performed at the server's side when sending the HTML document and includes only computational latency. Moreover, this latency is amortized over all the objects needed to be downloaded locally, while the other schemes need to redirect each HTTP GET request separately.

The approaches of [13], [14] and [15] focus on dynamic replication. In [13] the authors propose an algorithm called ADR that changes the replication scheme of an object in order to minimize the network traffic due to reads and updates. To do so they impose a logical tree structure for the network, which in practice, would require separate TCP connections for each node pair, accounting for significant overhead. The algorithm proposed in [14] load balances the workload among replicas. It burdens, however, routers with keeping track of the replicas. Authors in [15] propose a general scheme that creates, migrates and deletes replicas so as to improve the client's proximity to them without overloading any of the servers. The use of threshold values though, makes the performance of the scheme dependent upon their chosen values. In addition to this, it requires a rather high amount of messages to be exchanged between hosts, when replica creation/migration is needed.

The above approaches are not so suitable for a central multimedia repository environment because their scope is too broad and dynamic, targeting the entire set of objects moving inside an ISP. Their performance is highly dependent on the time period of algorithm execution. A small time period can result in creating replicas at one time slot only to delete them in the next one, while a large in changing the replication scheme too slowly to make the algorithms truly adaptive. These trade-offs and design problems seem unavoidable when dynamic replication is needed. In our system, static replication seems a better choice (see [21] for a discussion on the use of static vs. dynamic replication). Clients can access an MO either from the repository or from a local server. This reduces the difficulty of deciding where and how many replicas should be created, while central control can be maintained if needed. Prepartitioning of objects so as to maximize the benefits of concurrent downloads is another important aspect of our work. In the experimental section we show that this policy outperforms both the download-all-locally and an LRU network caching scheme even when the available bandwidth varied significantly from the estimation used when running the allocation algorithm. The idea of analyzing the page structure is also used in the HTTP_DRP [11] protocol. The target there though, is to create an index file for each page and assign a Uniform Resource Identifier (URI) to every object of the page, in order to download only the objects that have changed during page refreshment.

For the file allocation problem, substantial work has been done in the past. A thorough survey can be found in [17]. In the database field, file allocation was studied in [19]. The problem was studied for the case of a multimedia database [20]. A distributed policy was proposed to solve the problem [18]. Most works in this context assume decisions to be taken centrally [19], [20], focus on modelling the problem as a linear programming one [17], or lack details on how to implement the proposed algorithms over the Internet [18]. Our work benefits from the above results in building a file allocation-like cost model, but also proposes a decentralized replication algorithm that considers implementation details on where and how to perform redirection.

7 Conclusions

In this paper we proposed a replication/redirection scheme to increase the availability of a company's web pages, when they include heavy multimedia objects. Under a wide range of the network attributes, such as the latency and transfer rate, our policy outperformed in most cases an ideal LRU caching scheme.

References

[1] CISCO DistributedDirector. White paper at http://www.cisco.com/warp/public/cc/cisco/mkt/scale/distr/tech/dd_wp.htm.

[2] Scaling the Internet Web Servers. White paper at http://www.cisco.com/warp/public/cc/cisco/mkt/scale/locald/tech/scale_wp.htm

[3] NetDispatcher: A TCP Connection Router. White paper at ftp://ftp.software.ibm.com/software/network/dispatcher/whitepapers/research_tr.pdf.

[4] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson and D. Culler, "Using smart clients to build scalable services", *in 1997 Annual Technical Conference, USENIX*, Jan. 6-10, 1997, Anaheim, CA, pp. 105-117.

[5] M. Baentsch, L. Baum, G. Molter, S. Rothkugel and P. Sturm, "Enhancing the web infrastructure - from caching to replication", *IEEE Internet Computing*, pp. 18-27, Mar-Apr 1997.

[6] M. Beck and T. Moore, "The Internet-2 distributed storage infrastructure project: An architecture for internet content channels.", *in 3rd Int. WWW Caching Workshop*, Manchester, UK, June 1998.

[7] S. Bhattacharjee, M. Ammar, E. Zegura, V. Shah and Z. Fei, "Application-layer anycasting.", *in INFOCOM 1997*.

[8] 9. M. Colajanni, Ph. S. Yu and D. M. Dias, "Scheduling Algorithms for Distributed Web Servers.", *in Proc. 17th IEEE International Conf. On Distributed Computing Syatems*, May 1997.

[9] R. Carter and M. Crovella, "Server selection using dynamic path characterization in Wide-Area Networks", *in IEEE INFOCOM 1997*.

[10] M. Sayal, Y. Breitbart, P. Scheuermann and R. Vingralek, "Selection algorithms for replicated web servers.", *Workshop on Internet Server Performance*, June 1998.

[11] "The HTTP Distribution and Replication Protocol", WWW Consortium, at: http://www.w3.org/TR/NOTE-drp.

[12] WindDance Networks Corp. "JetStream" at http://www.winddancenet.com/jetstream/index.html.

[13] O. Wolfson, S. Jajodia and Y. Huang, "An Adaptive Data Replication Algorithm", *ACM Trans. On Database Systems (TODS)*, Vol. 22(4), June 1997, pp. 255-314.

[14] A. Heddaya and S. Mirdad, "WebWave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents.", *in Proc. 17th Intl Conf. On Distributed Computing Systems*.

[15] M. Rabinovich, I. Rabinovich, R. Rajaraman and A. Aggarwal, "A dynamic object replication and migration protocol for an Internet hosting service." *IEEE Int. Conf. on Distributed Computing Systems* , May 1999.

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

[17] L.W. Dowdy and D.V. Foster, "Comparative Models of the File Assignment problem", *ACM Computing Surveys*, Vol.14(2), June 1982.

[18] B. Awerbuch, Y. Bartal and A. Fiat, "Optimally-Competitive Distributed File allocation", *25th Annual ACM STOC*, Victoria, B.C., Canada, 1993, pp. 164-173.

[19] P.M.G. Apers, "Data Allocation in Distributed Database Systems," *ACM Trans. Database Systems*, 13(3), Sep. 1988, pp. 263-304.

[20] Y.K. Kwok, K. Karlapalem, I. Ahmad and N.M. Pun, "Design and Evaluation of Data Allocation Algorithms for Distributed Database Systems", *IEEE Journal on Sel. areas in Commun.(Special Issue on Distributed Multimedia Systems)*, Vol. 14, No. 7, pp. 1332-1348, Sept. 1996.

[21] M. Rabinovich, "Issues in Web Content Replication", *in Data Engineering Bulletin*, Invited Paper, Vol.21 No.4, Dec. 1998.

[22] J. Mogul, "The case for persistent connection HTTP.", *Proc. ACM SIGCOMM'95*, Aug. 1995, pp. 299-313.

[23] A. Bestavros, "WWW Traffic Reduction and Load Balancing through Server-based Caching", I*EEE Concurrency: Special Issue on Parallel and Distributed Technology*, Vol.5, pp. 56-67, Jan.-Mar. 1997.