World Wide Web
DOI 10.1007/s11280-007-0027-8

Prefetching in Content Distribution Networks via Web Communities Identification and Outsourcing

Antonis Sidiropoulos · George Pallis · Dimitrios Katsaros · Konstantinos Stamos · Athena Vakali · Yannis Manolopoulos

Received: 30 November 2005 / Revised: 5 March 2007 /
Accepted: 19 March 2007

Abstract Content distribution networks (CDNs) improve scalability and reliability, by replicating content to the "edge" of the Internet. Apart from the pure networking issues of the CDNs relevant to the establishment of the infrastructure, some very crucial data management issues must be resolved to exploit the full potential of CDNs to reduce the "last mile" latencies. A very important issue is the selection of the content to be prefetched to the CDN servers. All the approaches developed so far, assume the existence of adequate content popularity statistics to drive the prefetch decisions. Such information though, is not always available, or it is extremely volatile, turning such methods problematic. To address this issue, we develop self-adaptive techniques to select the outsourced content in a CDN infrastructure, which requires no apriori knowledge of request statistics. We identify clusters of "correlated" Web pages in a site, called *Web site communities*, and make these

A. Sidiropoulos · G. Pallis (✉) · D. Katsaros · K. Stamos · A. Vakali · Y. Manolopoulos
Informatics Department, Aristotle University,
Thessaloniki, 54124, Greece
e-mail: gpallis@csd.auth.gr

A. Sidiropoulos
e-mail: asidirop@csd.auth.gr

K. Stamos
e-mail: kstamos@csd.auth.gr

A. Vakali
e-mail: avakali@csd.auth.gr

Y. Manolopoulos
e-mail: manolopo@csd.auth.gr

D. Katsaros
Computer & Communication Engineering Department,
University of Thessaly, Gklavani 37, Volos, 38221, Greece
e-mail: dkatsaro@csd.auth.gr

communities the basic outsourcing unit. Through a detailed simulation environment, using both real and synthetic data, we show that the proposed techniques are very robust and effective in reducing the user-perceived latency, performing very close to an unfeasible, off-line policy, which has full knowledge of the content popularity.

Keywords data dissemination techniques on the web · content distribution networks · web communities · web prefetching · internet and web-based · web data mining

1 Introduction

The Web has evolved rapidly from a simple information-sharing mechanism offering only static text and images to a rich assortment of dynamic and interactive services, such as video/audio conferencing, e-commerce, and distance learning. However, its explosive growth has imposed a heavy demand on networking resources and Web servers. Users experience long and unpredictable delays when retrieving Web pages from remote sites. The obvious solution to improve the quality of Web services would be the increase of the bandwidth, but such a choice involves increasing the economic cost. Besides, the higher bandwidth would solve only temporarily the problems since it would ease the users to create more and more resource-hungry applications, bunching again the network.

A traditional method to cure this situation is *caching* [21, 42, 46]. Although, caching offers several benefits, like reduced network traffic, shorter response times, it has drawbacks, e.g., small hit rates [25] and compulsory misses. To compensate for such problems, traditional caching is coupled with *prefetching*, which aims at predicting future requests for Web objects and bringing those objects into the cache in the background, before an explicit request is made for them. The most common prefetching practice is to make predictions [22] based on the recent history of requests of individual clients, which is called *short-term prefetching* [32]. This *pull-based* model of content access is problematic, because it does not improve availability during "flash crowds", and can not resolve the performance problems related to Web server processing and Internet delays.

Content distribution networks (CDNs) promise to resolve such problems, by moving the content to the "edge" of the Internet, closer to the end-user. With the "key" content outsourced, the load on the origin server is reduced, the connection from a local content delivery server is shorter than between the origin Web server and the user, thus reducing latency, and since many users share the CDN servers, this service greatly increases the hit ratio. In this paper, we propose a novel prefetcing scheme for CDNs by developing an effective self-adaptive outsourcing policy.

The rest of this paper is organized as follows: Section 2 provides background information for CDNs. Section 3 outlines the motivation and contribution of this work. Section 4 provides a formal definition of the Web site communities and the prefetching problem. Section 5 describes the algorithm for the identification of the communities. In Sects. 6 and 7, we describe the simulation testbed and show the performance evaluation of the proposed prefetching scheme and finally, Sect. 8 concludes the paper.

2 Content outsourcing policies in CDNs

A CDN [37] is a network of cache servers, called *surrogate servers*, owned by the same Internet service provider that delivers content to users on behalf of content providers. More specifically, CDNs maintain multiple points of presence with clusters of surrogate servers which store copies of identical content, such that users' requests are satisfied by the most appropriate site (Figure 1). Thus, the client-server communication is replaced by two communication flows: one between the client and the surrogate server, and another between the surrogate server and the origin server, resulting into reduced congestion, increased availability and more effective content distribution. Typically, a CDN consists of:

1. a set of surrogate servers (distributed around the world), which cache the origin servers' content,
2. routers and network elements which deliver content requests to the optimal location and the optimal surrogate server,
3. an accounting mechanism which provides logs and information to the origin servers.

Apart from the networking issues involved in the establishment of the CDN's infrastructure, like client request redirection [39], logical overlay topologies for large scale distribution [44], storage capacity allocation [28], server placement strategies to improve average response time or bandwidth consumption [40] and empirical CDN performance measurement studies [24], in order to exploit the full potential of CDNs, crucial data management issues must be addressed, as well. Among them, the content outsourcing policy to follow lies in their backbone. Up to now, four distinct content outsourcing policies appeared in the literature (Figure 2):

1. Uncooperative pull-based: clients' requests are directed to their "closest" surrogate server (in terms of geographic proximity, or load balancing, etc.), which acts as a plain cache. The drawbacks of this practice is that CDNs do not always

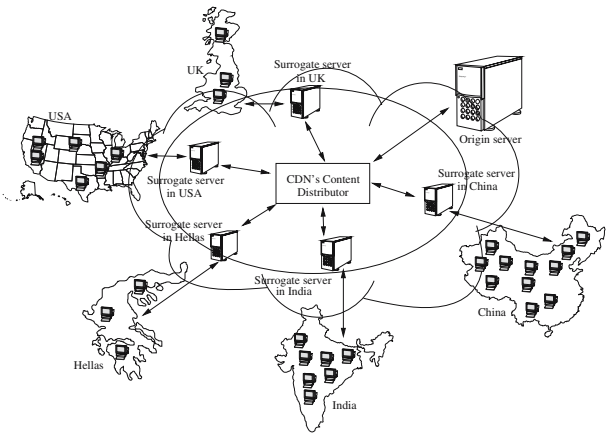

Figure 1 A typical CDN.

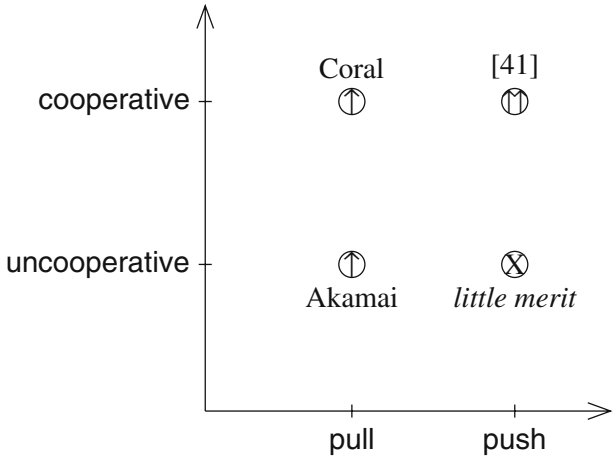

Figure 2 Content outsourcing policies.

choose the optimal server from which to serve the content [47] and they also incur excessive *replication redundancy*, that is, an object is replicated many times in different surrogate serves. However, many popular CDN's providers use uncooperative pulling [e.g., Akamai (http://www.akamai.com), LimeLight Network (http://www.limelightnetworks.com/)].

2. Cooperative pull-based: clients' requests are directed (through DNS redirection) to their closest surrogate server. The key in the cooperative pull-based CDNs [e.g., Coral (http://www.coralcdn.org/overview/)] is that the surrogate servers are cooperating with each other in case of cache misses. Specifically, using a *distributed index*, the surrogate servers find nearby copies of the requested objects, and store them in their caches [1]. The cooperative pull-based schemes are reactive wherein a data object is cached only when the client requests it, and consequently, these schemes impose a large communication overhead (in terms of the number of messages exchanged) when the number of clients is large. Moreover, this mechanism does not offer high fidelity when the content changes rapidly or when the coherency requirements are stringent.
3. Uncooperative push-based: the content is pushed (proactively) from the origin Web server to the surrogate servers. The requests can be satisfied either at a local surrogate server or at the origin Web server, but not at a nearby surrogate server due to the lack of informed request redirection. As a result, this scheme does not have much flexibility in adjusting replication and management cost.
4. Cooperative push-based: the content is pushed (proactively) from the origin Web server to the surrogate servers [40, 48]. Upon a request, if the surrogate server has an object replica, it serves the request locally, otherwise, it forwards the request to the closest server that has the object replica and relays the response to the client. In case that the requested object has not been replicated by some surrogate server (the requested object has not been outsourced), the request is served by the origin server. This replication policy is termed *long-term prefetching* [6, 12, 20, 48–50] and works by identifying collections of "valuable" objects to replicate. Although this scheme requires cooperation among the surrogate servers (incurring some communication and management cost to implement

Table 1 Main characteristics of content outsourcing policies.

Content outsourcing policies	Replication redundancy	Communication cost	Update cost	Temporal coherency
Uncooperative pull	High	High	High	Low
Cooperative pull	Low	High	Medium	Medium
Uncooperative push	High	Low	Medium	Medium
Cooperative push	Low	Medium	Low	High

the cooperation), this cost is amortized by the facts that the surrogate servers share efficiently the bandwidth among them and also reduce the replication redundancy, reducing in turn the cache consistency maintenance costs.

In contrast to the pull-based approaches (uncooperative and cooperative) which wait for the clients to request information, the push-based approaches let the content providers to proactively push the information into caches close to the user, expecting a further reduction to the access latency. Indeed, the authors in Chen et al. [6] concluded that the cooperative-push based policy provides the best overall results, when compared to the other approaches. Table 1 summarizes the pros and cons of the existing policies.

Motivated by the aforementioned conclusions, this paper adopts the cooperative-push based policy aiming to optimize it. For a content provider who wishes to outsource (part of) its content, the most important data management problems are the following:

(a) *Long-term prefetching problem*: Automatically select which objects to outsource.
(b) *Content placement problem*: Select which surrogates will replicate which objects.
(c) *Cache coherency problem*: Maintain the outsourced objects consistent (fresh).

In this paper, we deal with the problem of long-term prefetching for cooperative push-based content distribution. We do not examine the content placement problem for which adequate solutions already exist [6, 20, 28, 36, 38], neither examine coherency maintenance strategies, assuming the adoption of a high-performance algorithm for replica maintenance [34].

3 Motivation and paper's contributions

Straightforward approaches for performing content selection consist of the case where the site administrators decide about which content will be outsourced, and the case where (almost) the whole owner's site is replicated. Apparently, the first approach is out of question, since it is completely unscalable. The latter may seem feasible, since the technological advances in storage media and networking support have greatly improved. Though the very recent progresses in gaming and entertainment market proved that this is almost impossible to achieve. For instance, after the recent agreement between Akamai Japan and Sony Computer Entertainment (SCE), under which the first company is adopted as the content delivery platform by the PLAYSTATION@Network (an online service for PLAYSTATION 3), we can

deduce, since these are proprietary information, the huge storage requirements of the surrogate servers. Moreover, the evolution towards completely personalized TV, e.g., the GoTV (http://www.gotvnetworks.com/) reveal that the full content of the origin servers can not be completely outsourced as a whole.

Apart from the aforementioned naive and unscalable approaches for content selection, there exist some proposals for "automatically" selecting this content [6, 12, 20, 29, 48–50]. All of them assume knowledge of the popularity profile of each object, performing replication on *a per Website basis* [20, 48–50] (replicating all "hot" objects of the site as one unit), or performing replication on *a per cluster-of-objects basis* [6, 12, 29] (replicating "hot" content in units of groups of individual objects). Works by Chen et al. [6], Fujita et al. [12], Kangasharju et al. [20], Venkataramani et al. [48], and Wu and Kshemkalyani [49, 50] first perform a ranking of the objects based on their popularity, and then select an administratively tuned percentage of them to replicate to surrogate servers. Works by Kangasharju et al. [20], Venkataramani et al. [48], and Wu and Kshemkalyani [49, 50] replicate these objects as a single group, whereas Chen et al. [6] and Fujita et al. [12] cluster the objects into groups, based on time of reference, or number of references, etc, and perform replication in units of clusters. The clustering procedure is executed either by fixing the number of clusters or by fixing the maximum cluster diameter, thus using administratively tuned parameters, since neither the number or the diameter of the clusters can ever be known. Authors in Chen et al. [6] showed that long-term prefetching on a per cluster-of-objects basis is superior than on a per Web-site basis.

The use of popularity statistics has several drawbacks. Firstly, it requires quite a long time to collect reliable request statistics for each object. Such a long interval though may not be available, when a new site is published to the Internet and should be protected from "flash crowds" [19]. Moreover, the popularity of each object varies considerably [3, 6][Sect. IV]; for the WorldCup'98 trace, only 40% of the "popular" objects of the one day remain "popular" and the next day. In addition, the use of administratively tuned parameters to select the hot objects, or decide the number of clusters causes additional headaches, since there is no apriori knowledge about how to set these parameters. The authors in Chen et al. [6] and Fujita et al. [12], realizing the limitations of such solutions proposed the creation of clusters based on some ad hoc criteria, i.e., the directory structure of the Web objects. In addition, the greedy approaches [20] are not feasible to implement on real applications, due to their high complexity.[1]

Therefore, we need to implant intelligence into the cluster formation process. To reach this target we examined some qualitative characteristics of Web site creation and Web surfing processes. It is self-evident, that Web site creators (humans or application programs) tend to organize their sites into collections of interrelated objects, which usually deal with a (more or less) coherent topic and have dense linkage. Additionally, user groups access these collections according to their interests and navigation alternatives of the site. When a user enters such a collection, s/he is more likely to navigate within it rather than to jump to another one, since (a) his

[1]Because of the huge memory requirements, authors in Chen et al. [6] reported that they could not run all the experiments for the greedy heuristic policies.

interests lie within the collection, and (b) the dense linkage of the collection's pages imply a higher probability of selecting a link which will direct him to a page of the collection.

The collection of Web objects (pages) within a Web site, which refer to a coherent topic and have relatively dense linkage among them, is called *Web site community*. These communities are perfect candidate groups of objects to be long-term prefetched, as long as we can identify them by exploiting only the hyperlink information, and not using parameters, regarding their number, their diameter, or their topic (keywords).

Another motivation of this work is to study the content outsourcing problem under an analytic CDN simulation model which considers both the network traffic and the server load. Until now, the most noteworthy works [6, 20] do not take into account several critical factors, such as the bottlenecks that are likely to occur in the network, the number of sessions that can serve each network element (e.g., router, surrogate server) etc. Thus, the results presented may be misleading (they measure the number of traversed nodes (hops) without considering the TCP/IP network infrastructure). Therefore, the motivation for us is to develop a flexible simulation model that simulates in great detail the TCP/IP protocol as well as the main characteristics of a cooperative push-based CDN infrastructure model. Specifically, the main benefit of a detailed CDN simulation model is that it gives a (closely) realistic view to the CDNs' developers about which will be the profits for both the CDNs' providers and CDNs' customers if the proposed approach adapts to a real CDN's provider (e.g., Akamai).

3.1 Paper's contributions

This paper studies the problem of long term prefetching for CDNs on a per cluster-of-objects basis. The main contributions of this paper can be summarized as follows:

1. We provide evidence that Web communities exist within Web sites, revealing thus a "fractal" nature of the communities: communities at the small scale (*logical documents* [7, 30, 45]), communities at the medium scale (*site communities*, introduced in this paper), and communities at the giant scale (*hubs-authorities* [9, 14, 23]).
2. We provide a new, quantitative definition of the communities, different from the existing ones, which allows for overlapping communities, thus being more appropriate for our target application.
3. We provide a novel, self-tuning method for detecting such communities, which exploits only the hypertext links between pages, making no use of keywords to detect topics, and no ad hoc assumptions about the number or the nature of the communities. The proposed method is based on the plain intuition that a link between pages implies a *correlation* of the pages.
4. We develop a scalable and robust simulation environment, the *CDNsim* tool (found at http://oswinds.csd.auth.gr/~cdnsim/), to test analytically the efficiency of the proposed *C3i-PR communities-based long-term prefetching* scheme. Using real and synthetic test data, we show the efficiency of the proposed method, which can reap performance benefits almost equal to an analogous *C3i-Hot* off-line policy, which has a priori knowledge of the object popularity statistics.

4 Medium scale communities and long-term prefetching

This section provides a formal definition of the Web site communities and of the long-term prefetching problem. In the sequel, the terms "communities" and "clusters" are used interchangeably. Table 2 describes the variables used throughout the paper.

4.1 Earlier definitions of communities

The qualitative definition of a community as a part of a graph where internal connections are denser than external ones has already been introduced in the literature. For instance, in Greco et al. [14], Haveliwala [16], Kleinberg [23], and Page et al. [35], a Web community spans several sites, thus it is an *inter-site community*, and it is tightly connected to a specific topic, determined by a set of keywords. Analogous inter-site communities have been described in Flake et al. [9, 10], Kumar et al. [27], and Masada [31] using pure graph-theoretic techniques, like cliques [27], max-flow clusterings [9, 10, 18] or connected components [13, 31]. All these are *inter-site communities* and require heavy parameterization to work.

At the other extreme, *intra-site communities* have been introduced as *compound documents* [7] and *logical information units* [30, 45] at a much smaller scale, being comprised by a handful of Web objects of a single site. A compound document is a logical document authored by (usually) one author presenting a extremely coherent body of material on a single topic, which is split across multiple nodes (URLs). Similarly, a logical information unit is not a single Web page, but it is a connected subgraph corresponding to one logical document, organized into a set of pages connected via links provided by the page author as "standard navigation routes".

Table 2 Notation for variables.

Variable	Description
$G(V', E')$	Web graph where V' is the number of vertices and E' is the number of edges
V	Subgraph of the graph G
$M_{i,j}$	Adjacency matrix of the graph G
d_i	Graph's degree of node i
$d_i^{in}(V)$	Number of edges connecting node i to other nodes belonging to V
$d_i^{out}(V)$	Number of connections towards nodes in the rest of the network
X	Number of nodes in the internetwork topology
N	Number of surrogate servers
$C(N)$	Available storage capacity of N surrogate servers
S	Total size of all replicated objects
U	Number of users
O	Number of unique objects of a Web site
O_u	Number of objects requested by user u
$D_{U,N}$	Distance between the i-th user ($i \in U$) and node j ($j \in N$)
N_o	Set of locations where object o has been replicated
$l_{o(j)}$	Location of object o at the j-th surrogate server
C	Set of the Web communities where $C = \{C_1, C_2, ...C_n\} \subset G$
$c(x)$	Cohesion factor of community x
$F(G, V)$	Average cohesion factor $c(x)$ for all the communities of the Web graph G

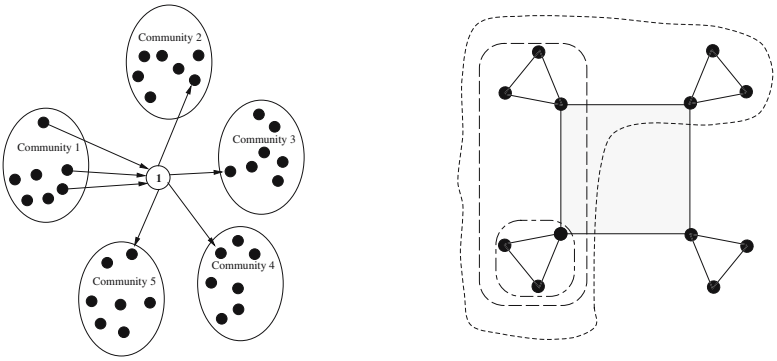

Figure 3 Sample communities where definition in Flake et al. [9] is not appropriate for long-term prefetching.

Apparently, compound documents and logical information units are not appropriate for the prefetching in CDNs. The notions of inter-communities could be used in our target application, but they present some important weaknesses. With clique-based communities, we could end up with a huge number of very small communities, since it is quite unprobable for a Web site to be comprised by large cliques; as a matter of fact we hardly expect to find any large cliques. With connected-component communities, we could end up with a single community comprised by the entire Web site. Max-flow communities seem more appropriate, but they suffer from their strict definition of the community, i.e., (a) they discover only non-overlapping communities, and (b) may leave some nodes outside of every community. For instance, according to Flake et al. [9] and Ino et al. [18], node 1 in the left part of Figure 3 will not be assigned to any community; similarly no community will be discovered for the graph shown in the right part of the figure, although at a first glance, we could recognize four communities based on our human perception, i.e., the four triangles; a careful look would puzzle us whether the four nodes at the corners of the square really belong to any community. (The meaning of the dashed and dotted lines will be explained later.)

In summary, at the giant scale, we have *inter-site communities*, at the small and "tiny" scale we have *logical-document communities*, but neither of them can be used gracefully to perform long-term prefetching in CDNs. Thus, we investigated the situation at the medium scale, i.e., the scale of a single Web site. We argue that communities do exist at this scale and they can be exploited in order to perform efficient long term prefetching in CDNs.

To support our claim we examined several Web sites with a crawl available on the Web. As an intuitive step, we confirmed the existence of such communities using graph visualization with *Pajek*.[2] As a sample, we present the drawing of http://www.hollins.edu (Figure 4), whose January 2004 crawl is publicly available.[3] We can easily see the co-existence of compound documents (at the lower right

[2] http://vlado.fmf.uni-lj.si/pub/networks/pajek/

[3] http://www.limfinity.com/ir/data/hollins.dat.gz

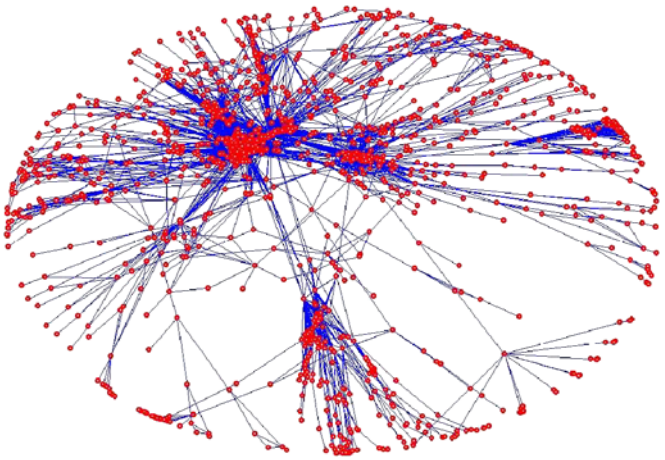

Figure 4 Web site communities at http://www.hollins.edu.

corner), with compact node clusters (at the upper center), and less apparent clusters (at the upper right of the image).

Therefore, for our target application we need a radically different formulation for the concept of a community, which: (a) is quantitative, (b) allows for overlapping communities, (c) is based only on the hyperlink information, assuming that two pages are "similar" or "correlated" if there exists a link between them and uncorrelated if such a link does not exist, (d) whose topic is much more generic than the logical document's topic, (e) does not make use of artificial "weights" on the edges of the Web site graph, and (f) does not make use of the direction of links in the Web site graph (because of the existence of the "back button"). Such a definition is described in the next subsection.

4.2 The proposed medium scale communities

To provide quantitative community definitions we consider d_i, the degree of a generic node i of the considered graph G. This quantity, in terms of the adjacency matrix $M_{i,j}$ of the graph, is $d_i = \sum_j M_{i,j}$. Considering a subgraph $V \subset G$, to which node i belongs, we can divide the total degree d in two portions: $d_i(V) = d_i^{in}(V) + d_i^{out}(V)$. The first term in the right-hand side of the equation is the number of edges connecting node i to other nodes belonging to V, i.e., $d_i^{in}(V) = \sum_{j \in V} M_{i,j}$. The second term in the right-hand side of the equation is the number of connections toward nodes in the rest of the graph, i.e., $d_i^{out}(V) = \sum_{j \notin V} M_{i,j}$. We define our communities as follows:

Definition 1 (Web site community) A subgraph V of a Web site graph G constitutes a Web site community, if

$$\sum_{i \in V} d_i^{in}(V) > \sum_{i \in V} d_i^{out}(V), \quad \text{or} \tag{1}$$

$$\frac{\sum_{i \in V} d_i^{in}(V)}{\sum_{i \in V} d_i^{out}(V)} > 1, \tag{2}$$

i.e., our sense of community implies that the sum of all degrees within the community V is larger than the sum of all degrees toward the rest of the graph G.

Large values of this fraction imply stronger communities. The above definition is an alternative concept for the original *modularity* measure of a network, which was introduced in Newman and Girvan [33] and it is roughly calculated as the number of edges falling within a community minus the expected number of edges between the nodes of the community in a completely random graph. Our concept of modularity departs from this probabilistic concept and it is able to characterize the most modular networks (e.g., collections of cliques with only minimal number of edges between them just to ensure connectivity), and non-clustered networks (e.g., random networks).

With the notation of Definition 1, Flake's community definition [9, 10] implies that $d_i^{in}(V) > d_i^{out}(V), \quad \forall i \in V$. Obviously, every community according to this definition is also a community according to our sense, but the converse is not always true. Thus, our definition overcomes the limitation explained in Figure 3 by implying more generic communities. The essence of definition 1 is that we are interested in "neighborhoods", i.e., paying attention to the communities as a whole, trying to increase their strength, rather than focusing on each individual member of the community, like the definition in Flake et al. [9], which is an extremely local consideration.

The astute reader will recognize that this definition does not lead to unique communities, since by adding or deleting "selected" nodes the property may still hold. For instance, looking at the right part of Figure 3, we see that the property is satisfied if: (a) we consider as a community each of the four triangles, or (b) if we consider as a community each of the two right triangles and the nodes contained inside the dashed line, or finally, (c) if we consider as a community the bottom right triangle and as a second community the nodes enclosed by the dotted line. None of these groupings is counterintuitive; on the contrary, as we move from case a to case c, the clustering justifies the belonging of the boundary nodes (nodes at the corners of the square) to their cluster.

Given the aforementioned definition, we can describe our communities identification (clustering) problem as follows:

Definition 2 (Web site communities discovery problem) Given a graph with ν vertices, find the (maximum number of) possibly overlapping groups of vertices (i.e., a non-crisp clustering), such that the number of edges within clusters is maximized and the number of edges between clusters is minimized, such that Definition 1 holds for each group.

The nice feature of this formulation is that we do not need to specify the number of clusters as a predetermined parameter, as in k-median or min-sum or min-max clustering. Instead, the optimal number of clusters could be any value between 1 and n, depending on the node connections. Moreover, it provides the freedom to stop the community discovery process in very early stages as long as definition 1 holds for each discovered community, without the need to discover the best possible grouping.

This formulation bears some similarity with the *correlation clustering* problem defined in Bansal et al. [2], but, in their computation of agreements (disagreements),

they also take into account any non-existing connections between vertices of the same cluster. Such a consideration is not compatible with definition 1 and also creates problems into the consideration of compound documents (in general, trees) as communities or members of them.

4.3 The cooperative push-based CDN infrastructure

In this subsection, we formally define the long-term prefetching problem, stating explicitly any assumptions made about the CDN environment. We consider a cooperative push-based CDN infrastructure, where each surrogate server knows what content is cached to all the other surrogate servers. This could be easily achieved with a summary cache architecture [8]. Furthermore, similar to previous works [6, 12, 20, 48–50], we consider that the Web objects fetched upon a cache miss are not inserted into the surrogate's cache, but simply forwarded to the requesting client.

Regarding the networking issues of the CDN, we assume knowledge of the following quantities: (a) network access costs, (b) placement of surrogate servers, and (c) storage capacity of the surrogate servers.

The input to our problem consists of the following information: (a) a network topology X consisting of $|X|$ network nodes, (b) a set N of $|N|$ surrogate servers, whose locations are appropriately [40] selected upon the network topology, (c) a set O of $|O|$ unique objects of the Web site which has an outsourcing contract with the CDN provider, where N_o denotes the set of locations where object o has been replicated and $l_{o(j)}$ denotes that the object o is located at the j-th surrogate; (d) a set U of $|U|$ users, where O_u is the set of objects requested by client u; (e) a distance function $D_{U,N} : U \times N \longrightarrow R^+$ associated with the location of the i-th user ($i \in U$) and j-th node ($j \in N$). This distance captures the cost when user i retrieves an object from surrogate j. We can describe the long-term prefetching problem for CDNs as follows:

Definition 3 (Long-Term Prefetching Problem) The long-term prefetching problem for CDNs is to select the content to be outsourced such that it minimizes:

$$repli\_cost = \sum_{i \in U} \left(\sum_{o \in O_i} \left(\min_{j \in N_o} D_{i,l_{o(j)}} \right) \right) \tag{3}$$

subject to the constraint that the total replication cost is bounded by $OS \leq C(N)$, where OS is the total size of all the replicated objects to surrogate servers, and $C(N)$ is the cache capacity of the surrogate servers.

A naive solution to this problem is to replicate all the objects of the Web site (full-mirroring) to all the surrogate servers. Such a solution is not feasible/practical because, although disk prices are continuously dropping, the sizes of Web objects increase as well (e.g., VOD, audio). Moreover, the problem of updating such a huge collection of Web objects is unmanageable. Therefore, we must resort to a selective policy, which will outsource only groups of "valuable" objects, and the proposal is to form groups of objects in terms of Web site communities. The next section develops such an algorithm.

5 Web communities identification and management

From the discussion in Subsection 4.2 we deduced that we need an efficient *non-crisp correlation clustering algorithm* in order to identify the communities. In the following paragraphs, we develop a novel, simple, heuristic algorithm, which, given a Web site graph, finds clusters of correlation-connected Web pages (Web site communities). We name this algorithm correlation clustering communities identification, in short, *C3i*.

5.1 The C3i algorithm

Let G be our Web-graph and $C = \{C_1, C_2, ...C_n\}$ the set of clusters such that $C_1 \cup C_2 \cup ..... \cup C_n \subseteq G$. Also, may exist i and j ($i \neq j$) such that $C_i \cap C_j \neq \emptyset$. Following Definition 1, we define a measure of strength of a community:

Definition 4 (Cohesion Factor) Let x be a cluster; the *cohesion factor* $c(x)$ of x is equal to:

$$c(x) = \frac{number\ of\ \text{“}external\text{”}\ links}{number\ of\ \text{“}internal\text{”}\ links} \tag{4}$$

Our target is to minimize this factor for all the clusters at the maximum number of clusters; therefore we must minimize the function $F(G, C)$, defined as:

$$F(G, C) = \frac{1}{|C|} \sum_{\forall x \in C} c(x) \tag{5}$$

where G the graph, C is the set of clusters and $|C|$ is the cardinality of C. It follows easily that finding the optimal correlated clusters is unfeasible, since this clustering problem is NP-hard (similar to Bansal et al. [2]). To attack the problem, we exploit some form of sampling. The algorithm we develop consists of two phases. In the first phase, the “kernel” nodes of clusters – the nodes around which we will build the clusters – are selected and a set of clusters is produced. Each cluster consists of only one kernel node. In the second phase, we expand each cluster until all nodes belong to some cluster. In the subsequent paragraphs, we explain our algorithm in detail. We also provide a pseudo-code in Figure 5, illustrating only the important parts of C3i.

Phase 1 (Selection of kernel nodes) The first step in our communities detection algorithm is the selection of the kernel nodes. There are two issues involved into this decision. The first decision regards the number of such nodes and the second decision regards their identity. Suppose we knew in advance the “best” communities of our input graph, i.e., the communities that solve the problem of Definition 4. Clearly, the number of kernel nodes should be equal to the number of “best” communities or large enough, so as successive cluster mergings to finally give the “best” communities. A naive approach could be to select all the nodes of the graph as kernel nodes, and start the clustering. Although, this approach seems appealing from a clustering quality perspective, the involved time overhead is prohibitive. Therefore, we have to select a portion of the total graph nodes, close to an estimate of the expected number of communities. Although there do not exist accurate estimates for this number,

Algorithm 1 The C3i algorithm

Input: $G(V, E)$: Web site graph
1: K=0;
2: **repeat**
Phase I: Selection of Kernel Nodes
3: select_kernel_nodes(G);
4: make each kernel node a cluster C_i;
5: $K = k \cup \{C_i\}$;
Phase II: The Clustering Procedure
6: Cohesion_clustering(K,G);
7: **until** no growth for K
8: **return** K;

Correlation_clustering(K,G)
Input: K: set of clusters
Input: $G(V, E)$: Web site graph
Step 1:
1: **repeat**
2: **for each** cluster c in K **do**
3: Check_1_node_expand(c);
4: **end for**
5: find_unions(K);
6: **until** no change in K;
Step 2:
7: **repeat**
8: **for each** cluster c in K **do**
9: Check_1_node_remove(c);
10: **end for**
11: **until** no change in K;
Step 3:
12: **repeat**
13: **for each** cluster c in K **do**
14: Check_x_nodes_expand(c,t);
15: **end for**
16: find_unions(K);
17: **until** no change in K;

Check_1_node_expand(c)
Input: c: cluster
1: **for each** node n connected to c **do**
2: **if** $F(c) > F(c \cup \{n\})$ **then**
3: $c = c \cup \{n\}$;
4: **end if**
5: **end for**

Check_1_node_remove(c)
Input: c: cluster
1: **for each** node n of c **do**
2: **if** $F(c) > F(c \setminus \{n\})$ **then**
3: $c = c \setminus \{n\}$;
4: **end if**
5: **end for**

Check_x_nodes_expand(c,t)
Input: c: cluster; t: integer
1: **for each** subgraph $S(|S| \leq t$ connected to c) **do**
2: **if** $F(c) > F(c \cup \{S\})$ **then**
3: $c = c \cup \{S\}$;
4: **end if**
5: **end for**

Figure 5 Justification for step 2 of C3i.

we can use the approximation reported in Guimera et al. [15], and select $n = \sqrt{|G|}$ kernel nodes.

The decision about which graph nodes will be selected as kernel nodes is easier, because we know that these nodes are "central" (visible) to their clusters. We can devise several methods to select "central" nodes. The naive and low complexity method is to select the nodes completely at random; this decision may prove useful in random graphs with "average" connectivity and equi-sized communities. A second method is to consider the nodes with the highest degree. A third alternative is to get the nodes with the highest PageRank value. Finally, we could select nodes with the highest HITS value.

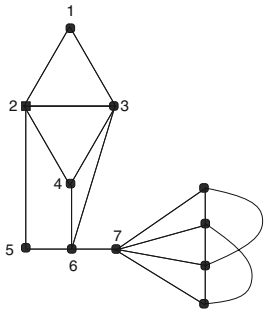

Figure 6 The C3i communities discovery algorithm.

Each of these methods has the obvious virtues, but some drawbacks as well. The first method will not work well in scale free graphs or with graphs with non equi-sized communities. The second method misses weakly connected clusters and clusters that do not actually have a center. The third method misses also important nodes, because PageRank is very sensitive to circles. So, if some nodes belong to big circles, then all of them get a high rank. As a result, we will select a lot of nodes which belong to the same cluster. In addition, PageRank misses the hubs (nodes with large out-degree), which may also be kernels of clusters. Finally, the last method misses clusters without "centroids".

The aforementioned drawbacks are intuitive, but they were verified through experimentation, as well. The investigation showed us that as long as we select as kernel nodes, the highest ranked nodes suggested by each heuristic, we manage to cover almost all the final "best" communities. Favoring a little, one over the other heuristic was not critical, provided that the bias was not tremendous. Thus, we selected the top $n/4$ from PageRank, the top $n/4$ authorities, the top $n/4$ hubs and $n/4$ randomly selected nodes, provided that they did not coincide with each other. From the experimental investigation, we found out that this solution covers 85% to 100% of the clusters, depending on the characteristics of the Web graph.

Phase 2 (The clustering procedure) After setting up the "seed" clusters, the main part of the algorithm consists of three steps and a procedure for performing clusters' unions, which is repeated after each step.

Step 1 We expand the clusters one node at a time. Addition of a node takes place if and only if the resulting function F (Eq. 5) of the new clustering is improving the previous choice. This step is repeated until none of the clusters can be expanded. At the end of each cycle, we check if we can perform cluster union.

Step 2 It is similar to step 1, but the nodes are checked whether they can be removed from the clusters. If a node removal results in a better function F, then it is removed from the cluster. This step could be avoided if the order of the "checks" in step 1 was perfect, i.e., have knowledge of the future. Although it is counter-intuitive can be explained though the example in Figure 6. Suppose that nodes 1 to 4 have already been to a cluster. This cluster can not grow beyond this, since the addition of none of the nodes 5 to 7 can give better cohesion factor (Eq. 4). At step 3 though, 6 and 7 or 5, 6

and 7 will be added to this cluster, depending on the order of consideration. Though, it is clear that node 7 does not belong to this cluster, and thus it will be removed at a later execution of step 2.

Step 3 We expand the clusters adding sets of nodes. Ideally, we should compute all possible sets S_i that are connected to each cluster x. For each S_i the function F would be tested to see whether S_i should be merged to x. The powerset $S = \{S_1, S_2, ...S_n\}$ is huge and this generate-and-test is unfeasible. So, we restrict S only to sets of size at most 6. This has minor effect in the quality of clustering, since larger components will compose other clusters which will be checked for merging with *find-unions*.

Finding Clusters' Union (find-unions) For every pair of clusters a and b, this procedure checks two conditions: the first one is whether the merging (union) of the two clusters is possible. The second one is whether the merging results in a better clustering. The selection of the union condition is very critical, because it affects both the speed and the quality of the clustering results. The selection could be performed dynamically, depending on the size of the graph and the quality we need. A large number of alternatives can be devised for checking for similarity, for instance see Spertus et al. [43]. Each one of them has its virtues and drawbacks, but the final selected metric should be evaluated really fast and allow for the creation of a reasonable number of large clusters. Some representative cluster merging policies include the following:

1. $|a \cap b| \geq 1$ or ($|a \cap b| = 0$ and a node a_i in a is connected to a node b_j in b): they have at least one common node, or they are just connected.
2. $|a \cap b| \geq 1$: they have at least one common node.
3. $|a \cap b| \geq \sqrt{max(|a|, |b|)}$: the number of common nodes is at least the square root of the size of the larger cluster.
4. $|a \cap b| > max(|a|, |b|)/2$: their intersection is at least half the larger cluster.
5. NEVER: we merge a and b only iff $a = b$.

By selecting case 5 (NEVER), we get the best quality; by selecting case 1 we have the higher speed of convergence. Although it is not possible to derive analytical formulas supporting the appropriateness of one metric over the others, their experimental evaluation, along the lines of the work by Spertus et al. [43], gave us useful hints about their performance. We concluded that the fourth cluster-merging condition is the best compromise between speed and quality.

When the merging condition holds, then the actual merging takes place iff the resulting clustering is better than the previous one (Definition 5). The above two phases are repeated until every node belongs to at least one cluster. If there are nodes which do not belong to any cluster ("orphaned" nodes), then Phase 1 is repeated. The selection for the new kernel nodes is made only for the set of "orphaned nodes".

Finally, following similar reasoning to [41], the resulting clusters are checked whether they are indeed correlated (Definition 4) and the non-correlated clusters are deleted. This is necessary since, during the execution of the algorithm, the value of $c(x)$ for each cluster x is decreasing, but there is no guarantee that all the $c(x)$ functions will converge to a value less than 1.

Time complexity Apparently, an exact analysis of the computational complexity of the algorithm can not be carried out, due to its depedence on the characteristics of the input graph and due to the randomized nature of the algorithm itself (i.e., order of examination of the nodes). Though, in the next lines we provide a simplistic analysis capturing the essential features of the algorithm and its polynomial-time nature.

Suppose that the input graph consists of n nodes with average degree d_{avg}, that we have selected c_{init} initial kernel nodes, and when the algorithms terminates, it produces c_{fin} clusters; recall that $c_f = \sqrt{n}$. It is obvious that for two clusters c_1 and c_2, we can check the truth of the merging condition 4 in time $O(|c_1| + |c_2|)$, by scaning in parallel the sorted lists of the node IDs of the clusters. Since $c_1 \geq c_2/2$ and $c_2 \geq c_1/2$, we deduce that the merging operation terminares in time $O(|c_i|)$, where c_i is the largest cluster.

Firstly, we make some simplifying assumptions which will ease our analysis and which represent average or worst cases for our algorithm: (a) the clusters contain approximately the same number of nodes, (b) the clusters grow node-by-node despite the fact that both routines *check_1_node_expand* and *check_x_nodes_expand* are called, (c) the cluster-merging test fails after such elementary growths, (d) the merging of clusters takes place only at the very final stage, when the clusters have reached the maximum possible growth and then the algorith terminates, and (e) each cluster node contains approximately $d_{avg}/4$ links to non-cluster nodes. The assumption c is an extremely bad situation for the our algorithm, because it will call too many times the *find_unions* routine and will apply it to too many pairs of clusters since no merging takes place. Moreover, this node-by-node cluster growth accounts for the effect of the *check_1_node_remove* routine.

Let the size of a typical cluster during an iteration of the algorithm execution is s_1 (whose size is $O(\sqrt{n})$ in the worst case) and the number of clusters is nc_i. The value of s_i ranges from 1 to the size of the final clusters, for which we can assume that is equal to $O(\frac{n}{\sqrt{n}}) = O(\sqrt{n})$.

Then, the cost of algorithm can be roughly estimated as

$$\begin{aligned} Cost = (|s_i| d_{avg}/4 * \ & \text{cost}(\textit{check\_1\_node\_expand}) + \\ & \frac{nc_i * (nc_i - 1)}{2} * \text{cost}(\textit{check\_1\_node\_expand}) + \\ & \binom{s_i}{6} * \text{cost}(\textit{check\_x\_nodes\_expand})) * \\ & \text{final\_cluster\_size} * c_{fin} \Rightarrow \end{aligned}$$

$$Cost = (|s_i| * log_2(n) * O(|s_i|) + O(n_c^2) * O(|s_i|) + (|s_i|^6) * O(|s_i|)) * O(\sqrt{n}) * O(\sqrt{n}) \Rightarrow$$

$$Cost = (|s_i| * log_2(n) * O(|s_i|) + n * O(|s_i|) + O(|s_i|^7)) * O(n) \Rightarrow Cost = O(n^{\frac{9}{2}}).$$

In practice, the performance of the algorithm is two orders of magnitude faster than this bound.

5.2 Community maintenance and outsourcing

There are a couple of issues regarding the communities identification, namely (a) the ranking of the nodes inside the community to deal with cases that whole communities cannot be replicated due to storage constraints, and (b) the incremental maintenance of the communities, in case that new nodes are added/deleted in the Web site or the linkage changes. In the present paper, we address only the first issue, which is the most critical to the implementation of the proposed long-term prefetching scheme. We leave the second issue for future work, since it comprises a whole new problem, and we only point out some directions for coping with it.

It is quite probable that the generated communities do not fit into a surrogate server due to space constraints or it may be the case that we wish to store part(s) of community. In these cases, we must deal with the problem of ranking the nodes of community, i.e., measuring the nodes' relative "value". There exist various methods to do so, (a) classical graph traversal techniques, like breadth-first or depth-first traversal, (b) spectral techniques, like PageRank, node degree, and (c) object popularity-based techniques, etc. The adoption of each one of these alternatives gives a variant to our original C3i algorithm. The combination of C3i with popularity statistics, which results in the *C3i-Hot*, an off-line algorithm requiring knowledge of request statistics or training, is an interesting variant providing a performance upper bound for any on-line C3i algorithm. From the on-line variants of C3i, our preferred choice is the exploitation of the PageRank, due its direct connection to the "random surfer" model. The combination of C3i with PageRanking, referred to as *C3i-PR* is the proposed on-line algorithm for the communities identification problem, whose performance we are interested in quantifying.

The incremental community maintenance refers to the problem of identifying the resulting communities, when new nodes are added or deleted to the original Web site. Modification of the connections between existing nodes is another face of the above problem; it is equivalent to deletion of current nodes and addition of new nodes with the new connections. This problem has been addressed in Charikar et al. [5] (and the references therein) for clusterings that strive to maintain constant number of clusters or minimize the cluster diameter. Such heuristics could also be applied in our case as well. Additionally, we could cope with the incremental clustering in a greedy manner by assigning new nodes to the cluster that maximizes Eq. 4 or to the cluster for which the "betweenness" [11] of this vertex is the largest among all clusters. (The betweenness centrality index of a vertex measures the degree to which this vertex is between pairs of other vertices, i.e., on shortest paths connecting them.)

6 Simulation testbed

To evaluate the proposed methods we use trace-driven simulations developing an analytic simulation environment. Therefore, we need a system model for simulating the CDN's functionalities and the internet topology. Furthermore, we need a collec-

tion of Web users' traces which access a Web site through a CDN, as well as, the topology of this Web site (in order to identify the Web communities). Although we can find several users' traces on the Web[4], we do not have the respective Web site's topology, and vice versa. Therefore, we have no other choice than to use artificial workloads.

In this framework, we have developed a complete simulation environment, which includes the following: (a) a system model simulating the CDN infrastructure, (b) a network topology generator, (c) a client request stream generator, which captures the main characteristics of Web users' behavior, and (d) a Web site generator, modeling file sizes, linkage, etc.

6.1 Simulation model

We have implemented a simulation model for CDNs, called *CDNsim*. It is based on the ParaSol library,[5] which is a parallel discrete event simulation system. The tool is publicly available at http://oswinds.csd.auth.gr/~cdnsim/.

In this work, we consider a CDN infrastructure consisting of N surrogate servers. We assume the case of homogeneous servers (all the servers have the same storage capacity) and each surrogate server's cache can hold a percentage of the Web site's total size. Then, we group the users based on their domains. The number of client groups is equal to the number of surrogate servers. Thus, each client group is connected with only one surrogate server and contains a few thousands clients. All CDN networking issues, like surrogate server selection, propagation, queueing, bottlenecks and processing delays are computed dynamically via our simulation model, which provides an implementation as close as possible to the working TCP/IP protocol, implementing packet switching, packet retransmission upon misses, etc. Finally, another important characteristic of CDNsim is that it manages efficiently the objects stored in surrogate servers by modeling their disks using the Bloom filters, as in Kulkarni et al. [26].

6.2 Network topology

Using the GT-ITM internetwork topology generator [51], we generated two random network topologies: Waxman and Transit-stub with a total of 1008 nodes. In Waxman model, the nodes are randomly assigned to locations on a plane, but an edge is created between a pair of node u and v with probability $P(u, v) = \alpha e^{\frac{-d}{\beta L}}$, where $d = |\overrightarrow{u} - \overrightarrow{v}|$, L is the maximum Euclidean distance between any two vertices, $\alpha > 0$ and $\beta \leq 1$. The Transit stub generates internetwork topologies composed of interconnected transit-stub domains and better reflects the hierarchical structure of real networks. We also constructed an AS-level Internet topology with a total of 3314 nodes, using BGP routing data collected from a set of 9 geographically-dispersed BGP peers in January 2003. At this point, we should stress that obtaining a complete and current AS Internet topology is a very difficult problem that can only be solved heuristically, and although the challenge is quite old, a significant step towards this

[4] http://ita.ee.lbl.gov/html/traces.html

[5] http://www.cs.purdue.edu/research/PaCS/parasol.html

goal has been made very recently by combining skitter, Routeviews, IRRs, IXPs [17]. In our work, we simply need a realistic AS topology, and not the complete and curtent AS topology, to use for proving the benefits of the communities outsourcing approach.

6.3 Web site generation

We used artificially generated Web graphs, constructed by the R-MAT tool [4] coupled with a size generation procedure which models the heavy-tail feature of file sizes. Specifically, the R-MAT tool produces synthetic but "realistic" Web-graphs, which match many of the patterns found in real-world graphs, including power-law and lognormal degree distributions, small diameter and "community" effects. Although, we performed extensive experiments with various types of graphs, in the paper, we present the results for a few representative graphs, namely dense, moderate-density and sparse graphs with the following characteristics. The sparse graphs consist of 4000 nodes and 15000 edges, the moderate-density graphs have 3000 nodes and 30000 edges, and the dense graphs consist of 2000 nodes and 40000 edges. For each graph set (dense, moderate-density, sparse), we generated graphs with 4 up to 1000 communities. Finally, for each of the generated density-community graph, we "altered" it by adding some "noise" edges between the communities; the percentage of these noise edges takes on low values (1% and 10%), leaving the communities practically unchanged, and high values (20% and 50%) altering significantly the picture of the communities.

6.4 Request streams generation

The workloads to the above Web graphs are streams of requests, called *client transactions*. To generate transactions, we used the generator described in [32], which reflects quite well the real access patterns. Specifically, this generator, given a Web site graph, generates transactions as sequences of page traversals (random walks) upon the site graph, by modeling the Zipfian distribution to pages. After producing the transactions, we follow three steps in order to convert them to a log file.

Step 1. We define the number of clients and distribute the transactions to the clients, so that each client will make at least one transaction).
Step 2. We define the time window that the transactions will be spread out; the length of the window determines how "heavy" or "light" the system load is. The default value that we used is one week.
Step 3. For each transaction, repeat the following:

- Step 3a. Assign a client who has made no transactions yet to the current transaction. If such a client does not exist, we select a client at random.
- Step 3b. A random timestamp is selected uniformly within the time window. This timestamp determines the starting time of the transaction. The time interval between to successive requests of the same transaction is selected uniformly with an average of 2 minutes.

7 Performance evaluation

In our experiments, we use the following performance measures in order to evaluate our proposed scheme:

1. Average response time (latency): the elapsed time between when a user issues a request and when it receives the response; it measures the user satisfaction and it should be as small as possible.
2. Replica factor: the percentage of the number of replica objects to the whole CDN infrastructure with respect to the total outsourced objects, i.e.,

$$replica\ factor = \frac{\sum_{i=1}^{K'} \frac{X_i \star outsourced\_o_i}{N}}{\sum_{i=1}^{K'} outsourced\_o_i},$$

where K' is the total outsourced objects, N is the number of surrogate servers and X_i is the number of replicas of the outsourced object i. This measure quantifies the cost of maintaining the replicas fresh and should be as small as possible.

Apparently, the measures of latency and replica factor are competing, in the sense that we can achieve minimum latency (high users' satisfaction) with high replication redundancy. Such a solution though is not satisfactory because it would imply huge coherency maintenance cost. Moreover, a large replica factor corresponds to a high cost (for CDNs' customers), since a large amount of objects should be hosted in surrogate servers [37]. Taking into account that the most popular Web sites have a competitive motivation to offer better service to their clients (low latency) at lower cost (low replica factor), we should consider both these measures in order to evaluate our proposed scheme.

7.1 Examined methods

In order to evaluate the proposed cohesion clustering scheme, we examined the methods described in the sequel. The algorithms described as "Hot" are off-line, in the sense that they have a priori knowledge of the request statistics and thus can perform "optimal" replication decisions. Some others (described as "Random") are baseline in the sense that they made random decisions for the respective issue. Apart from the proposed on-line *C3i-PR* algorithm and its off-line variant, the *C3i-Hot*, we considered various combinations of randomly creating clusters/communities (RandCl), with spectral (PageRank) or off-line (Hot) methods for ranking within communities, to compare the resulting methods with our proposed technique *C3i-PR*. These techniques are explained below:

1. Random Communities-based PageRank Replication (*RandCl-PR*): by randomly selecting graph nodes, we form communities. For each community, its objects are sorted in decreasing order of their PageRank value.
2. Random Communities-based Hot Replication (*RandCl-Hot*): by randomly selecting graph nodes, we form communities. For each community, its objects are sorted in decreasing order of their popularity.
3. Hot-based Replication (*Top-Hot*): each surrogate server replicates the most popular objects from the whole Web site [20, 48–50], restricted by the available cache space. There are no communities.

4. PageRank-based Replication (*Top-PR*): each surrogate server replicates the highest PageRanked objects from the whole Web site, under the restrictions of the available cache space. There are no communities.
5. No Replication (*W/O CDN*): All the objects are placed on the origin server and there is no CDN. This policy represents the "worst-case" scenario.

The placement of surrogates as well as the content (community) placement to surrogates were performed according to the approach provided in [6].

7.2 Evaluation with synthetic data

7.2.1 Response time analysis

Using our testbed, we performed an analytic investigation of the performance of the proposed on-line method *C3i-PR* with the aforementioned baseline and off-line methods for different competing performance measures. We performed extensive experiments with various graph sizes (in terms of the number of vertices, edges, communities), with various client populations and request patterns, etc. The simulation model has been configured so as the CDN's infrastructure consists of 20 surrogate servers, where each of them has cache capacity equal to 20% of the total objects' volume.

In general, we expect the *-Hot* methods to exhibit better performance, since they are off-line and have knowledge of the request statistics. Apparently, the *W/O CDN* method will be the worst method. The interesting point to look at is *C3i-PR*'s performance with respect to the performance of the off-line algorithms. If it is only marginally worse than them, then our intuition about the role of communities in the long-term prefetching will prove very effective in designing a blind, self-tuning scheme. Of course, all the results should be interpreted by examining both the average latency and the corresponding replication factor.

Firstly, we tested the competing algorithms, for each network topology (AS, Transit-Stub and Waxman) with respect to the average response time, with varying number of communities. The results are reported in Figures 7, 8 and 9 for sparse, for moderate-density and for dense graphs, respectively. The *y*-axis represents time units according to Parasol's internal clock and not some physical time quantity, like seconds, minutes. So the results should be interpreted by comparing the relative performance of the algorithms. This means that if one technique gets a response time 0.5 and some other gets 1.0, then in the real world the second one would be twice as slow as the first technique.

In general, we observe that the best algorithms are those that exploit knowledge of the future, e.g., the *C3i-Hot* over *C3i-PR*, the *Top-Hot* over *Top-PR*. This observation does not hold for the pair of *RandCL-*, whose behavior is less stable. In most of the cases, *RandCL-Hot* is better than *RandCL-PR*, but their performance shows a lot of variation. This variation could not be filtered by the repetition of the experiments, which is expected since it simply confirms the "randomness" in the decision on the construction of the communities. The variation is so dramatic, that in a case the *RandCL-Hot* method outperforms the *C3i-PR* method; though, this behaviour is exceptional.

What is quite surprising and not exceptional in the performance of the *RandCL-* techniques, is the fact that, despite the variation in their performance, they perform

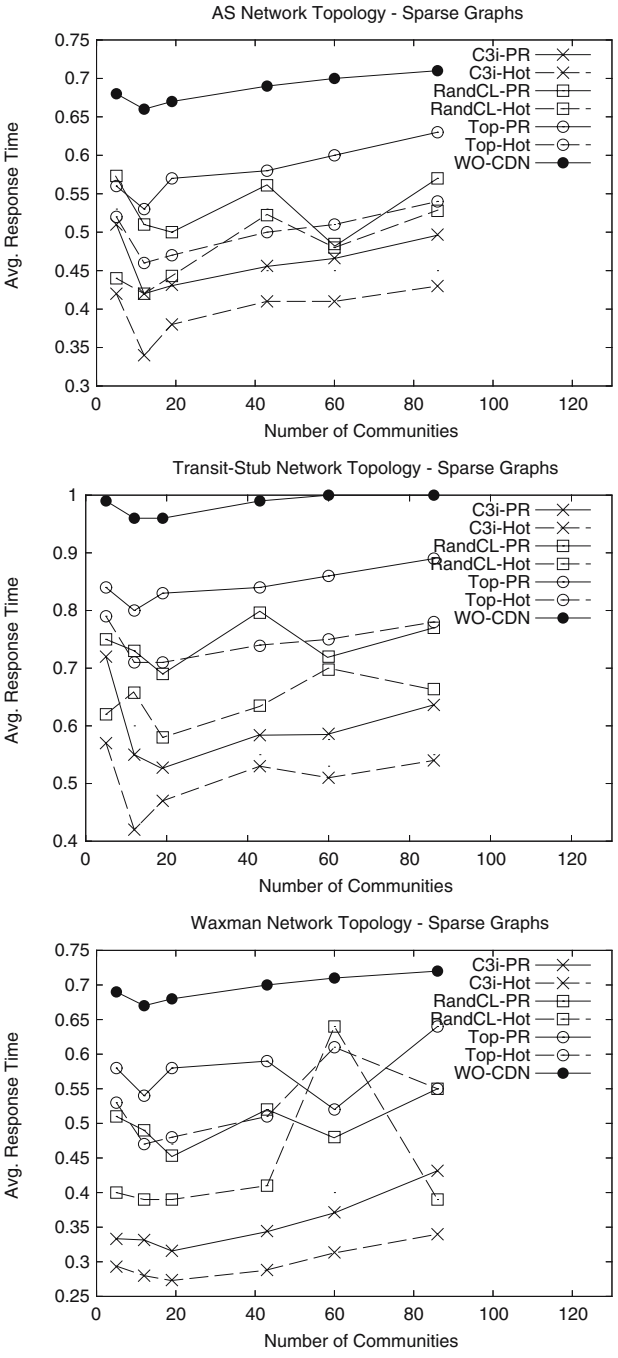

Figure 7 Average response time for sparse web graphs.

better than the *Top-* methods, across all network topologies (AS, Transit-Stub and Waxman) and Web site graph types (sparse, moderate dense and dense). This consistent behavior can not be ascribed to the "randomness", but comprises a sign of proof that treating content outsourcing in terms of communities does provide

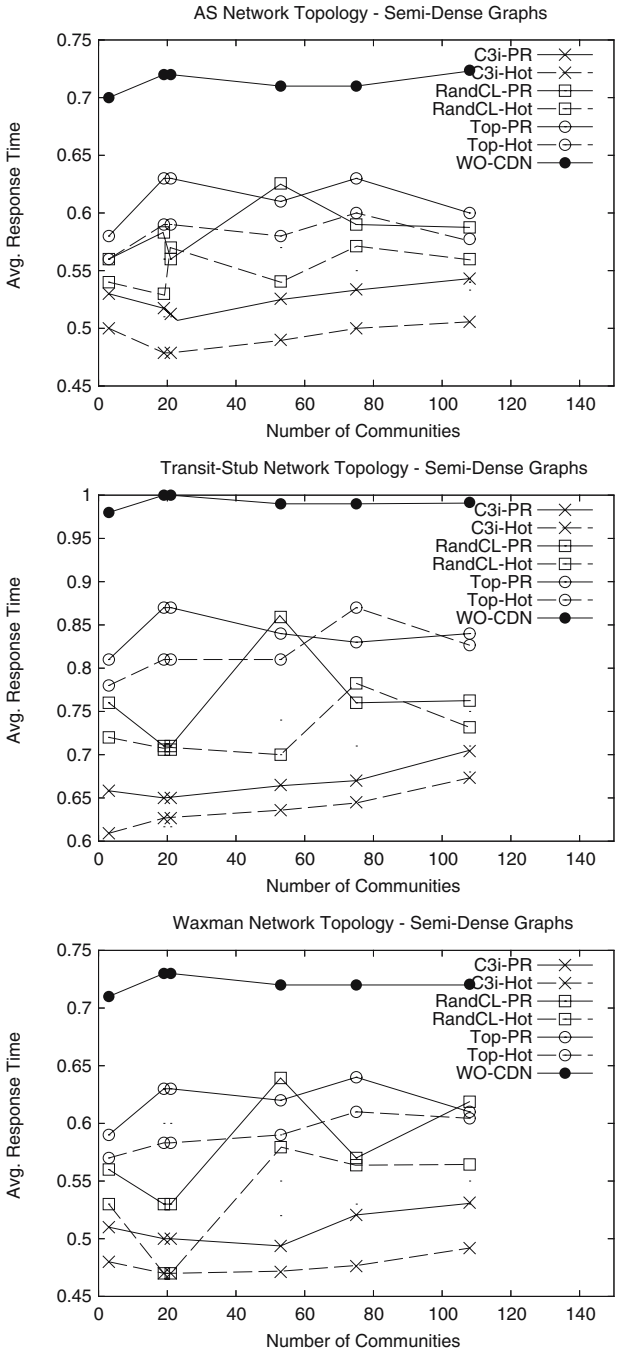

Figure 8 Average response time for moderate-density graphs.

significant benefits, compared to the case when outsourcing is performed in a "hotness" only basis.

Another point, which is apparent in almost all settings (in all topologies and in all Web graphs), that deserves attention is the relatively bad performance of all

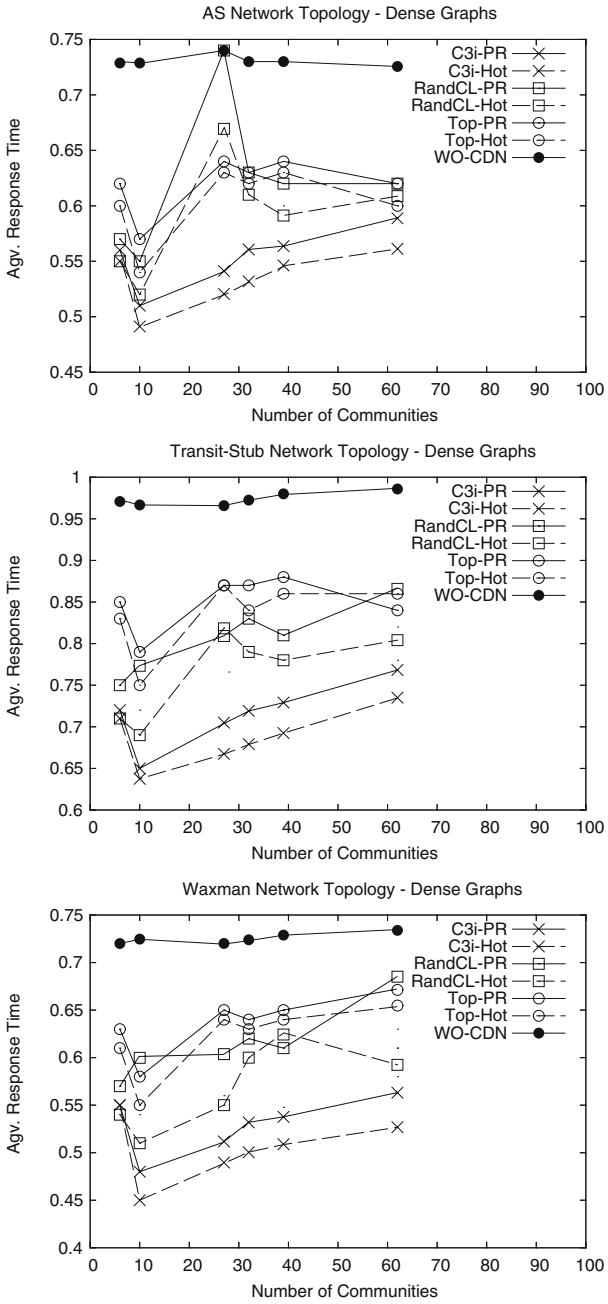

Figure 9 Average response time for dense graphs.

algorithms in the case of the existence of only very few communities in the Web site graph, i.e., the far left part of each graph. The existence of very few communities in a Web site graph implies two facts. The first is that there are no "attractors" for the visitors, thus the behavior of a surfer is completely random; the surfer might explore the whole site, instead of focusing only on a relatively small subgraph of the site,

i.e., a community. The second fact is that the content placement algorithm does not have many alternatives in the placement of communities. The bad performance of the algorithms based on "hotness" issues is explained by the former fact, whereas the bad performance of the algorithms based on communities is attributed to both of them.

Observing the performance of the *W/O CDN* we can see that it is practically constant, without showing any visible up/down trend with growing number of communities. This is a natural consequence, if we consider that it performs no clustering or hot content selection. Any observable variations, in dense graphs for instance, are due to the graph and request generation procedure.

Focusing now on the pair of our proposed algorithms we can see that in all network topologies and site graphs they offer the smallest response time. The general trend is that the response time is increasing with larger number of communities. This pattern is also apparent in algorithms that do not exploit any form of clustering (e.g., *Top-Hot*). The explanation for this increase is that, due to the request stream generation method, which simulates a random surfer (see Subsection 6.4), as the number of communities grows, the random surfers select different communities to visit, thus spreading the "popularity mass" to different collections of objects.

The performance gap between the online and offline variant of C3i widens as the number of communities grows, for the same reason mentioned before, since, when the probability mass spreads into more communities, the highest pagerank nodes are not those with the highest preference in the user visits (as generated with the request generation procedure).

Examining the relative performance of the two C3i algorithms w.r.t. the graph size, we observe that the performance gap between the two is larger for the case of sparse graphs than for the cases of dense and moderately-dense graphs. This is explained from the fact that in dense and moderately-dense graphs, the communities are not blurred by the noise introduction procedure and thus can be effectively discovered and exploited by *C3i-PR*.

Finally, with respect to the impact of the network topology on the performance of the algorithms, the pattern that seems to prevail, especially in the case of dense graphs, is that the Waxman topologies offer the lowest response times followed by the AS topologies, whereas the Transit-Stub topologies perform worst. The uniform placement of surrogates servers in the Waxman topology is advantageous, because the existence of hierarchies, like those in AS and Transit Stub, imply longer travels for the packets.

7.2.2 Replication redundancy analysis

Table 3 presents the algorithms' performance with respect to the replica factor for the AS network topology; analogous results were obtained for the other topologies. Each table's cell is the replica factor of the algorithm with respect to that of *C3i-PR*. A plus sign indicates a surplus in replication redundancy, equal to the percentage value that follows the plus sign, whereas a minus sign indicates a reduced redundancy, equal to the percentage following the minus sign.

In general, replication and response time are interrelated and, in systems where the storage capacity is unrestricted and the access pattern is stable, the pattern of dependence among them follows the rule that the increased replication results in

reduced response times, because the popular data are outsourced closer to the final consumers. In such situations we quantify the content acceleration (response time decrease) per unit of replication factor, e.g., a gain of X seconds in response time results in an increase of the replication as of a factor Y.

Though, the situation changes when (a) the storage capacity is restricted, and (b) when cooperation among the replication servers is allowed. This is exactly the case of cooperative push-based content distribution. This architecture allows reduction in the replication redundancy and in the response time, at the same time. This is the explanation behind the behavior of the *C3i-* techniques which is in accordance with the central idea of the cooperative push-based architecture; *C3i-* methods achieve small response times and small replication factor. Regarding the performance of the proposed techniques, we observe that *C3i-PR* is beaten only by its off-line variant, and though in several cases the difference between the two is insignificant.

On the other hand, the family of *Top-* algorithms, which do not support cooperation, are not able to achieve the low response times of *C3i-* techniques, even though they have a huge replication factor. Finally, *RandCL-* methods which support the notion of communities can exploit to some extent the cooperation and achieve improved response times, but at the cost of a high replication redundancy.

7.3 Evaluation in larger environments

To investigate the performance of the algorithms in larger parameter settings, we conducted one experiment with 100 surrogate servers when each of them can hold up to 20% of the data objects (keeping the rest of the parameters unchanged), and

Table 3 C3i-PR gain-loss (%) w.r.t. replica factor.

# Communities	C3i-Hot	RandCl-PR	RandCl-Hot	Top-PR	Top-Hot
Sparse graphs					
5	−7	+17	+25	+197	+197
12	+9	+73	+81	+345	+345
19	−12	+14	+21	+335	+335
43	−3	+49	+56	+324	+324
60	−2	+62	+33	+266	+266
86	+5	+83	+48	+431	+431
Moderate-density graphs					
3	−3	+28	+26	+179	+179
19	+4	+38	+24	+298	+298
21	+2	+48	+28	+258	+258
53	+4	+49	+45	+315	+315
19	−12	+45	+41	+325	+325
108	+12	+23	+40	+266	+266
Dense graphs					
6	−5	+26	+37	+214	+214
10	−2	+46	+53	+254	+254
27	+1	+92	+76	+490	+490
32	+9	+45	+41	+158	+158
39	+2	+33	+35	+139	+139
62	+11	+116	+92	+311	+311

a second experiment for the case of 20 surrogate servers when each of them can hold up to 40% of the data objects (keeping the rest of the parameters unchanged). The results of the first experiment are illustrated in the left part of Figure 10, and of the second experiment, are presented in the right part of the same figure; here, we present only the results for the average response time, since the respective results about the replication factor follow the same pattern as the experiments presented earlier.

Evaluating the results of the first experiment, we see that an increase in the number of surrogate servers results in a reduction of the average response time, for the methods which exploit some form of intelligent content outsourcing. Also, the impact of the number of communities is less intense, and the response time increases less steeply. In the second experiment which examined the performance of the algorithms with larger cache sizes, we observe a rather intuitive phenomenon; the *Top* methods seem to outperform the *RandCL* methods. Indeed, with infinite cache capacity the percentage of outsourced hot objects increases and thus contributes to the reduction of the response time. Finally, for both experiments we observe that the variation in the performance of the *Rand* method does exist, but it is less intense.

7.4 Evaluation with real data

We conclude the evaluation by reporting on some experiments conducted using a real Web graph. The real Web site we used is the Stanford Web site from a September 2002 crawl, available at http://www.stanford.edu/~sdkamvar/research.html that consists of 281903 Web pages and ~2.3 million links. Unfortunately, there are not freely available large real site graphs with different characteristic to experiment with, but for the sake of completeness we present some results with real data. This site graph resembles the case of a synthetically generated graph, less dense than the moderately-dense synthetic graphs, with very few communities. In particular, for the synthetic moderately-dense graphs the ratio of links to nodes is 10, whereas for this real graph is 8.15. Regarding the number and size of communities, we observe that the ratio of number of nodes to number of communities is extremely small, around 137, when this ratio for the synthetically generated graphs is always larger

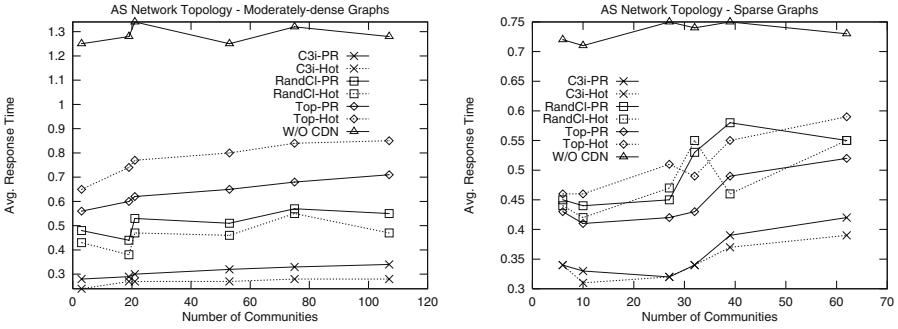

Figure 10 Experiments in larger settings. *Left*: With 100 surrogate servers—surrogate cache capacity equal to 20% of the total objects' volume. *Right*: With 20 surrogate servers—surrogate cache capacity equal to 40% of the total objects' volume.

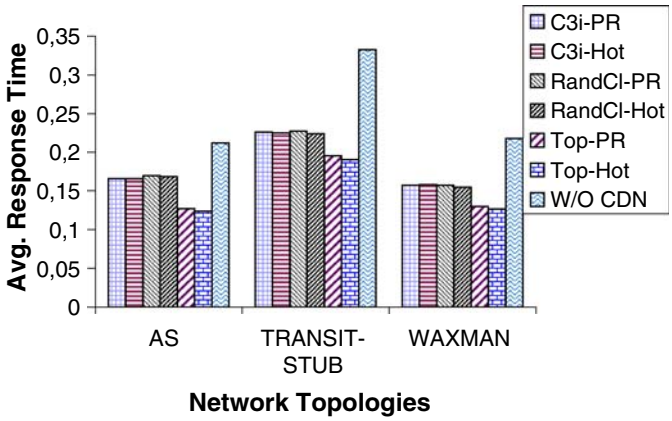

Figure 11 Average response time with a real Web graph consisting of 2043 communities.

than 200. From these data, we conclude that the expected performance results should be more or less similar to what is observed in the far left part of each graph of Figure 7. Therefore, we do not expect significant differences in the performance of the algorithms with respect to the response time.

We evaluate the performance of different replication schemes on a variety of network topologies using this dataset. The network topologies, client populations and request stream generation are the same as with synthetic data. The simulation model has been configured so as the CDN's infrastructure consists of 20 surrogate servers, where each of them has cache capacity equal to 20% of the total objects' volume. Our first experiment demonstrates the average response time for various network topologies. The results of this set of experiments are reported in Figure 11. From this figure, it can be seen that the *W/O CDN* approach, as expected, gives the worst response times and the *Top-Hot* outperforms all the approaches, but this is achieved at the expense of huge replication redundancy (refer to Table 4). On the other hand, the average response time for *C3i-PR*, and of the rest of the algorithms, is very close to *Top-Hot*'s value; the difference is in the order of 0.02, fairly smaller than the differences observed in our earlier experiments.

Finally, Table 4 summarizes the algorithms' performance with respect to the replication factor for the AS topology (with the other topologies showing analogous results). Again, we observe *C3i-PR*' robustness and the inefficiency of *Top-Hot* and *Top-PR* which, although outperform all the other approaches in terms of latency, they present the worst replica factor.

Table 4 C3i-PR gain-loss w.r.t. replica factor.

C3i-PR vs	2043 Communities (%)
C3i-Hot	−15
RandCl-PR	+5
RandCl-Hot	+5
Top-PR	+1299
Top-Hot	+1299

The results exhibit the robustness and efficiency of the *C3i-PR* scheme, which significantly reduces the average latency that the end-users see, keeping the replication redundancy at very low levels. The difference in performance (average response time) between the artificial Web sites and the Stanford Web site is explained by the fact that the former ones have on the average larger objects than the latter.

7.5 Summary of observations

Summarizing the above results, we see that the "champion" algorithm is *C3i-Hot*, the proposed off-line communities-based prefetching scheme. It is able to exploit the nice features of the cooperative push-based architecture, and the knowledge of the user preferences. This policy is closely followed by the proposed on-line variant, *C3i-PR*, which is proved to be a high performance (low average response time), low overhead (small replica factor) technique. Therefore, *C3i-PR* is an effective and realistic solution, bridging the performance gap between user satisfaction (low latency) and content provider benefit (small replica factor).

8 Conclusions

We addressed the long-term prefetching problem for CDNs. Differently from all other relevant approaches, we refrained from using any request statistics in determining the outsourced objects. We exploited the plain assumption that "communities" do exist within Web sites, which act as attractors for the visitors, due to their dense linkage and the fact that they deal with coherent topics. We provided a new algorithm to detect such communities; this algorithm is based on a new, quantitative definition of the community structure. We made them the basic outsourcing unit, thus providing the first "predictive" use of communities, which so far have been used for descriptive purposes.

The virtue of our approach stems from the fact that a CDN provider (e.g., Akamai) can execute an on-line community identification algorithm, e.g., *C3i-PR* on the Web site of its client before publishing its site so as to protect it from flash crowds, and after collecting sufficient of reliable statistics can execute an off-line method, e.g., *C3i-Hot*, to better tune the overall system performance.

To gain a basic, though solid understanding of our proposed methods' strengths, we implemented a detailed simulation environment to test the communities-based long-term prefetching method. Using both synthetic and real data we showed that the proposed method provides low average client response time with low replication redundancy.

References

1. Annapureddy, S., Freedman, M.J., Mazieres D.: Shark: scaling file servers via cooperative caching. In: Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA, 2–4 May 2005
2. Bansal, N., Blum, A., Chawla, S.: Correlation clustering. Mach. Learn. **56**(1–3), 89–113 (2004)

3. Bent, L., Rabinovich, M., Voelker, G.M., Xiao, Z.: Characterization of a large Web site population with implications for content delivery. World Wide Web J. **9**(4), 505–536 (2006)
4. Chakrabarti, D., Zhan, Y., Faloutsos, C.: R-MAT: A recursive model for graph mining. In: Proceedings of the SIAM Data Mining Conference (SDM) (2004)
5. Charikar, M., Chekuri, C., Feder, T., Motwani, R.: Incremental clustering and dynamic information retrieval. SIAM J. Comput. **33**(6), 1417–1440 (2004)
6. Chen, Y., Qiu, L., Chen, W., Nguyen, L., Katz, R.H.: Efficient and adaptive Web replication using content clustering. IEEE J. Sel. Areas Commun. **21**(6), 979–994 (2003)
7. Eiron, N., McCurley, K.S.: Untangling compound documents on the Web. In: Proceedings of the ACM Conference on Hypertext and hypermedia (HT), Nottingham, 26–30 August 2003
8. Fan, L., Cao, P., Almeida, J.M., Broder, A.Z.: Summary cache: a scalable wide-area Web cache sharing protocol. IEEE/ACM Trans. Netw. **8**(3), 281–293 (2000)
9. Flake, G.W., Lawrence, S., Giles, C.L., Coetzee, F.M.: Self-organization and identification of web communities. IEEE Comput. **35**(3), 66–71 (2002)
10. Flake, G.W., Tarjan, R.E., Tsioutsiouliklis, K.: Graph clustering and minimum cut trees. Internet Math. **1**(4), 385–408 (2003/2004)
11. Freeman, L.C.: A set of measures of centrality based on betweenness. Sociometry **40**(1), 35–41 (1977)
12. Fujita, N., Ishikawa, Y., Iwata, A., Izmailov, R.: Coarse-grain replica management strategies for dynamic replication of Web contents. Comput. Networks **45**(1), 19–34 (2004)
13. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proc. Natl. Acad. Sci. U. S. A. **99**(12), 7821–7826 (2002)
14. Greco, G., Greco, S., Zumpano, E.: Web communities: models and algorithms. World Wide Web J. **7**(1), 58–82 (2004)
15. Guimera, R., Sales-Pardo, M., Nunes Amaral, L.A.: Modularity from fluctuations in random graphs and complex networks. Phys. Rev. E **70**, 025101 (2004)
16. Haveliwala, T.H.: Topic-sensitive PageRank: a context-sensitive ranking algorithm for Web search. IEEE Trans. Knowl. Data Eng. **15**(4), 784–796 (2003)
17. He, Y., Siganos, G., Faloutsos, M., Krishnamurthy, S.: A systematic framework for unearthing the missing links: measurements and impact. In: Proceedings of the USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), pp. 187–200. Cambridge, USA, 11–13 April 2007
18. Ino, H., Kudo, M., Nakamura, A.: Partitioning of Web graphs by community topology. In: Proceedings of the ACM World Wide Web Conference (WWW), Chiba, 10–14 May 2005
19. Jung, Y., Krishnamurthy, B., Rabinovich, M.: Flash crowds and denial of service attacks: characterization and implications for CDNs and Web sites. In: Proceedings of the ACM World Wide Web Conference (WWW), Honolulu, 7–11 May 2002
20. Kangasharju, J., Roberts, J.W., Ross, K.W.: Object replication strategies in content distribution networks. Comput. Commun. **25**(4), 376–383 (2002)
21. Katsaros, D., Manolopoulos, Y.: Caching in Web memory hierarchies. In: Proceedings of the ACM Symposium on Applied Computing (SAC), Nicosia, 14–17 March 2004
22. Katsaros, D., Manolopoulos, Y.: Prediction in wireless networks by Markov chains. IEEE Wireless Communications magazine, 2007 (in press)
23. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. J. ACM **46**(5), 604–632 (1999)
24. Krishnamurthy, B., Wills, C., Zhang, Y.: On the use and performance of content distribution networks. In: Proceedings of the ACM SIGCOMM Internet Measurement Workshop (IMW), San Francisco, 1–2 November 2001
25. Kroeger, T., Long, D.E., Mogul, J.: Exploring the bounds of Web latency reduction from caching and prefetching. In: Proceedings of the USENIX Symposium on Internet Technologies and Services (USITS), Monterey, 8–11 December 1997
26. Kulkarni, P., Shenoy, P.J., Gong, W.: Scalable techniques for memory-efficient CDN simulations. In: Proceedings of the ACM World Wide Web Conference (WWW), Budapest, 20–24 May 2003
27. Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A.: Trawling the Web for emerging cybercommunities. Comput. Networks **31**(11–16), 1481–1493 (1999)
28. Laoutaris, N., Zissimopoulos V., Stavrakakis, I.: On the optimization of storage capacity allocation for content distribution. Comput. Networks **47**(3), 409–428 (2005)
29. Leung, K.Y., Eric Wong, W.M., Yeung, K.H.: Designing efficient and robust caching algorithms for streaming-on-demand services on the Internet. World Wide Web J. **7**(3), 297–314 (2004)

30. Li, W.-S., Candan, K.S., Vu, Q., Agrawal, D.: Query relaxation by structure and semantics for retrieval of logical Web documents. IEEE Trans. Knowl. Data Eng. **14**(4), 768–791 (2002)
31. Masada, T., Takasu, A., Adachi, J.: Web page grouping based on parameterized connectivity. In: Lee, Y.-J., Li, J., Whang, K.-Y., Lee, D. (eds.) Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA), pp. 374–380. Springer, Heidelberg (2004)
32. Nanopoulos, A., Katsaros, D., Manolopoulos, Y.: A data mining algorithm for generalized Web prefetching. IEEE Trans. Knowl. Data Eng. **15**(5), 1155–1169 (2003)
33. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E **69**, 026113 (2004)
34. Ninan, A.G., Kulkarni, P., Shenoy, P.J., Ramamritham, K., Tewari, R.: Scalable consistency maintenance in content distribution networks using cooperative leases. IEEE Trans. Knowl. Data Eng. **15**(4), 813–828 (2003)
35. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford University (1999)
36. Pallis, G., Stamos, K., Vakali, A., Katsaros, D., Sidiropoulos, A., Manolopoulos, Y.: Replication based on objects load under a content distribution network. In: Proceedings of the IEEE International Workshop on Challenges in Web Information Retrieval and Integration (WIRI), Atlanta, 3–7 April 2006
37. Pallis, G., Vakali, A.: Insight and perspectives for content delivery networks. Commun. ACM **49**(1), 101–106 (2006)
38. Pallis, G., Vakali, A., Stamos, K., Sidiropoulos, A., Katsaros, D., Manolopoulos, Y. A latency-based object placement approach in content distribution networks. In: Proceedings of the IEEE Latin American Web Congress (LA-Web), Buenos Aires, 31 October–2 November, 2005
39. Pan, J., Hou, Y.T., Li, B.: An overview of DNS-based server selections in content distribution networks. Comput. Networks **43**(6), 695–711 (2003)
40. Qiu, L., Padmnanabhan, V.N., Voelker, G.M.: On the placement of Web server replicas. In: Proceedings of the IEEE Conference on Computer Communications (INFOCOM), vol. 3, pp. 1587–1596. IEEE, Piscataway (2001)
41. Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., Parisi, D.: Defining and identifying communities in networks. Proc. Natl. Acad. Sci. U. S. A. **101**(9), 2658–2663 (2004)
42. Sivasubramanian, S., Pierre, G., van Steen, M., Alonso, G.: Analysis of caching and replication strategies for web applications. IEEE Internet Computing **11**(1), 60–66 (2007)
43. Spertus, E., Sahami, M., Buyukkokten, O.: Evaluating similarity measures: a large-scale study in the Orkut social network. In: Proceeding of the ACM International Conference on Knowledge Discovery in Data Mining (SIGKDD), pp. 678–684 (2005)
44. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for Internet applications. In: Proceedings of ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), San Diego, 27–31 August 2001
45. Tajima, K., Hatano, K., Matsukura, T., Sano, R., Tanaka, K.: Discovery and retrieval of logical information units in Web. In: Proceedings of the ACM Workshop on Organizing Web Space, in conjunction with ACM Digital Libraries (1999)
46. Vakali, A.: Proxy cache replacement algorithms: a history-based approach. World Wide Web J. **4**(4), 277–298 (2001)
47. Vakali, A., Pallis, G.: Content delivery networks: status and trends. IEEE Internet Computing **7**(6), 68–74 (2003)
48. Venkataramani, A., Yalagandula, P., Kokku, R., Sharif, S., Dahlin, M.: The potential costs and benefits of long-term prefetching for content distribution. Comput. Commun. **25**(4), 367–375 (2002)
49. Wu, B., Kshemkalyani, A.D.: Objective-greedy algorithms for long-term Web prefetching. In: Proceedings of the IEEE Conference on Network Computing and Applications (NCA), pp. 61–68. IEEE, Piscataway (2004)
50. Wu, B., Kshemkalyani, A.D.: Objective-optimal algorithms for long-term Web prefetching. IEEE Trans. Comput. **55**(1), 2–17 (2006)
51. Zegura, E., Calvert, K., Bhattacharjee, S.: How to model an internetwork. In: Proceedings of the IEEE Conference on Computer Communications (INFOCOM), pp. 594–602. IEEE, Piscataway (1996)