Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Communications 28 (2005) 1062–1073
computer communications
www.elsevier.com/locate/comcom

Combining replica placement and caching techniques in content distribution networks

Spiridon Bakiras*, Thanasis Loukopoulos

Department of Computer Science, Hong Kong University of Science and Technology, Clearwater Bay, Kowloon, Hong Kong, China

Received 3 May 2004; revised 16 December 2004; accepted 10 January 2005
Available online 12 March 2005

Abstract

Caching and replication have emerged as the two primary techniques for reducing the delay experienced by end-users when downloading web pages. Even though these techniques may benefit from each other, previous research work tends to focus on either one of them separately. In particular, caching has been studied mostly in the context of proxy server systems, while replication is the technology behind Content Distribution Networks (CDNs). In this paper we investigate the potential performance gain by using a CDN server both as a replicator and as a proxy server. We develop an analytical model to quantify the benefit of each technique, under various system parameters, and propose a greedy algorithm to solve the combined caching and replica placement problem. Our simulation results indicate that a simple LRU caching scheme can improve significantly the response time of HTTP requests, when utilized over a replication-based infrastructure. Moreover, due to its simplicity, this hybrid approach does not affect the administrative overhead of the CDN architecture.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Replica placement; Caching; Content distribution network (CDN); Hybrid algorithm; LRU model

1. Introduction

The explosive growth of the World Wide Web and the increasing availability of fast Internet access to the end-user, have turned centralized web servers into a performance bottleneck. Popular web sites (e.g. news sites) receive millions of HTTP requests per day, which may easily overload a state-of-the-art web server and increase significantly the delay perceived by end-users.

Proxy caching was the first step towards reducing the latency of HTTP requests. It is realized by placing proxy servers in front of the clients, which store the most frequently accessed documents (this is commonly referred to as 'forward proxy', as opposed to 'reverse proxy' which is usually placed in front of the servers). User requests are forwarded to the proxy server, and only cache misses result in requests being forwarded to the web server. Proxy caching, however, has several disadvantages that limit its potential benefit: (i) the hit ratio reported in the literature is typically below 50% [1], and (ii) cache misses will normally incur a large delay, since these requests will have to be redirected to the origin server.

While caching tries to minimize the latency of downloading the most popular documents, the underlying principle of replication is to move the web content as close to the end-user as possible. Content distribution networks (CDNs), for example, accomplish that by replicating popular web sites across a number of geographically distributed servers. The key objectives of a CDN are to increase the availability of the hosted sites and, most importantly, to minimize the response time of HTTP requests.

Even though these two techniques may benefit from each other, previous research work tends to focus on either one of them separately. In particular, caching has been studied mostly in the context of proxy server systems (although [2] considers the case of caching at the server side, i.e. reverse proxy), while replication is the key technology of CDNs. However, implementing a caching scheme as part of a CDN architecture, may help overcome some of its limitations.

* Corresponding author. Tel.: +852 23586971; fax: +852 23581477.
E-mail addresses: sbakiras@cs.ust.hk (S. Bakiras), luke@cs.ust.hk (T. Loukopoulos).

0140-3664/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2005.01.012

For example, the study in [3] shows that the file popularity of a busy web server tends to follow a Zipf-like distribution with a parameter θ, which is much higher than the one observed in proxy server traces. Consequently, higher hit ratios may be achieved when caching is performed within a CDN system. Furthermore, cache misses may be redirected to a nearby CDN server, instead of the origin server, which can reduce the delay penalty of cache misses.

In this paper we investigate the potential performance gain by using a CDN server both as a replicator and as a proxy server. To the best of our knowledge this possibility was overlooked in the past. We develop an analytical model to quantify the benefit of each technique, under various system parameters, and propose a greedy algorithm to solve the combined caching and replica placement problem. Our simulation results indicate that a simple LRU caching scheme can improve significantly the response time of HTTP requests, when utilized over a replication-based infrastructure. Moreover, due to its simplicity, this hybrid approach does not affect the administrative overhead of the CDN architecture.

The remainder of the paper is organized as follows. In Section 2 we discuss the motivation behind our work, and also give a brief overview of previous research work on replica placement algorithms. In Section 3 we present the system model, and also state some assumptions regarding the CDN architecture. The proposed replica placement algorithm is introduced in Section 4, while the simulation results are illustrated in Section 5. A general discussion follows in Section 6, while Section 7 concludes our work.

2. Motivation and previous work

2.1. Motivation

A generic replication scheme works as follows: (i) the ‘objects’ to be replicated are defined, (ii) statistics are collected, (iii) based on some optimization criteria and constraints, replica placement is decided, and (iv) a redirection method is provided that sends client requests to the best replicator that can satisfy them. Regardless of the location, where redirection happens (DNS [4], or server level [5]), and the criteria with which a suitable replicator is selected [5,6], for each object an entry of the form <object_id,list_of_replicators> must be kept. Selecting objects to be single web pages will cause scalability problems, since updates need to be made whenever a new page is created, deleted or relocated. Therefore, the silent consensus in the papers dealing with replica placement is that objects are large, representing whole sites or large parts of them, e.g. whole directories. This is also indicated by the number of objects considered in the experiments, which is usually in the order of hundreds, e.g. [7] or thousands, e.g. [8–10]. Here, we adopt per site replication, meaning that either the whole content of a site is replicated, or none of it. However, our work is also applicable in the intermediate cases, where objects represent groups of pages.

Although creating site replicas helps on bringing the content closer to the clients, it does suffer from two drawbacks. First, the placement decisions should remain fairly static for a considerable time period. This is due to the fact that replica creation and migration incurs a high transfer cost. Second and foremost, the storage space is not used optimally. Ideally, we would require that the replicas of a page be proportional to their popularity (assuming the network parameters being otherwise equal). However, what we can only achieve is that pages are assigned replicas proportionally to the popularity of the site they belong. This is not efficient, since it is recorded that a relatively small set of pages within a site accounts for the largest number of requests [3].

In order to alleviate the above problems we decided to deploy caching in conjunction with replication. Caching operates on a per page level and is inherently dynamic. The intuition behind, is that by splitting the available storage space at each CDN server between replica placement and caching, we will end up with a network that stores sufficient site replicas, while keeping the most popular pages of all sites at the caches of the available servers. Deciding the percentage of storage space to devote in caching should not be an ad-hoc process. Therefore, we developed an analytical model that predicts the hit ratio of the LRU cache replacement scheme, given site access frequencies and the available storage capacity.

A recent study [11] addressed this problem from a different point of view. Motivated by the same observations, the authors proposed a few clustering techniques to efficiently group web pages into clusters. They also provided some heuristic algorithms for the cluster-based replica placement problem, and showed that clustering can improve considerably the performance, compared to coarse-grain (i.e. per site) replication. Their work, however, is orthogonal to ours, since it essentially deals with the problem of constructing clusters for efficient replication. As we mentioned earlier, our work is also applicable in the case of cluster-based replication.

Our primary contributions in this paper include the following: (i) we show that the hybrid distribution policy outperforms stand-alone replica placement, (ii) we show that the hybrid distribution policy outperforms pure caching, and (iii) we develop an analytical model to characterize the LRU cache replacement policy, which can be used independently.

2.2. Previous work

The implementation of a CDN service essentially involves three major design considerations: (i) replica placement, i.e. where and which documents to replicate, (ii) where to redirect a client request (i.e. which server), and (iii) who makes the redirection decision, e.g. client, server, DNS. In this paper we mainly focus on replica placement algorithms, but the reader may refer to [12,13] for more complete surveys on Internet data replication.

Models for replica placement date back to early 1970s under the context of the file allocation problem (FAP) [14] and received attention from diverse research areas, e.g. distributed databases [15], video servers [16], etc. [17] provides a thorough categorization of replica placement papers and the assumptions they use. An old survey of FAP formulations can be found in [18]. The basic form of the FAP is the following: given a network with N servers and M files exhibiting various read frequencies from each server, allocate replicas in order to optimize a performance parameter, subject to certain constraints. Usually, the resulting problem is (0,1) integer programming, NP-complete, and requires heuristics to solve.

In the context of CDNs, FAP-like formulations were used in [7,8,10,19–22], to name a few. The target functions considered in these papers include client-replica distance [10], read access cost [8,20], read and update cost [7,19,21], and replica availability [22]. Depending on the formulation various constraints were considered, e.g. server storage capacity [8,19,21], processing capacity [20], bandwidth [10], etc. Another distinguishing factor for the above papers is whether they tackle the dynamic version of the problem. [7,19,20] are works towards this direction. Given an input stream of requests, they alter replica distribution so as to minimize the total answering cost (potentially after each request). [20] also aims at balancing the load between the replicators. Load balancing is also the target of [23] with the assumption that the network has a tree structure, while it is also considered in [24], where the problem is replicating the contents of a single site.

Another option to formulate replica placement is by using the k-median problem [25], which can be briefly described as follows: given a graph with weights on the nodes representing number of requests, and lengths on the edges, place k servers on the nodes, in order to minimize the total network cost. The difference between k-median and FAP formulations, is that k-median decides about the replicas of a single object and, therefore, consecutive calls must be executed in order to distribute all objects. [9,10,26–28] are papers based on k-median formulations. In [27] the authors solve the problem to optimality for a tree network, using dynamic programming. [9] proposes a greedy heuristic that outperforms dynamic programming in non-tree networks, while [10] compares various heuristics and concludes that a greedy one that performs back tracking offers the better results. [26] provides heuristics specifically tailored for the Internet topology, while [28] studies placement in the case, where there exists no knowledge about the replication scheme of an object. Finally, [29] discusses systematic methods to aid in the process of choosing (among the various proposals that exist in the literature) the most suitable replica placement heuristic to implement in a specific environment.

To conclude, in a recent study [30], the authors raise the question whether replica placement algorithms are really important. Using extensive simulation experiments, they compare the majority of the replica placement algorithms from the literature, and conclude that a simple delayed-LRU caching scheme can perform at least as well as the best replica placement algorithms. They argue, however, that replica placement is still needed for other reasons, such as reliability and availability.

In this paper, we take advantage of the previous research work on replica placement, in order to build the model of Section 3. More specifically, we decided to use a FAP-like formulation with the target function representing read costs. Since our scope is large CDN providers, we also included storage capacity constraints. Our goal is not to propose a new replica placement scheme, but rather provide evidence that such schemes when coupled with caching, perform considerably better. Therefore, this work is complimentary with the above described efforts.

3. System model

Consider a generic CDN infrastructure consisting of N geographically distributed servers (Fig. 1). Let $S^{(i)}$, $s^{(i)}$ be the name and the total storage capacity (in bytes) of server i, where $1 \le i \le N$. The N servers of the system are interconnected through a communication network, and the communication cost between two servers $S^{(i)}$ and $S^{(j)}$, denoted by $C(i, j)$, is the cumulative cost of the shortest path between the two nodes (e.g. the total number of hops). We assume that the values of $C(i, j)$ are known a priori, and that $C(i, j) = C(j, i)$. As it will become apparent, the model does not depend on the link costs being symmetric or any other assumption concerning their values, as long as the relevant costs are known a priori. However, for reasons of simplicity in the simulation setup, we assume that link costs are symmetric.

Let there be M different web sites, named $\{O_1, O_2, \ldots, O_M\}$, that have subscribed to the CDN provider's hosting service. The size of site O_j, denoted by o_j, is also measured in bytes. Each site j consists of L_j distinct objects, named $\{O_{j1}, O_{j2}, \ldots, O_{jL_j}\}$, and the popularity of these objects follows the Zipf-like distribution with parameter θ_j.

The replication policy assumes the existence of one primary copy for each site in the network. Let SP_j be the site which holds the primary copy of O_j, i.e. the only copy in the network that cannot be deallocated, hence referred to as *primary site* of O_j. Each primary site SP_j contains information about the whole replication scheme of O_j. This can be done by maintaining a list of the servers that the jth site is replicated at, called from now on the *replicators* of O_j. Moreover, every server $S^{(i)}$ stores a two-field record for each site. The first field is the primary site SP_j of it, and the second the nearest server $SN_j^{(i)}$ of server i, which holds a replica of O_j. In other words, $SN_j^{(i)}$ is the server for which

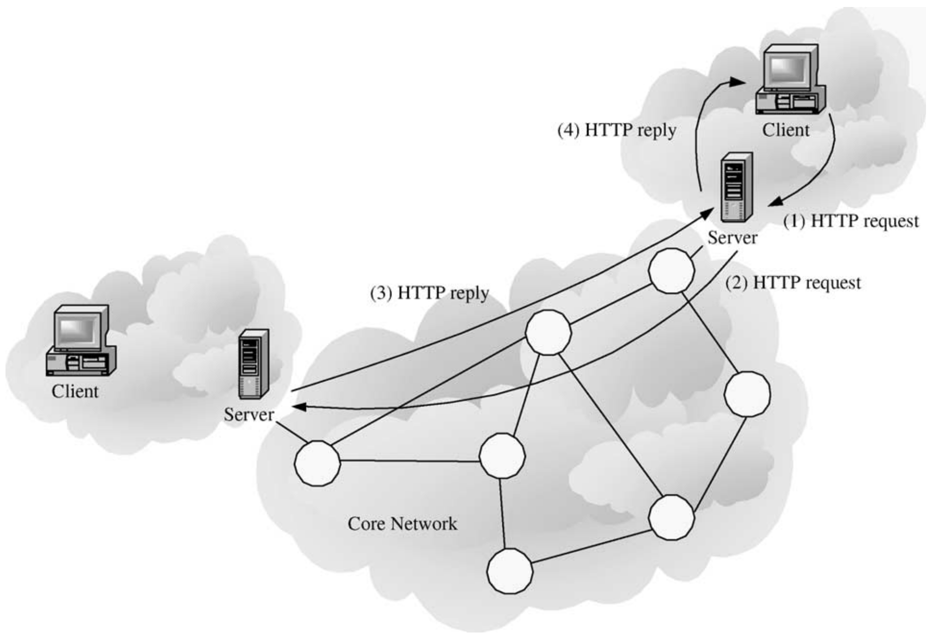


Fig. 1. The CDN architecture.

the requests from $S^{(i)}$ for O_j, if served there, would incur the minimum possible communication cost. It is possible that $SN_j^{(i)} = S^{(i)}$, if $S^{(i)}$ is a replicator of O_j. Another possibility is that $SN_j^{(i)} = SP_j$, if the primary site is the closest one holding a replica of O_j.

Finally, we assume that the storage capacity at each server can be used for both replication and caching. Consequently, the overall functionality of the CDN system may be summarized as follows. Whenever a client issues an HTTP request for one of the M hosted sites, the DNS resolver at the client side will reply with the IP address of the nearest, in terms of network distance, server (step (1) in Fig. 1). We will call this server a *first hop* server. The first hop server will act essentially as a proxy server, and if the requested document is neither replicated nor cached locally, it will redirect the client request to the appropriate server (i.e. the corresponding $SN_j^{(i)}$). The HTTP reply will be sent back to the CDN server, which in turn will forward it to the client, and also keep a copy in its local cache.

3.1. Problem formulation

Let $r_j^{(i)}$ be the number of requests for O_j, initiated from the client population behind $S^{(i)}$ during a certain time period. Our objective is to minimize the total cost, due to object transfer. Let $R_j^{(i)}$ denote the total cost due to $S^{(i)}$'s requests for site O_j, addressed to the nearest server $SN_j^{(i)}$. This cost is given by the following equation

$$R_j^{(i)} = [r_j^{(i)} - l_j^{(i)}]C\left(i, SN_j^{(i)}\right)$$

where $l_j^{(i)}$ is the number of requests that are satisfied locally by $S^{(i)}$. Notice, that if $SN_j^{(i)} = S^{(i)}$ (i.e. $S^{(i)}$ is a replicator of O_j), $r_j^{(i)} = l_j^{(i)}$ and $R_j^{(i)} = 0$. Otherwise, $l_j^{(i)}$ will represent the total number of requests served by the local cache. Therefore, the cumulative cost, denoted by D, is given by

$$D = \sum_{i=1}^{N} \sum_{j=1}^{M} R_j^{(i)}$$

Let us define an $N \times M$ replication matrix, named $\mathbf{X}$, with boolean elements. An element X_{ij} of this matrix will be equal to 1 if O_j is replicated at $S^{(i)}$, and 0 otherwise. Then, the replica placement problem may be formulated as follows

(1) Find the assignment of 0, 1 values at the $\mathbf{X}$ matrix that minimizes D.
(2) Subject to the storage capacity constraints

$$\sum_{j=1}^{M} X_{ij} o_k \leq s^{(i)}, \quad \forall\, i = 1, 2, \ldots, N$$

3.2. Cache hit ratio

In order to quantify the benefit of caching as part of a generic CDN architecture, we need an analytical model that can predict the achievable hit ratio under various system parameters. Assuming a simple LRU cache replacement policy, we derive, in the following paragraphs, an approximation for the cache hit ratio that can be achieved at a single CDN server for a specific web site.

Let us consider the general case of server $S^{(i)}$ and site O_j. The LRU cache may be modeled as a buffer that can store a finite number of objects B (Fig. 2). Since the object size for web documents is variable, B is approximated by $c^{(i)}/o_i$, where $c^{(i)}$ is the amount of storage space allocated for

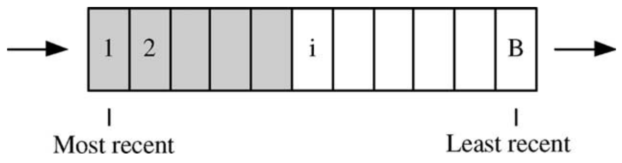


Fig. 2. The LRU buffer.

caching, and o_i is the average request size. When an object O_{jk} is first stored in the cache, it occupies the rear part of the buffer (i.e. it becomes the most recent one). If this object is not requested again for a long period of time, it moves gradually towards the front part of the buffer, and is eventually evicted from the cache after $K \geq B$ subsequent object requests. If, on the other hand, O_{jk} is requested before its eviction, it moves back to the rear of the buffer.

Assuming that each object is requested independently of the others, we calculate the steady-state probability that a specific object O_{jk} is present in the cache of server $S^{(i)}$. In steady-state, this object spends on average $\bar{h}$ time slots (i.e. request instants) inside the cache, followed by $\bar{m}$ time slots during which it is not present in the cache. These two time intervals may be calculated as follows

$$\bar{h} = \sum_{i=1}^{K}(i+\bar{h})p^{i-1}(1-p) + \sum_{i=K+1}^{\infty} Kp^{i-1}(1-p)$$
$$= \frac{p^{-K}-1}{1-p}$$

$$\bar{m} = \sum_{i=1}^{\infty} ip^{i-1}(1-p) = \frac{1}{1-p}$$

where p is the probability that O_{jk} is not requested.

Then, the steady-state probability $h_{jk}^{(i)}$ that O_{jk} is present in the LRU cache of sever $S^{(i)}$ is equal to

$$h_{jk}^{(i)} = \frac{\bar{h}}{\bar{h}+\bar{m}} = 1-p^K$$

which is essentially the probability that this object is requested at least once within K consecutive time slots. Therefore, the hit ratio for the whole site O_j is equal to

$$h_j^{(i)} = \sum_{k=1}^{L_j}\left[1-\left(1-p_j^{(i)}\frac{\alpha_j}{k^{\theta_j}}\right)^K\right]\frac{\alpha_j}{k^{\theta_j}} \qquad (1)$$

where $p_j^{(i)} = r_j^{(i)}/\sum_{k=1}^{M} r_k^{(i)}$ is the popularity of O_j at $S^{(i)}$, and α_j is the normalization factor for the Zipf-like distribution.

The only unknown variable in the above formula is K, i.e. the expected number of time slots that an object may spend in the cache before it is evicted, given that it is never requested. Consider the general case, where an object enters the cache at position 1, gradually moves towards the front of the buffer, and finally arrives at position B without ever being requested. Let us first determine what happens when the object is in a random place inside the buffer (e.g. position i in Fig. 2). During each time slot, it either stays at position i with probability p_i or moves to position $i+1$ with probability $1-p_i$, where p_i is the cumulative probability that one of the objects in positions 1 through $i-1$ is requested (i.e. the objects in the shaded part of the buffer in Fig. 2). Therefore, the expected number of time slots spent at each position i is equal to

$$t_i = \sum_{j=1}^{\infty} jp_i^{j-1}(1-p_i) = \frac{1}{1-p_i}$$

In order to approximate K, we make the following simplifying assumption. We assume that when the object in question has been pushed to the front of the buffer (i.e. at position B), all the previous positions are occupied by the $B-1$ most popular objects. Let p_B denote the cumulative probability that any one of these objects is requested at a given time slot. This probability may easily be calculated by sorting all the individual objects according to their popularity, and then selecting the top $B-1$ among them. Moreover, we assume that, while this object is pushed from position 1 towards position B, the popularity of every newly inserted object will be equal to $p_B/(B-1)$ (i.e. all the new objects will have identical popularity). Thus, K may be approximated as follows

$$K = \sum_{i=1}^{B} t_i = \sum_{i=1}^{B}\frac{1}{1-(i-1)\frac{p_B}{B-1}} \qquad (2)$$

3.3. Cache consistency and uncacheable documents

Before we continue, we should briefly discuss two issues that may affect the performance of a caching scheme. The first one is cache consistency, which tackles the problem of staleness in cached objects. Depending on the level of staleness allowed, consistency mechanisms fall in two categories: strong consistency (accessed copies are always up to date) and weak consistency (accessed copies might be stale). There has been an extensive amount of literature work on cache consistency mechanisms, so we do not address this problem in our present work. We assume that an appropriate consistency mechanism is implemented inside the CDN architecture, according to the specific policy of the CDN provider. However, we should point out two facts which are relevant to our work: (i) the stability of the CDN architecture (i.e. fixed number of servers and web sites) makes it easier to enforce strong consistency (e.g. through server-based invalidation [31]), and (ii) the study in [3] showed that the duration between successive modifications of an object is relatively large (between one and 24 h), hence the probability of requesting a stale object is very small.

The second issue is related to HTTP requests, which return uncacheable objects. URLs containing ‘cgi-bin’ or ‘?’ substrings, for instance, are considered as uncacheable at proxy servers. Furthermore, the content provider might explicitly prohibit certain pages, such as advertisement banners, from being cached. If these types of requests are frequent, they will affect the performance of the caching mechanism, since the value of $h_j^{(i)}$ in Eq. (1) will become an overestimation of the actual hit ratio. To overcome this problem, we assume that each web site O_j provides an estimation of the fraction λ_j of requests that return uncacheable documents. The values of λ_j can be calculated by analyzing the log files at the CDN servers. Then, the hit ratio $h_j^{(i)}$ may be adjusted by multiplying it with $(1-\lambda_j)$, i.e. the probability that the requested document is cacheable.

4. The hybrid algorithm

The replica placement problem with storage capacity constraints has been shown in many studies to be NP-complete (e.g. in [8,21]). Therefore, in this section we introduce a simple greedy algorithm to get an approximate solution for the combined caching and replica placement problem. The detailed pseudo-code is illustrated in Fig. 3, and the main flow of this program follows the greedy global approach that was adopted in other studies as well (e.g. [8,9,30]). In each iteration of the algorithm, all the server-site pairs are compared, and the one that produces the largest benefit value is selected for replication.

```
replicaPlacement (Input: $C(i,j), r_j^{(i)}, s^{(i)}$)
(1)  $D \leftarrow 0$;
(2)  for (all pairs $S^{(i)}$ and $O_j$)
(3)      $SN_j^{(i)} = SP_j$;
(4)      calculate $h_j^{(i)}$;
(5)      $D \leftarrow D + (1-h_j^{(i)})r_j^{(i)}C(i,SP_j)$;
(6)  while (true)
(7)      for (all pairs $S^{(i)}$ and $O_j$)
(8)          if (($X_{ij} = 0$) and ($s^{(i)} - o_j \geq 0$))
(9)              $b_{ij} \leftarrow (1-h_j^{(i)})r_j^{(i)}C(i,SN_j^{(i)})$;
(10)             for (all $O_k$)
(11)                 calculate $h_{k,new}^{(i)}$;
(12)                 $\Delta h \leftarrow h_k^{(i)} - h_{k,new}^{(i)}$;
(13)                 $b_{ij} \leftarrow b_{ij} - \Delta h r_k^{(i)} C(i,SN_k^{(i)})$;
(14)             for (all $S^{(k)} \neq S^{(i)}$ with $X_{kj} = 0$)
(15)                 $\Delta c \leftarrow C(S^{(k)},SN_j^{(k)}) - C(S^{(k)},S^{(i)})$;
(16)                 if ($\Delta c > 0$)
(17)                     $b_{ij} \leftarrow b_{ij} + \Delta c(1-h_j^{(k)})r_j^{(k)}$;
(18)     select pair $(S^{(i)},O_j)$ with max benefit $b_{ij}$;
(19)     $D \leftarrow D - b_{ij}$;
(20)     $X_{ij} \leftarrow 1$;
(21)     $s^{(i)} \leftarrow s^{(i)} - o_j$;
(22)     for (all $O_k$)
(23)         $h_k^{(i)} \leftarrow h_{k,new}^{(i)}$;
(24)     for (all $S^{(k)} \neq S^{(i)}$ with $X_{kj} = 0$)
(25)         update $SN_j^{(k)}$;
(26) return X, $D$;
```

Fig. 3. The hybrid algorithm.

The algorithm terminates when the benefit value is negative for all the server-site pairs or when all the servers have reached their storage capacity.

Lines 1–4 constitute the initialization part of the algorithm. In particular, it is assumed that all the storage capacity is allocated to caching, and the initial per site hit ratios, as well as the total initial cost, are calculated. The 'for' loop in lines 7–17 is the main part of the algorithm, where the benefit value b_{ij} for every server $S^{(i)}$ and site O_j is calculated. Specifically, line 9 is the local benefit for server $S^{(i)}$, while lines 14–17 take into consideration the relative benefit for any server $S^{(k)}$ for which $S^{(i)}$ is closer than the current $SN_j^{(k)}$. Furthermore, the benefit value is properly adjusted in lines 10–13, since the new replica will effectively reduce the hit ratios of all the non-replicated sites at $S^{(i)}$ (the LRU buffer size B will decrease). Finally, lines 19–25 perform some book-keeping operations to account for the new replica.

The complexity of this greedy algorithm is $O(RMN^2+RM^2N)$, where R is the total number of replicas created. For comparison reasons, the complexity of the typical greedy global algorithm [8] (with no caching) is $O(RMN^2)$. Notice, however, that we make the implicit assumption that the complexity of evaluating the hit ratio $h_{k,\text{ new}}^{(i)}$ in line 11, is $O(1)$. In the following paragraphs, we introduce some implementation details to justify the above assumption.

Let us consider first the approximation of K in Eq. (2), which essentially involves the sorting of L elements for the estimation of p_B. L is the total number of objects available for caching, i.e. all the objects for which the corresponding sites are not replicated. In the simulation experiments, though, we observed that calculating K during each iteration, produced the same result as in the case, where K was only calculated once at the initialization step of the algorithm (line 4 in Fig. 3). The intuitive explanation is that whenever the objects of a site O_j are removed from the sorted list, the popularity of the rest of the objects is increased accordingly, thus keeping the value of p_B at approximately the same level.

Having made this simplification, estimating the hit ratio from Eq. (1) is trivial. Notice, that $h_j^{(i)}$ depends only on the site popularity $p_j^{(i)}$ and the value of K. Then, the obvious solution to achieving the $O(1)$ complexity, is to pre-compute (off-line) the hit ratio of each site O_j under different values of $p_j^{(i)}$ and K. In the simulation experiments, the granularity of $p_j^{(i)}$ for the pre-computed values was set to 10^{-5}, while the granularity of K was set to 500 time slots.

5. Simulation experiments

In this section we investigate the impact of various replication and caching strategies on the response time of HTTP requests. Section 5.1 gives an overview of the simulation setup, while Section 5.2 presents the detailed simulation results.

5.1. Simulation setup

5.1.1. Network topology

We consider a CDN infrastructure consisting of $N=50$ servers, which is required to provide hosting service to $M=200$ web sites. Using the GT-ITM topology generator [32], we generated a random transit-stub graph with a total of 1560 nodes, and then placed each server and primary site inside a randomly selected stub domain. Finally, using Dijkstra's algorithm, we calculated the shortest path (in terms of number of hops) from each server $S^{(i)}$ towards every other server $S^{(k)}$ and primary site SP_j. Since the performance metric is the response time of HTTP requests, we set the propagation, queueing and processing delay inside the core network to be equal to 20 ms/hop. Notice that the maximum hop count between any pair of nodes is equal to 14, setting an upper bound of 280 ms for the end-to-end delay. In addition, we consider the case of homogeneous servers, i.e. all the servers have the same storage capacity s (given as a percentage of the cumulative size of all the web sites).

5.1.2. Datasets

Due to the absence of CDN traces in the public domain, we used the SURGE model [33] to generate a synthetic workload for our trace-driven simulation. In order to simplify the calculations, we used the same parameters θ_j and L_j for all the web sites, but we varied the total number of requests for each site in order to make the trace more realistic. Specifically, we generated 50 sites of low popularity (with 80,000 requests each), 100 sites of medium popularity (with 160,000 requests each), and 50 sites of high popularity (with 320,000 requests each). Furthermore, the popularity of each site O_j at server $S^{(i)}$ followed a normal distribution with mean $\mu=1/N$ and standard deviation $\sigma=1/4N$. However, we limited the possible values in the interval $\mu\pm3\sigma$. Table 1 summarizes the various parameters used in the simulation experiments.

5.2. Simulation results

In this section, we compare the performance, in terms of user-perceived latency, of the following four content delivery mechanisms

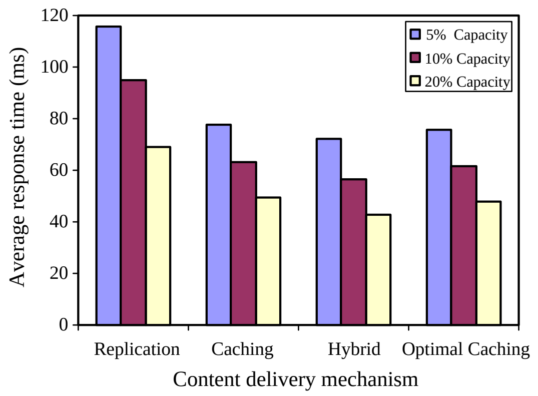
Table 1
Parameter settings

Parameter	Values
Number of servers (N)	50
Number of sites (M)	200
Web site popularity	Low, medium, high
Zipf parameter (θ)	1.0
Objects per site (L)	2000
Uncacheable requests (λ)	0, 10%
Site popularity at each server ($p_j^{(i)}$)	Normal$\sim$(1/N, 1/4N)
Storage capacity (s) (% of total)	5, 10, 20%
Network path delay (ms/hop)	20

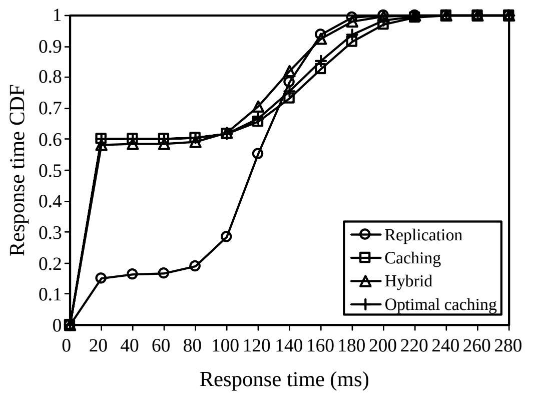
(1) *Replication*. This is the stand-alone replica placement algorithm, using the greedy global approach [8].
(2) *Caching*. All the storage capacity at the servers is allocated to caching.
(3) *Hybrid*. This is the combined caching and replica placement algorithm introduced in Fig. 3.
(4) *Optimal caching*. This policy corresponds to the case where all the servers have full and consistent knowledge about the cache contents of all the other servers in the network. Therefore, cache misses are always redirected to the nearest copy (either a CDN server or the primary site), i.e. we assume an ideal cooperative caching environment [34].

In the first experiment, we consider the case, where all the requested objects are allowed to be cached at the CDN servers (i.e. $\lambda=0$). These results will give us an indication of the 'upper-bound' on the performance of the pure caching scheme. More specifically, Fig. 4(a) shows the average latency per request for different content delivery mechanisms, under various storage capacity constraints. The hybrid approach clearly outperforms the stand-alone replication strategy, and reduces the average response time by approximately 40%. Compared to the pure caching technique, the performance difference is not that large, but savings up to 13% may still be achieved for large storage capacities. Finally, the optimal cooperative caching scheme has very similar performance to the non-cooperative version, which is an anticipated result, since cooperative caching is expected to be effective only among proxy servers that are physically close to each other.

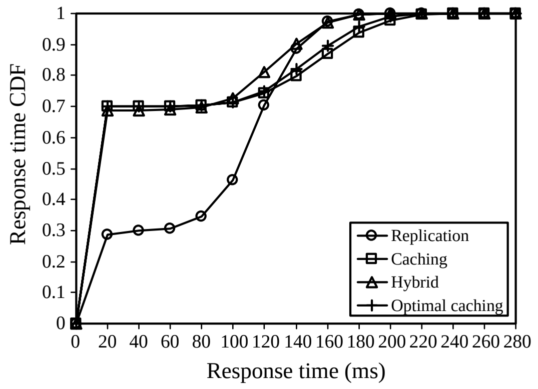
Fig. 4(b)–(d) give another perspective of the relative performance of the various mechanisms. Specifically, these figures depict the cumulative distribution function (CDF) of the response time, i.e. the percentage of requests that are satisfied within a certain time period. From these graphs we can actually get a clear picture of the potentials and limitations of each technique. First, the effect of pure replication is to distribute very normally the user-perceived latency. The majority of client requests experience the average response time, and only a small percentage of the requests receive better or worse service. Caching, on the other hand, produces a more 'heavy-tailed' distribution for the response time. In particular, a large fraction of the requests are satisfied locally at the CDN servers (the 20 ms value corresponds to requests being satisfied at the *first hop* servers), while a significant amount of client requests experience relatively large delays. Finally, the hybrid approach is able to offer the best performance overall. It has a high hit ratio at the first-hop servers, but also avoids excessive delays, by placing the right amount of replicas inside the network. Consequently, the CDF curve of the hybrid policy is a combination of the previous two: it initially follows the curve of the caching scheme for small delays, and later coincides with the curve of the replication scheme for larger delays.



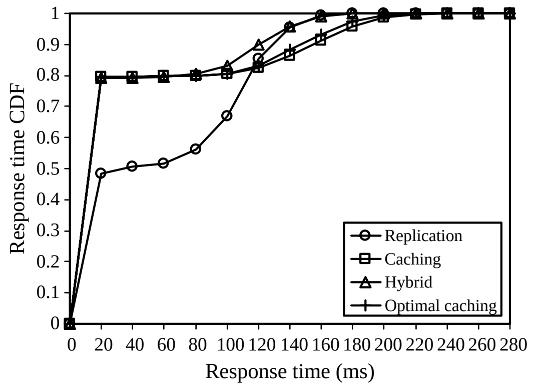
(a) Average response time.



(b) Response time CDF (5% capacity).



(c) Response time CDF (10% capacity).



(d) Response time CDF (20% capacity).

Fig. 4. Performance comparison of different content delivery mechanisms ($\lambda = 0$).

The second experiment investigates the performance of the various mechanisms under a more realistic environment. Specifically, we consider the case, where 10% of the requests involve uncacheable objects. The detailed results are illustrated in Fig. 5, and may be summarized as follows. The hybrid approach again outperforms both the caching and replication counterparts. As expected, the performance gain against pure replication is decreased, but it is still in the order of 30%. However, the performance of the two caching policies is degraded, and the hybrid scheme results in savings of up to 22% in user-perceived latency. Similar observations hold for the CDF of the response time. In general, the greedy algorithm predicts very accurately the relative benefit of caching and replication, and is thus able to make the correct replica placement decisions, in order to maximize the overall performance.

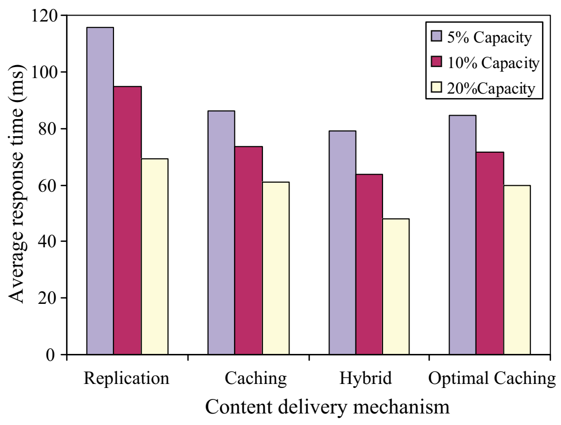
In our next experiment, we compare the performance of the greedy algorithm (Fig. 3) against a few ad-hoc hybrid approaches. In particular, we try to answer the following question: 'what if we allocate a fixed percentage of the storage space to caching, and run the standard replication algorithm for the remaining part of the storage space?' We tested three different versions, namely, for a cache percentage of 20, 50 and 80%, and the corresponding CDF plots for various parameters are shown in Fig. 6. The main conclusion that can be drawn, is that the ad-hoc approach is not very effective. The results for the different versions vary according to the storage capacity and the percentage of uncacheable objects. The greedy algorithm, on the other hand, performs at least as well as any of the ad-hoc schemes, under any system parameters. We should note, that the percentage of storage space allocated to caching by the greedy algorithm, ranged between 30–100% at different CDN servers.

Finally, Fig. 7 illustrates the accuracy of the analytical model for the hit ratio of the LRU cache (Section 3.2). It shows the predicted cost (in number of hops) per request that is returned by the greedy algorithm vs. the actual cost obtained by the trace-driven simulation. The results are very promising, and indicate that the proposed model can provide a very accurate approximation of the achievable hit ratio at different CDN servers. It tends to slightly overestimate the total cost, especially for large buffer sizes, but the overall error is less than 7%.

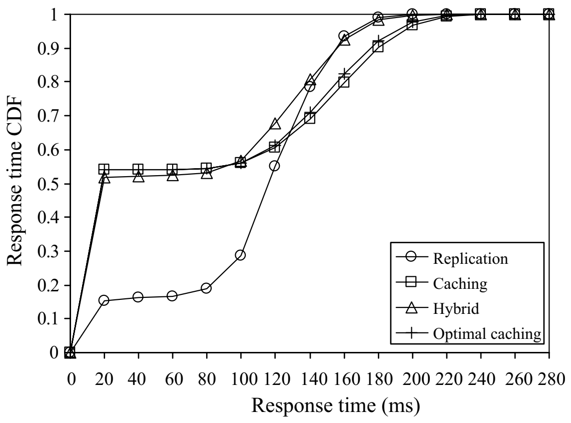
6. Discussion

6.1. Summary of experiments

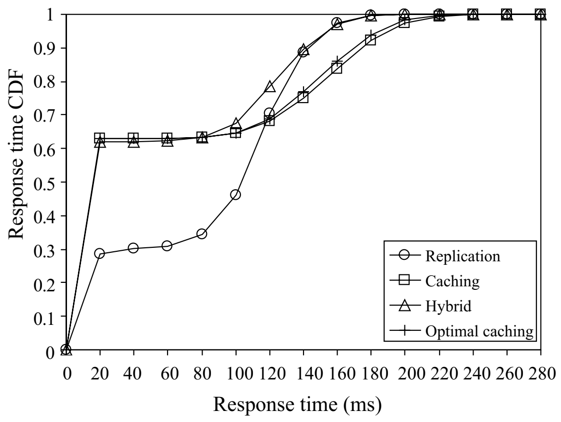
The overall conclusion is that combining caching and replica placement mechanisms yields significantly better



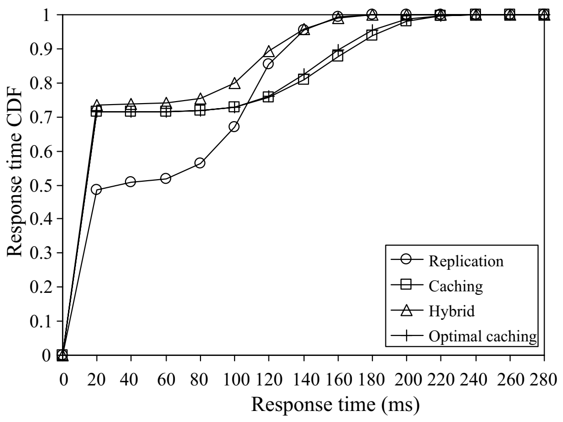
(a) Average response time.



(b) Response time CDF (5% capacity).



(c) Response time CDF (10% capacity).



(d) Response time CDF (20% capacity).

Fig. 5. Performance comparison of different content delivery mechanisms ($\lambda=0.1$).

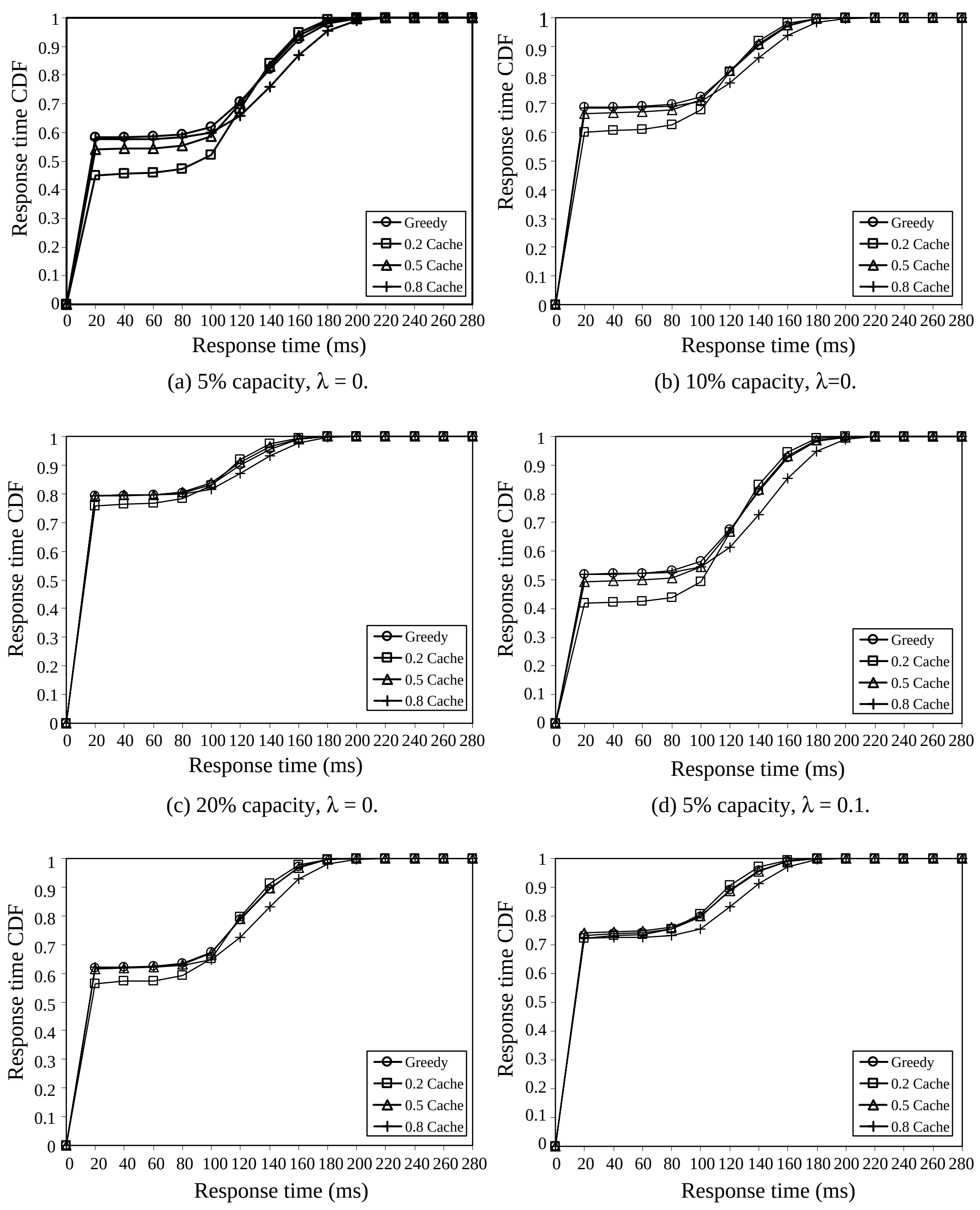
performance, compared to the stand-alone versions. This performance improvement is due to the following two facts: (i) a sufficient number of replicas are stored inside the network, so that the maximum delay is bounded ([10] also showed that increasing the number of replicas yields no significant performance improvement, beyond a small number of copies), and (ii) the most popular pages from all the available sites are stored locally at each server, so that a large percentage of requests do not need to be redirected.

Another advantage of the hybrid approach is the low administrative overhead for the CDN system. Specifically, the caching part operates locally in a completely decentralized manner, while the per site replication approach is very scalable, and easy to maintain in terms of the redirection mechanisms. We have shown that against a per-site replication scheme, the client-perceived latency can be reduced considerably (30–40%). Judging from the fact that the hybrid scheme also outperforms pure caching, even by a narrow margin (5–10%), we expect that against a per-cluster replication scheme, hybrid will be again the winner with the latency reduction varying in between the per-site and the caching case (depending on how the clusters are constructed). Proving the validity of the above claim is left for future work.

6.2. Limitations and extensions

A potential limitation of our work, concerns the scope of the replica placement model used in Section 3, which essentially aims at minimizing the client-object distance given the storage capacity constraints. Obviously, other system parameters (e.g. available bandwidth) and performance goals (e.g. used bandwidth, client-perceived response time) might have to be included. However, applying and evaluating our framework for the major replica placement problem (RPP) formulations that exist in the literature can not be done within the limits of a single paper, and it would have diverted the primary focus of our work, which is to demonstrate that a hybrid caching-replication scheme outperforms pure replica placement.

Nevertheless, we would like to point out that our model is extensible, provided that the impact of a single page placement on the constraints and the performance metrics, can be properly characterized (usually from the pure-RPP formulation). The hybrid problem can be solved by incorporating these changes to the replica allocation part of the algorithm in Fig. 3 (lines 7–17). For instance, assume that bandwidth was also a constraint. Then, for each server *i* we could define



(a) 5% capacity, λ = 0.

(b) 10% capacity, λ=0.

(c) 20% capacity, λ = 0.

(d) 5% capacity, λ = 0.1.

(e) 10% capacity, λ = 0.1.

(f) 20% capacity, λ = 0.1.

Fig. 6. Greedy algorithm vs. ad-hoc schemes.

a load factor LF_i, given by

$$LF_i = \max\left\{\frac{\text{used\_bandwidth}}{\text{total\_bandwidth}}, \frac{\text{used\_storage}}{\text{total\_storage}}\right\}$$

The hybrid algorithm would then iterate until the target function D can not be further reduced, and $LF_i \leq 1$ for all i. To summarize, most RPP formulations that base their decisions on object access frequencies, can be incorporated to a hybrid scheme that combines them with caching. Unfortunately, this has the side-effect that, in case the original RPP formulation bases its computations on uncertain parameters and erroneous estimations, it will negatively affect the performance of the hybrid scheme.

6.3. Practical considerations

Our work is based on the assumption that the storage space at the CDN servers is not sufficient to replicate 100% of the objects. However, one may argue that since disc space is extremely cheap these days, full mirroring at each CDN server is possible that would result in the minimum response

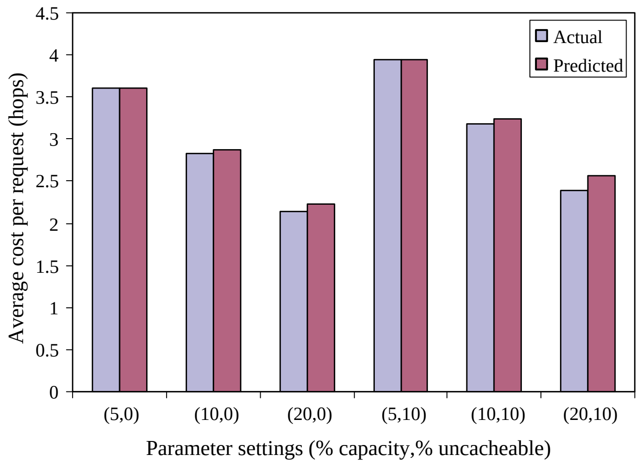


Fig. 7. Accuracy of the LRU hit ratio approximation.

time for client requests. There are several reasons, though, why this approach is either non-feasible or inefficient. First, not only the disk storage, but also the typical web content increases in size as well, with more and more content providers incorporating flash files and streaming media in their pages. Consequently, the storage space needed to hold one web site may become very large. Second, even if we assume that full mirroring is possible, the effect of updates would make such a system very expensive to manage. For instance, if the CDN provider has 10,000 servers distributed around the world, a simple update of a 10 KB object would generate an amount of traffic equal to 100 MB.

Another issue relates to the service quality that the content provider expects from the CDN provider, which brings us back to the point of full mirroring. Since the content provider pays for the hosting service, he or she may require that the content is replicated everywhere. In this case, there should be some sort of pricing agreement that would reflect the quality of the service. Some possible scenarios are: (i) the client pays proportionally to the data transferred at latency less than some threshold, and (ii) the client pays for a fixed number of replicas (primary replicas), but at the same time it is up to the CDN provider to optimize the download time of the content, by creating more replicas as appropriate (this is done for reasons of competition against other hosting services).

6.4. Future work

Nowadays, peer-to-peer (P2P) systems are responsible for the majority of file sharing traffic, while in the near future it is likely that they will also account for significant content delivery activity (especially for bandwidth consuming objects such as video). For instance, the SCRIBE system [35] (based on the Pastry project) provides group communication primitives for P2P overlay networks, Freenet [36] ensures anonymity of user requests, while Skipnet [37] provides the means to control in which peers data can be stored. All these three functionalities are necessary for a successful deployment of a CDN-like P2P system.

The basic concept of our work is applicable in P2P-CDNs, albeit after significant changes. Replication is already used in structured P2P systems, in order to avoid partitions whenever a peer departs from the network. It would be interesting to evaluate the system's performance when part of the resources at each peer are devoted to replication and another part to caching. All the related decisions should be taken by each peer in an autonomous manner. The problem becomes even more difficult if we take into account the fact that some peers will likely adopt a selfish behavior preferring caching to replication. These and other P2P related issues are parts of future work.

Another research direction is to apply our framework specifically for video delivery, after taking into account group communication and channel allocation issues [38]. The basic idea is to allow two methods of storage for each video object. The first is replication, whereby the whole video is stored, and the other one is caching, whereby only the first part of the video is kept. The rationale behind caching is that the client can start watching the video, while waiting to obtain a free stream from the occupied video server. Evaluating the relative dropout ratios of a hybrid versus a pure-replication scheme might result in new optimization opportunities.

7. Conclusions

In this paper, we investigated the potential performance gain of combining replica placement and caching techniques in a CDN architecture. We introduced an analytical model to predict the relative benefit of each technique, and proposed a simple greedy algorithm to solve the combined caching and replica placement problem. The simulation results indicate that there is indeed much room for performance improvement, compared to stand-alone replication or caching mechanisms. More specifically, savings up to 40% in user-perceived latency were observed, under various system parameters. Since we only considered coarse-grain (i.e. per site) replication, a straightforward extension of this work is to investigate the performance gain of the hybrid approach under finer-grain replication schemes (e.g. cluster-based replication [11]).

Acknowledgements

This work was completed while Spiridon Bakiras was at the University of Hong Kong, and was supported in part by the Areas of Excellence Scheme established under the University Grants Committee of the Hong Kong Special Administrative Region, China (Project No. AoE/E-01/99).

References

[1] T.M. Kroeger, D.D.E. Long, J.C. Mogul, Exploring the bounds of web latency reduction from caching and prefetching, in: Proceedings of USENIX Symposium on Internet Technologies and Systems, 1997.

[2] A. Bestavros, WWW traffic reduction and load balancing through server-based caching, IEEE Concurrency 5 (1) (1997) 56–67.

[3] V.N. Padmanabhan, L. Qiu, The content and access dynamics of a busy web site: findings and implications, in: Proceedings of ACM SIGCOMM, 2000, pp. 111–123.

[4] A. Shaikh, R. Tewari, M. Agrawal, On the effectiveness of DNS-based server selection, in: Proceedings of IEEE INFOCOM, 2001, pp. 1801–1810.

[5] J.D. Guyton, M.F. Schwartz, Locating nearby copies of replicated Internet servers, in: Proceedings of ACM SIGCOMM, 1995, pp. 288–298.

[6] Z. Fei, S. Bhattacharjee, E.W. Zegura, M.H. Ammar, A novel server selection technique for improving the response time of a replicated service, in: Proceedings of IEEE INFOCOM, 1998, pp. 783–791.

[7] O. Wolfson, S. Jajodia, Y. Huang, An adaptive data replication algorithm, ACM Transactions on Database Systems 22 (4) (1997) 255–314.

[8] J. Kangasharju, J. Roberts, K.W. Ross, Object replication strategies in content distribution networks, Computer Communications 25 (4) (2002) 367–383.

[9] L. Qiu, V.N. Padmanabhan, G.M. Voelker, On the placement of web server replicas, in: Proceedings of IEEE INFOCOM, 2001, pp. 1587–1596.

[10] S. Jamin, C. Jin, A.R. Kurc, D. Raz, Y. Shavitt, Constrained mirror placement on the Internet, in: Proceedings of IEEE INFOCOM, 2001, pp. 31–40.

[11] Y. Chen, L. Qiu, W. Chen, L. Nguyen, R.H. Katz, Clustering web content for efficient replication, in: Proceedings of IEEE International Conference on Network Protocols (ICNP), 2002, pp. 165–174.

[12] T. Loukopoulos, I. Ahmad, D. Papadias, An overview of data replication on the Internet, in: Proceedings of IEEE International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN), 2002, pp. 27–32.

[13] M. Rabinovich, O. Spatschek, Web Caching and Replication, Addison-Wesley, New York, 2002.

[14] W.W. Chu, Optimal file allocation in a multiple computer system, IEEE Transactions on Computers 18 (10) (1969) 885–889.

[15] P.M.G. Apers, Data allocation in distributed database systems, ACM Transactions on Database Systems 13 (3) (1988) 263–304.

[16] C.C. Bisdikian, B.V. Patel, Cost-based program allocation for distributed multimedia-on-demand systems, IEEE Multimedia 3 (3) (1996) 62–72.

[17] M. Karlsson, C. Karamanolis, M. Mahalingam, A framework for evaluating replica placement algorithms, Technical Report HPL-2002-219, HP Labs, 2002.

[18] L.W. Dowdy, D.V. Foster, Comparative models of the file assignment problem, ACM Computer Surveys 14 (2) (1982) 287–313.

[19] B. Awerbuch, Y. Bartal, A. Fiat, Optimally-competitive distributed file allocation, in: Proceedings of ACM STOC, 1993, pp. 164–173.

[20] M. Rabinovich, I. Rabinovich, R. Rajaraman, A. Aggarwal, A dynamic object replication and migration protocol for an Internet hosting service, in: Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS), 1999, pp. 101–113.

[21] T. Loukopoulos, I. Ahmad, Static and adaptive data replication algorithms for fast information access in large distributed systems, in: Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS), 2000, pp. 385–392.

[22] R. Tewari, N. Adam, Distributed file allocation with consistency constraints, in: Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS), 1992.

[23] A. Heddaya, S. Mirdad, WebWave: Globally load balanced fully distributed caching of hot published documents, in: Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS), 1997.

[24] L. Zhuo, C.L. Wang, F.C.M. Lau, Load balancing in distributed web server systems with partial document replication, in: Proceedings of IEEE International Conference on Parallel Processing (ICPP), 2002, pp. 305–312.

[25] P. Mirchandani, R. Francis, Discrete Location Theory, Wiley, New York, 1990.

[26] P. Radoslav, R. Govindan, D. Estrin, Topology-informed Internet replica placement, Computer Communications 25 (4) (2002) 384–392.

[27] B. Li, M.J. Golin, G.F. Italiano, X. Deng, K. Sohraby, On the optimal placement of web proxies in the Internet, in: Proceedings of IEEE INFOCOM, 1999, pp. 1282–1290.

[28] X. Tang, J. Xu, On replica placement for QoS-aware content distribution, in: Proceedings of IEEE INFOCOM, 2004.

[29] M. Karlsson, C.T. Karamanolis, Choosing replica placement heuristics for wide-area systems, in: Proceedings of IEEE International Conference on Distributed Computing Systems ICDCS, 2004, pp. 350–359.

[30] M. Karlsson, M. Mahalingam, Do we need replica placement algorithms in content delivery networks?, in: Proceedings of International Workshop on Web Content Caching and Distribution (WCW), 2002, pp. 117–128.

[31] C. Liu, P. Cao, Maintaining strong cache consistency in the world-wide web, in: Proceedings of International Conference on Distributed Computing Systems (ICDCS), 1997, pp. 12–21.

[32] K. Calvert, E. Zegura, GT Internetwork Topology Models (GT-ITM), Available at: http://www.cc.gatech.edu/projects/gtitm/.

[33] P. Barford, M.E. Crovella, Generating representative web workloads for network and server performance evaluation, in: Proc. ACM SIGMETRICS, 1998, pp. 151–160.

[34] A. Wolman, G.M. Voelker, N. Sharma, N. Cardwell, A.R. Karlin, H.M. Levy, On the scale and performance of cooperative web proxy caching, in: Proceedings ACM Symposium on Operating Systems Principles (SOSP), 1999, pp. 16–31.

[35] M. Castro, M.B. Jones, A.-M. Kermarrec, A.I.T. Rowstron, M. Theimer, H.J. Wang, A. Wolman, An evaluation of scalable application-level multicast built using peer-to-peer overlays, in: Proceedings of IEEE INFOCOM, 2003.

[36] I. Clarke, O. Sandberg, B. Wiley, T.W. Hong, Freenet: A distributed anonymous information storage and retrieval system., in: Proceedings of Workshop on Design Issues in Anonymity and Unobservability, 2000, pp. 46–66.

[37] N.J.A. Harvey, M.B. Jones, S. Saroiu, M. Theimer, A. Wolman, Skipnet: A scalable overlay network with practical locality properties, in: Proceedings of USENIX Symposium on Internet Technologies and Systems, 2003.

[38] M. Guo, M.H. Ammar, E.W. Zegura, Selecting among replicated batching video-on-demand servers, in: Proceedings of NOSSDAV, 2002, pp. 155–163.