Increasing the Performance of CDNs Using Replication and Caching: A Hybrid Approach

Spiridon Bakiras
Department of Computer Science
Hong Kong University of
Science and Technology
Clear Water Bay, Hong Kong
sbakiras@cs.ust.hk

Thanasis Loukopoulos
Department of Informatics
Technical Educational Institute of Lamia
3rd km. Old National Road
Lamia, Greece
luke@teilam.gr

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 this paper we investigate the potential performance gains by using a CDN server both as a replicator and as a proxy server. We assume a common storage space for both techniques, and develop an analytical model that characterizes caching performance under various system parameters. Based on the models predictions, we can reason whether it is beneficial to reduce the caching space in order to allocate extra replicas. The resulting problem of finding which object replicas should be created where, given that any free space will be used for caching, is NP-complete. Therefore, we propose a hybrid heuristic algorithm (based on the greedy paradigm), in order to solve the combined replica placement and storage allocation problem. Our simulation results indicate that a simple LRU caching scheme can considerably improve the response time of HTTP requests, when utilized over a replication-based infrastructure.

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 in front of the clients, proxy servers storing the most frequently accessed documents. User requests are forwarded to the proxy server which responds with the document if present, while cache misses result in requests being redirected to the origin server. Proxy caching, however, has certain disadvantages that limit its potential benefit: (i) the hit ratio reported in the literature is typically below 50% [16], and (ii) cache misses usually incur large redirection delay.

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 objective of a CDN is to increase the availability of the hosted sites while minimizing the response time of HTTP requests.

Even though caching and replication may benefit from each other, research work so far tends to focus on either one of them separately. In particular, research on CDNs focuses primarily on replica placement schemes, while caching was studied in the context of proxy servers (see related work). One reason for this (among others), is that a generic caching scheme offers no guarantees on content availability. While this is of no concern for proxies, it is less than acceptable for a CDN that wants to provide QoS guarantees to its subscribers.

Performance wise, implementing a caching scheme as part of a CDN architecture that already employs replication, can have a positive effect. For example, the study in [22] shows that the file popularity of a busy web server tends to follow a Zipf-like distribution with a parameter θ that is much higher than the one observed

Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)
1530-2075/05 $ 20.00 **IEEE**

in proxy server traces. Consequently, higher hit ratios may be achieved when caching is performed within a CDN system. 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 benefits of each technique, under various system parameters, and propose a hybrid 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 state the assumptions regarding the CDN architecture. The proposed hybrid replica placement algorithm is introduced in Section 4, while the simulation results are illustrated in Section 5. Finally, Section 6 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 [26], or server level [10]), and the criteria with which a suitable replicator is selected [10, 9], 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., [28] or thousands, e.g., [12, 13, 23]. Here, we consider 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 [22].

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 [6] 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.

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

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 [20] for a more complete survey on Internet data replication.

Models for replica placement date back to early 70s under the context of the file allocation problem (FAP) [7] and received attention from diverse research areas, e.g., distributed databases [1], video servers [4], etc. [14] provides a thorough categorization of replica placement papers and the assumptions they use. An old survey of FAP formulations can be found in [8]. 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 [2, 12, 13, 19, 24, 27, 28], to name a few. The target functions considered in these papers include client-replica distance [12], read access cost [13, 24], read and update cost [2, 19, 28], and replica availability [27]. Depending on the formulation various constraints were considered, e.g., server storage capacity [2, 13, 19], processing capacity [24], bandwidth [12], etc. Another distinguishing factor for the above papers is whether they tackle the dynamic version of the problem. [2, 24, 28] 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). [24] also aims at balancing the load between the replicators. Load balancing is also the target of [11] with the assumption that the network has a tree structure, while it is also considered in [29], where the problem is replicating the contents of a single site.

Another option to formulate replica placement is by using the k-median problem [21], 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. [12, 17, 23, 25] are papers based on k-median formulations. In [17] the authors solve the problem to optimality for a tree network, using dynamic programming. [23] proposes a greedy heuristic that outperforms dynamic programming in non-tree networks, while [12] compares various heuristics and concludes that a greedy one that performs back tracking offers the better results. Finally, [25] provides heuristics specifically tailored for the Internet topology.

Here, we take advantage of past research on replica placement, in order to build the model of Section 3. More specifically, we 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.

To conclude, in a recent study [15], the authors investigate how replica placement algorithms perform compared to pure caching. 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 close to the best replica placement algorithms. However, they acknowledge that replica placement is essential for reasons such as reliability and availability. Their findings are supportive for our research, since they illustrate the need for a hybrid approach.

3. System Model

Consider a generic CDN infrastructure consisting of N geographically distributed servers. Let $S^{(i)}$, $s^{(i)}$ be the name and the total storage capacity (in bytes) of server i, where $1 \leq i \leq N$. 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 $C(i,j) = C(j,i)$.

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 main-

taining 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 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. 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

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

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 (Figure 1). 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 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.

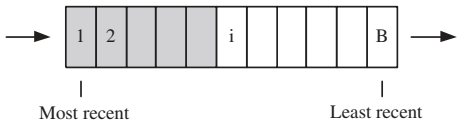


Figure 1. The LRU buffer.

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

$$\begin{aligned}\bar{h} &= \sum_{i=1}^{K}(i+\bar{h})p^{i-1}(1-p) + \sum_{i=K+1}^{\infty} Kp^{i-1}(1-p) \\ &= \frac{p^{-K}-1}{1-p} \\ \bar{m} &= \sum_{i=1}^{\infty} ip^{i-1}(1-p) = \frac{1}{1-p}\end{aligned}$$

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

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

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 Figure 1). 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 Figure 1). 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 [18]), and (ii) the study in [22] showed that the duration between successive modifications of an object is relatively large (between one and 24 hours), 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 Equation (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 stand-alone replica placement problem with storage capacity constraints has been shown to be NP-complete (e.g., in [13, 19]) and thus can not be solved to optimality. Various heuristics have been proposed in the literature, as discussed in Section 2.2. Notice that from an algorithmic perspective, the only real difference between the stand-alone replica placement problem and the hybrid case is in the computation of the cost function D (which, in the later case, includes the cache hit ratio). Therefore, it is easy to see that most heuristics for the replica placement problem can be suitably modified to account for the hybrid case. Clearly, applying such modifications to all the main heuristics proposed in the literature exceeds the scope of the paper, which rests in demonstrating that combining replication and caching yields considerable performance gains compared to stand-alone approaches. Thus, we chose as our evaluation basis the greedy-global heuristic [13, 15, 23] which, as shown in [14], achieves very good solution quality. Greedy-global performs in iterations; during each iteration, all the server-object pairs are compared, and the one that produces the largest benefit value is selected for replication. The heuristic terminates when all the servers reach their storage capacity, or the remaining possible replica creations result in negative benefit.

The hybrid case begins with a CDN network that only stores the primary replicas, meaning that all storage space is given to caching. During each iteration, all possible replica creations are evaluated, and the best one is selected. The evaluation consists of calculating the benefit due to the new replica, and comparing it to the cost increase due to restricting the cache size (i.e., decreasing the cache hit ratio). If it is larger, the replica becomes a candidate. At the end of the iteration, the best replica candidate is selected and created. The detailed pseudo-code is illustrated in Figure 2.

Lines 1–4 constitute the initialization part of the algorithm. 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, whereby 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.

```
replicaPlacement (Input: C(i,j), r_j^(i), s^(i))
(1)  D ← 0;
(2)  for (all pairs S^(i) and O_j)
(3)      SN_j^(i) = SP_j;
(4)      calculate h_j^(i);
(5)      D ← 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 ≥ 0))
(9)              b_ij ← (1 − h_j^(i)) r_j^(i) C(i, SN_j^(i));
(10)             for (all O_k)
(11)                 calculate h_{k,new}^(i);
(12)                 Δh ← h_k^(i) − h_{k,new}^(i);
(13)                 b_ij ← b_ij − Δh r_k^(i) C(i, SN_k^(i));
(14)             for (all S^(k) ≠ S^(i) with X_kj = 0)
(15)                 Δc ← C(S^(k), SN_j^(k)) − C(S^(k), S^(i));
(16)                 if (Δc > 0)
(17)                     b_ij ← b_ij + Δc(1 − h_j^(k)) r_j^(k);
(18)     select pair (S^(i), O_j) with max benefit b_ij;
(19)     D ← D − b_ij;
(20)     X_ij ← 1;
(21)     s^(i) ← s^(i) − o_j;
(22)     for (all O_k)
(23)         h_k^(i) ← h_{k,new}^(i);
(24)     for (all S^(k) ≠ S^(i) with X_kj = 0)
(25)         update SN_j^(k);
(26) return X, D;
```

Figure 2. The hybrid algorithm.

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 (with no caching) is $O(RMN^2)$. Notice, however, that we make the implicit assumption that the complexity of evaluating the hit ratio $h_{k,new}^{(i)}$ in line 11, is $O(1)$. In the following paragraphs we introduce some implementation details to justify the above assumption.

First, consider the approximation of K in Equation (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 Figure 2). 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 Equation (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

5.1. Simulation Setup

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 [5], 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. 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. Finally, 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).

Datasets: Although many web proxy traces are available for analysis, to the best of our knowledge no CDN log files exist in the public domain. Using proxy traces in the experiments would be inappropriate, since they account for the requests towards one server. Therefore, we decided to use the SURGE model [3] in order to generate a separate synthetic workload for each of the 200 web sites in our simulation. For simplicity, 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, 100 sites of medium popularity and 50 sites of high popularity. 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 $\mu \pm 3\sigma$.

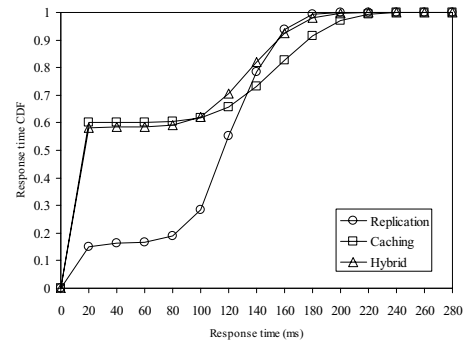
5.2. Simulation Results

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

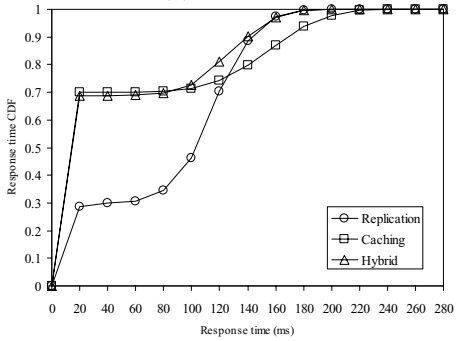
1. *Replication*: This is the stand-alone replica placement algorithm, using the greedy global approach [13].
2. *Caching*: All the storage capacity at the servers is allocated to caching. It is included in the experiments for comparison reasons only, since pure caching has its own administrative shortcomings.
3. *Hybrid*: This is the combined caching and replica placement algorithm introduced in Figure 2.

For reasons of fairness, we allowed an appropriate warm-up period before measuring the performance of the different alternatives, in order for the caches to reach their steady-state.

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. Figure 3 illustrates the relative performance of the various mechanisms for two different server capacities. 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 get a clear picture of the potentials and limitations of each technique. First, the result of pure replication is a very normal distribution of 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 at the CDN servers. Consequently, the CDF curve of the hybrid scheme 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. Overall, the hybrid approach outperformed the pure replication policy by approximately 40% on average, and the pure caching by 15% roughly.

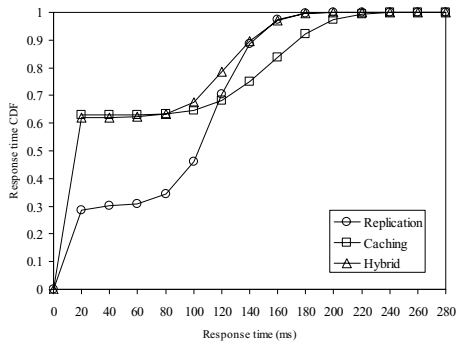


(a) 5% capacity.

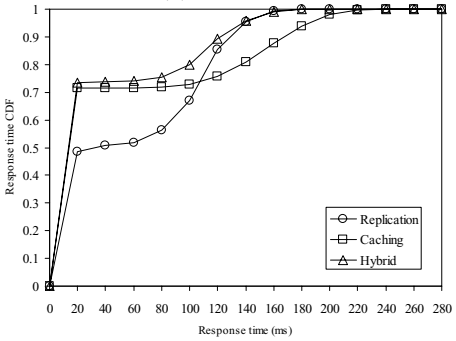


(b) 10% capacity.

Figure 3. Performance comparison of different content delivery mechanisms ($\lambda = 0$).



(a) 5% capacity.



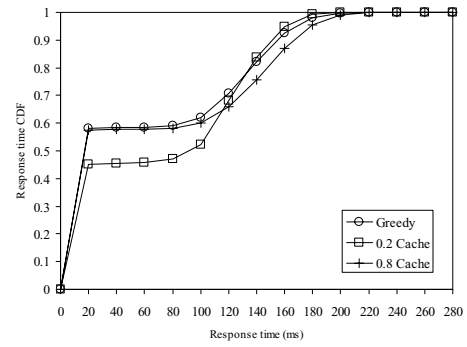
(b) 10% capacity.

Figure 4. Performance comparison of different content delivery mechanisms ($\lambda = 0.1$).

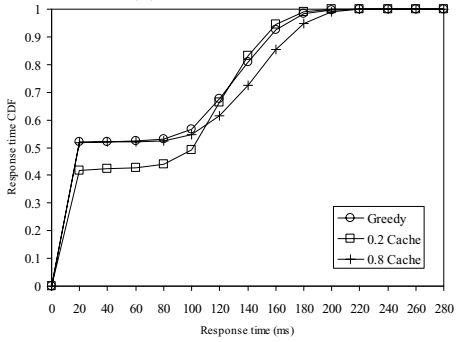
The second experiment investigates the performance difference between the hybrid policy and simple replication, under a scenario that does not favor caching. Specifically, we consider the case where 10% of the requests involve objects that have expired, and assume that the CDN imposes strong consistency. This means that site replicas are always consistent, while cached pages must be refreshed (accounting for additional delay). The detailed results are illustrated in Figure 4 and may be summarized as follows. The hybrid approach still 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% (on average). However, the performance gain against caching is increased, now reaching 20%. The same trends identified in Figure 3 also hold true in Figure 4. In general, the hybrid algorithm predicts 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 hybrid algorithm against ad-hoc 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 greedy global replication algorithm for the remaining part of the storage space?". We tested two different cases, one representing a favoring-replication ad-hoc approach and one a favoring-caching, fixing the cache sizes to 20% and 80%, respectively. The corresponding CDF plots for 0% and 10% staleness are shown in Figure 5. The main conclusion that can be drawn, is that ad-hoc approaches are not very effective. The hybrid algorithm constantly outperforms both alternatives. Further experiments with 40% and 60% cache sizes (not shown here) confirm this. Moreover, ad-hoc approaches are sensitive to changes in the Zipf parameter θ (fixed to 1.0 in all our experiments). The hybrid algorithm, however, takes the Zipf parameter as input and defines a cache size that leads to higher performance, compared to both pure replication and pure caching.

Finally, Figure 6 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 indi-



(a) 5% capacity, $\lambda = 0$.



(b) 5% capacity, $\lambda = 0.1$.

Figure 5. Greedy algorithm vs. ad-hoc schemes.

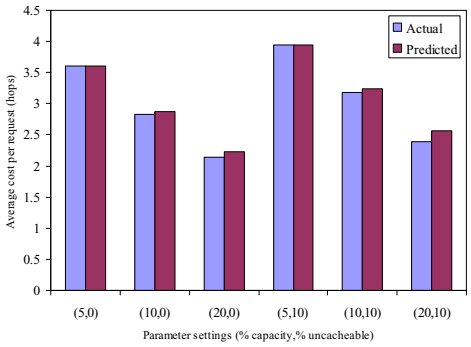


Figure 6. Accuracy of the LRU hit ratio approximation.

cate 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%.

5.3. Discussion

The overall conclusion is that combining caching and replica placement mechanisms yields significantly better 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 in the network so that the maximum delay is bounded, 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 (15%-20%), we expect that against a per-cluster replication scheme hybrid will again be the winner with the latency reduction varying in between the per-site replication and the caching case (depending on how the clusters are constructed [6]). Proving the validity of the above claim is left for future work.

6. Conclusions

In this paper we investigated the potential performance improvements by combining replica placement and caching techniques in CDN architectures. In order to avoid ad-hoc solutions, we introduced an analytical model to predict the relative benefits of each technique. The model itself estimates, within a very small error margin, the expected hit ratio of an LRU caching mechanism, and can be used as stand-alone mechanism whenever such estimations are required. We proposed a hybrid algorithm, based on the greedy approach, in order to solve the combined caching and replica placement problem. 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. These findings illustrate clearly that the hybrid algorithm successfully inherits the merits from both replica placement (bounding the maximum client-object distance) and caching (allocating the most popular objects in first-hop servers). Towards the question of whether to use caching or replication in a CDN infrastructure and more generally in distributed Web servers, the paper provides evidence that the best choice is to use both.

Acknowledgment

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

[2] B. Awerbuch, Y. Bartal, and A. Fiat. Optimally-competitive distributed file allocation. In *Proc. ACM STOC*, pages 164–173, 1993.

[3] P. Barford and M. E. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proc. ACM SIGMETRICS*, pages 151–160, 1998.

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

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

[6] Y. Chen, L. Qiu, W. Chen, L. Nguyen, and R. H. Katz. Clustering web content for efficient replication. In *Proc. IEEE International Conference on Network Protocols (ICNP)*, pages 165–174, 2002.

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

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

[9] Z. Fei, S. Bhattacharjee, E. W. Zegura, and M. H. Ammar. A novel server selection technique for improving the response time of a replicated service. In *Proc. IEEE INFOCOM*, pages 783–791, 1998.

[10] J. D. Guyton and M. F. Schwartz. Locating nearby copies of replicated Internet servers. In *Proc. ACM SIGCOMM*, pages 288–298, 1995.

[11] A. Heddaya and S. Mirdad. WebWave: Globally load balanced fully distributed caching of hot published documents. In *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS)*, 1997.

[12] S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt. Constrained mirror placement on the Internet. In *Proc. IEEE INFOCOM*, pages 31–40, 2001.

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

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

[15] M. Karlsson and M. Mahalingam. Do we need replica placement algorithms in content delivery networks? In *Proc. International Workshop on Web Content Caching and Distribution (WCW)*, pages 117–128, 2002.

[16] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *Proc. USENIX Symposium on Internet Technologies and Systems*, 1997.

[17] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby. On the optimal placement of web proxies in the Internet. In *Proc. IEEE INFOCOM*, pages 1282–1290, 1999.

[18] C. Liu and P. Cao. Maintaining strong cache consistency in the world-wide web. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, pages 12–21, 1997.

[19] T. Loukopoulos and I. Ahmad. Static and adaptive data replication algorithms for fast information access in large distributed systems. In *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 385–392, 2000.

[20] T. Loukopoulos, I. Ahmad, and D. Papadias. An overview of data replication on the Internet. In *Proc. IEEE International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, pages 27–32, 2002.

[21] P. Mirchandani and R. Francis. *Discrete Location Theory.* John Wiley and Sons, 1990.

[22] V. N. Padmanabhan and L. Qiu. The content and access dynamics of a busy web site: findings and implications. In *Proc. ACM SIGCOMM*, pages 111–123, 2000.

[23] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *Proc. IEEE INFOCOM*, pages 1587–1596, 2001.

[24] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal. A dynamic object replication and migration protocol for an Internet hosting service. In *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 101–113, 1999.

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

[26] A. Shaikh, R. Tewari, and M. Agrawal. On the effectiveness of DNS-based server selection. In *Proc. IEEE INFOCOM*, pages 1801–1810, 2001.

[27] R. Tewari and N. Adam. Distributed file allocation with consistency constraints. In *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS)*, 1992.

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

[29] L. Zhuo, C. L. Wang, and F. C. M. Lau. Load balancing in distributed web server systems with partial document replication. In *Proc. IEEE International Conference on Parallel Processing (ICPP)*, pages 305–312, 2002.