Caching Techniques on CDN Simulated Frameworks

Konstantinos Stamos, George Pallis, and Athena Vakali

1 Introduction

It is evident that in the new Web era, content volume and services availability play a major role, leaving behind typical static pages which have solely text and images. The majority of the business oriented service providers are concerned for the Quality of Services (QoS), in terms of content delivery. In this context, proxy servers and Content Delivery Networks (CDNs) have been prosposed as different technologies, dealing with this concern. Their common goal is to bring content close to the users, reducing the response time.

Both technologies demonstrate different advantages and disadvantages. CDNs are characterized by robustness in serving huge amounts of requests and content volumes. However, their main shortcoming is that due to replication and distribution cost, replica placements should be static for a large amount of time. This leads to unoptimized storage capacity usage since the surrogate servers would contain redundant, possibly outdated, or unwanted content. On the other hand, proxy servers adapt content caching according to varying access patterns, using cache replacement algorithms. However, proxy servers do not scale well for serving large volumes of data or user populations. In an effort to combine the advantages of both, earlier recent work [2, 20, 29, 30] investigated different approaches that enable Web caching in CDNs, taking proxy servers' characteristics into account. As new caching ideas emerge, the need for a CDN testbed, suitable for performance evaluation and stress testing, becomes evident. Such a testbed should provide a

Konstantinos Stamos and Athena Vakali
Department of Informatics, Aristotle University of Thessaloniki, e-mail: {kstamos, avakali}@csd.auth.gr

George Pallis
Department of Computer Science, University of Cyprus, e-mail: gpallis@cs.ucy.ac.cy

networking environment incorporating CDN components, clients, traffic, and sufficient support for caching schemes deployment.

While the ideal case would be to examine caching schemes in real networks and CDNs, this is not always feasible or appropriate. Setting up a real CDN environment from scratch is unfeasible since it introduces high infrastructure cost. Moreover, its configuration is a cumbersome task because it involves many parameters (traffic patterns, link speeds, network topologies, and protocols). Incorporating a new caching scheme requires large scale modifications to the execution environments of the various network elements. Furthermore, commercial CDNs are of proprietary nature and they are not usually accessible for research purposes. Finally, it is not straightforward to carry out experimentation in a real world framework, since it involves uncontrollable events (such as random noise and external network traffic), rendering the experiments unreproducible.

To overcome the difficulties imposed by the real world models, one may build simulated models. A simulated model, in our case a Web caching enabled CDN, introduces a new set of challenges. Dealing with the model itself, balance between accurate real world model representation and reasonable resources management (execution times and memory consumption) must be achieved. Furthermore, the model should provide base for incorporating CDN components, clients, traffic, services, content types, and especially caching schemes. The variety of possible network configurations and diversity of the caching schemes impose a large tree of implementation cases. Therefore the best choice is to adopt an open architecture, by maintaining a reasonable level of abstraction in the simulated entities.

Currently, there is quite limited number of CDN simulation environments and there is no standard roadmap for a practitioner to design and implement such a complex environment. The motivation of this chapter originates to these difficulties which emphasize the need for developing widely available and open CDN simulation environments. More specifically, the core contributions of this chapter are:

- *To provide sufficient background* for issues related to Web caching in the context of CDNs;
- *To identify the simulation requirements* of a Web caching enabled CDN;
- *To analyze and model the simulation* of various caching schemes in an actual CDN simulator; and
- *To suggest a roadmap* for the practitioner who would like to clarify performance issues related to such simulated frameworks.

In summary, the main goal of this chapter is to offer a solid design methodology and share implementation experiences, while covering most of the topics related to Web caching in a CDN simulation framework.

The rest of this chapter is structured as follows: we start by presenting issues related to the content delivery in Web via CDNs and proxy servers. Then, the potential of integrating caching characteristics of both CDNs and

proxy servers are examined. A categorization of dynamic content along with several techniques are provided, followed by solutions to the problem of cache consistency. We continue with an in depth examination on how the mentioned caching schemes can be modeled and implemented in a simulated environment.

2 Content Delivery on the Web

Distributing information to users over the Internet in an efficient and cost-effective manner is a challenging problem. Web data caching and replication techniques have become key practices for addressing this problem, due to their ability to offer increased scalable solutions [25]. *Web caching* is mainly implemented by proxy servers, whereas *content replication* is the main practice on CDNs. Broadly speaking, the intention of Web caching and content replication is to shift the workload away from overloaded content providers and satisfy user requests from the intermediaries (proxy servers or CDN servers). Internet Service Providers (ISPs) use proxies to store the most frequently or most recently requested content. In addition, Web content providers may sign a contract with a CDN provider (e.g. Akamai) in order to offer their sites content over the CDN servers. In the following subsections, we overview the main characteristics of these two intermediary infrastructures for the Web.

2.1 Proxy Servers

Proxy servers are deployed by ISPs to deal with increased Web traffic and optimize the content delivery on the Web [33]. In particular, proxy servers act as an intermediator between users and content providers, serving user requests from local storage. Users make their connections to proxy applications running on their hosts. At each request, the proxy server is contacted first to find whether it has a valid copy of the requested object. If the proxy has the requested object and it is updated, this is considered as a cache hit; otherwise a cache miss occurs and the proxy must forward the request on behalf of the user. Upon receiving a new object, the proxy services a copy to the end user and keeps another copy to its local storage.

Thus, the intermediate caching of objects reduces bandwidth consumption, network congestion, and network traffic. Also, because it delivers cached objects from proxy servers, it reduces external latency (the time it takes to transfer objects from the origin server to proxy servers). Finally, proxy caching improves fault-tolerance because users can obtain a cached copy even if the remote server is unavailable or uncacheable.

On the other hand, using a shared proxy cache has three significant drawbacks: If proxy is not properly updated, a user might receive stale data, and, as the number of users grows, content providers typically become bottlenecks. Furthermore, caching is problematic in terms of not improving availability during "flash crowd" events. The third drawback is related to the limited system resources of cache servers (i.e. memory space, disk storage, I/O bandwidth, processing power, and networking resources).

The above problems stem from the fact that proxy servers have been designed to work on a local basis. Thus, when a proxy server cannot satisfy a user request (cache miss), it should connect with the underlying Web content provider in order to fetch the requested content. However, this may lead to Denial of Service (DoS), since Web content provider cannot serve a huge amount of requests (each Web content provider supports a limited number of HTTP connections). Moreover, the communication between a Web content provider and a proxy server may cause increased latency. For instance, consider the scenario where a user from Australia requests a Web page, and its Web content provider is located in USA. In such a case, a large number of TCP connections should be setup in order to communicate the proxy server with the content provider.

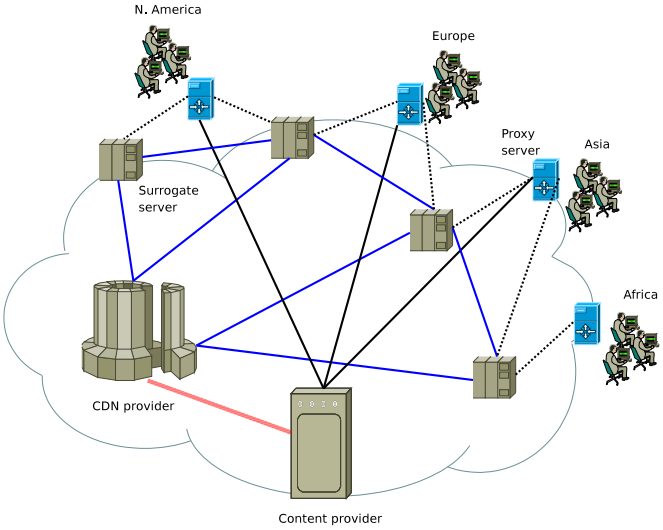

Fig. 1 Content Delivery on the Web.

2.2 Content Delivery Networks

Fig. 1 depicts how content is delivered on the Web using proxy and CDNs infrastructure. In case of cache misses, the proxy servers communicate with CDN servers in order to fetch the requested content. Specifically, a CDN maintains multiple *Points of Presence* (PoP) with Web server replicas (called *surrogate servers*) that store copies of the same content, and uses information about the user and the content requested to "route" the user request to the most appropriate site. The customers of a CDN are organizations that wish to offer their site content to a geographically distributed and potentially large audience. A CDN usually co-locates its surrogate servers within strategic data centers, using multiple network providers, on a globally distributed basis. Table 1 summarizes the main difference between proxy servers and CDNs. A comprehensive taxonomy with a broad coverage of CDNs in terms of organizational structure, content distribution mechanisms, request redirection techniques, and performance measurement methodologies can be found in Chapter 2 of this book.

Features	**Proxy Server**	**CDN**
Key practice	Web caching	content replication
Cached content	dynamically changes; content requested by users of an ISP	predefined content from the CDN-supported content providers
Scalability	low	high
Performance	vulnerable to flash crowd events	stable; suitable for resource-hungry applications (e.g. streaming media)

Table 1 Proxy Servers vs. CDNs.

3 Emerging Web Data Caching Techniques in CDNs

CDNs host distributed global information resources which are related to a large spectrum of applications. Users interact with (or within) companies, organizations, governmental agencies, and educational or collaborative environments. The popularity of the CDNs originates from its potential to efficiently deliver dynamic, distributed, heterogeneous, and unstructured data all over the world. Therefore, the need of various Web data caching techniques and mechanisms on CDNs has become obligatory towards improving information delivery over the Web.

3.1 Caching in CDNs

As we mentioned in the previous Section, Web caching and content replication have been developed as two distinct approaches in order to meet the increasing demand of user requests:

- *Web caching approach*: Proxy servers store the Web objects into their caches. However, the cached objects are determined by a cache replacement policy. The cache replacement policies refer to deciding which objects will evict from the cache to accommodate new objects. In such a policy, each object is defined by a "value", the so-called *cache utility value* (CUV). The objects with the smallest utility outcome will be the first candidates to evict from the cache. Podlipnig and Bszrmenyi in [23] conducted an extended survey of the existing cache replacement strategies.
- *Content replication approach*: Surrogate servers keep replicas of the Web objects on behalf of content providers. Contrary to proxy servers, the replicated content in CDNs remains static.

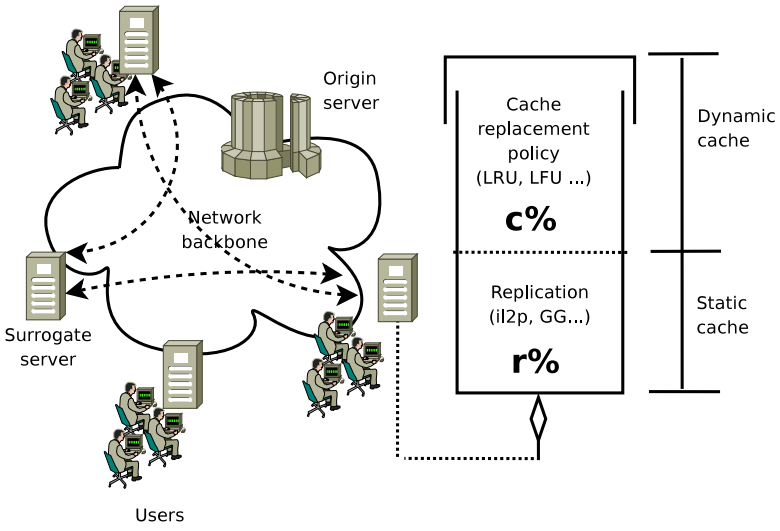

Fig. 2 Integrating caching in a CDN.

However, content replication practices of CDNs include inherent limitations. The major limitation is that a CDN infrastructure does not manage the replicated content in an efficient way. Moreover, replica placement is static for a considerable amount of time. The static nature of the outsourced content leads to inefficient storage capacity usage since the surrogate servers cache may contain unnecessary objects after a period of time. As a result, if

user access patterns change, the replicas in surrogate servers could not satisfy the user requests.

A solution to the above issue would be to integrate both caching and replication policies to the storage space of surrogate servers. The experimental results reported by Stamos et al. [30] show that an integration scheme outperforms the stand-alone Web caching and static content replication implementations.

To formally define the integration approach, consider a Web site representative W who has signed a contract with a CDN provider. The Web site contains N objects initially located only at the content provider (outside of the CDN). The total size of W is W^s and is given by the following equation:

$$W^s = \sum_{k=1}^{N} U_k^s \tag{1}$$

where U_k^s is the size of the k-th ($1 \leq k \leq N$) object.

Let M be the number of surrogate servers consisting the CDN. Each surrogate server M_i ($1 \leq i \leq M$) has a total cache size M_i^s dedicated for replicating the content of W. The original copies are located in the content provider. For simplicity, we consider that the surrogate servers are homogeneous (same storage capacity $M_i^s = M^s$ ($1 \leq i \leq M$)) and do not contain content from other Web sites.

As depicted in Fig. 2, the cache of surrogate server could be partitioned into two partitions:

- *Static cache partition*: Dedicated for static content replication. To formally define the static cache partition, we consider that its size is a percentage r ($r \in [0..1]$) of M^s. Therefore, the replicated objects, in static cache of a surrogate server, obey the following constraint:

$$\sum_{k=1}^{N} (f_{ik} U_k^s) \leq r M^s \tag{2}$$

  where f_{ik} is a function denoting whether an object k exists in the cache of surrogate server i. Specifically, $f_{ik} = 1$, if the k-th object is placed at the i-th surrogate server and $f_{ik} = 1$, otherwise. The content of the static cache is identified by applying a content replication algorithm. A wide range of content replication algorithms have been proposed in literature [12, 19, 21, 32, 37]. Kangasharju et al. [12] use four heuristic methods: 1) random, 2) popularity, 3) greedy-single, and finally 4) greedy-global. The experiments show that the greedy-global outperforms all other approaches. However, the greedy approaches are not feasible to implement on real applications due to their high complexity. Tse [32] study the content placement problem from another point of view. Specifically, the author presents a set of greedy approaches where the placement is occurred by balancing the loads and sizes of the surrogate servers. A quite similar approach

is also presented in Zhuo et al. [37]. Pallis et al. [21] present a self-tuning, parameterless algorithm (called Lat-cdn) for placing outsourced objects in CDN surrogate servers, which is based on network latency. Finally, in [19], Pallis et al. partition the content placement placement problem into two sub-problems. The first one defines the pairs of outsourced object - surrogate server which achieve the lowest latency. The second one determines which objects to replicate based on the users workload. This approach is called il2p.

- *Dynamic cache partition*: Reserved for Web caching using cache replacement policies. To formally define the dynamic cache partition, we consider that the size reserved for dynamic caching is a percentage c, ($c \in [0..1]$) of M^s. More specifically, the stored objects respect the following storage capacity constrain:

$$\sum_{k=1}^{N}(f_{ik}U_k^s) \leq cM^s \tag{3}$$

Initially, the dynamic cache is empty since it is filled with content at runtime according to the selected cache replacement policy. Thus, the surrogate servers would have the replicas with the best CUV in their dynamic cache partition. Other than the traditional cache replacement policies (e.g. LRU, LFU), Aioffi et al. [1] use an on-line heuristic algorithm in order to decide whether to add a new content replica or remove an existing one. The proposed algorithm (called on-line MDCDN) is based on a statistical forecasting method, called Double Exponential Smoothing (DES). Taking the user demand variations into account, MDCDN predicts the future demand at each surrogate server. These predictions determine the CUV of the the cached objects. Chen et al. [6] use an application-level multicast tree as a cache replacement policy for each CDN surrogate server. Presti et al. [24] determine the CUV of replicas by a non-linear integer programming formulation. In [3], Bartolini et al. decide whether to add a new content replica or remove an existing one using a semi-Markov decision process.

Given the above cache segmentation scheme, the percentages (r, c) must obey is the following:

$$r + c = 1 \tag{4}$$

The challenge for such an approach is to determine the surrogate server size which would be devoted to caching and replication as well. In other words, we should determine the percentages (r, c). Considering that this problem is NP complete [2], several heuristic approaches have been considered to efficiently integrate static and dynamic cache in CDN surrogate servers. Bakiras and Loukopoulos [2] propose a greedy *hybrid* algorithm that combines an LRU cache replacement policy with static content replication on a CDN. More specifically, initially the storage capacity of each surrogate server is reserved

for Web caching and at each iteration of the algorithm, objects are placed to surrogate servers maximizing a benefit value. The *hybrid* gradually fills the surrogate servers caches with static content at each iteration, as long as it contributes to the optimization of response times. Stamos et al. [29] have developed a placement similarity approach (the so called *SRC*) evaluating the level of integration of Web caching with content replication. According to this approach, a similarity measure is used to determine the surrogate server size which would be devoted to caching and replication. Finally, Pallis et al. [20] use a logistic sigmoid function in order to classify the surrogate server cache into two parts. The proposed approach, called R-P, classifies the replicas with respect to their quality values. In particular, the quality value of each replica is expressed by the users interest (increasing its value) or the lack of users interest (decreasing its value) for the underlying replica.

3.2 Caching Dynamic Content

Dynamic content can be classified into three categories, as depicted in Fig. 3, based on how frequently Web objects change and whether these changes can be predicted. The *periodic-update* category includes objects that the content provider updates at specified time intervals. For instance, consider a news Web page which is updated in every 5 minutes. The *on-demand-update* category consists of objects which are generated on demand and may have different attributes depending on the requesting user (e.g. the query forms). The *unpredictable-update* category includes objects that change unpredictably. The objects in *periodic-update* and *unpredictable-update* categories can be cached, whereas, the objects in the *on-demand-update* category are uncacheable.

Efficient distribution of dynamic content to end users is an important issue due to the growing number of dynamic data on the Web. A wide range of caching techniques have been proposed in order to accelerate the delivery of dynamic content to users [5, 27]. Fragment caching is an effective technique to accelerate current Web applications which usually generates heterogeneous contents with complex layout.

A fragment can be defined as a portion of a Web page which has a specific theme or functionality and is distinguishable from the other parts of the page. A Web page has references to these fragments, which are stored independently on the content provider or the surrogate servers. Challenger et al. [5] represent the relationships between Web pages and fragments by object dependence graphs.

The fragment-based approach has also been implemented in commercial CDN providers. For instance, the EdgeSuite network of Akamai is based on a fragment-based policy using the ESI (Edge Side Includes) specification accepted by the World Wide Web consortium. Specifically, the ESI specifi-

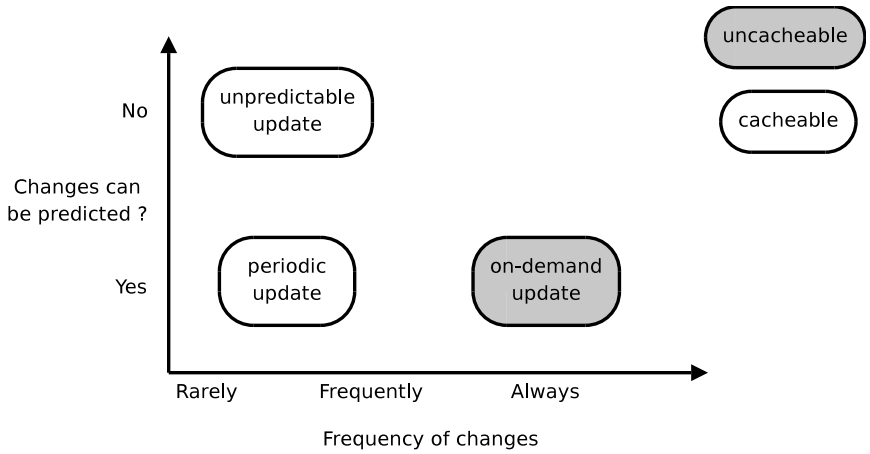

Fig. 3 Categorization of dynamic content.

cation defines an XML-based mark-up language for defining templates and identifying page fragments. A fragment-based policy is also used by the IBM Websphere [5], where the Web pages can be decomposed into a hierarchy of complex, atomic fragments.

Fragment-based approaches cannot be effectively applied on the objects which belong to the *on-demand-update* category, since these objects cannot be cached. Specifically, they perform well if the temporal locality of requests is high and if the underlying database is updated rarely. Applications that do not exhibit these behavior require more sophisticated techniques [28]. Therefore, instead of caching fragments of Web pages, another approach is to replicate a full copy of the application code at the surrogate servers [26]. In such an approach (known as *Edge Computing*), each surrogate server may connect with a centralized database. So, all database queries are forwarded to the content provider. Although this technique allows to distribute the computations to generate pages, it is limited by the latency incurred for each query, and by the throughput bottleneck of the origin database [28]. To address this issue, another approach is to keep a partial replica of the database (known as *Content-Aware Caching (CAC)* approach). In such an approach, the application programmers can choose the data replication that are best suited for the application. This approach can yield considerable gains in performance and availability, provided that the selected strategies are well suited for the application [28]. However, this is quite difficult since it requires significant insight of the application programmers in domains such as fault-tolerance and weak cache consistency. In this context, another technique (known as *Content-Blind query Caching (CBC)*) has been proposed to cache the results of database queries at the surrogate servers. Consistency of cached results must be maintained when the underlying database is updated. This technique allows to reduce the database query latency since a number of queries

can be answered locally. The total system throughput is also increased because less queries are addressed to the content provider [28].

3.3 Cache Consistency Mechanisms

Considering the dynamic nature of Web content, an important issue that must be addressed by CDNs is the consistency maintenance [36]. To prevent stale content from being transmitted to end users, the surrogate server must ensure that the locally cached data is consistent with that stored on servers. The exact cache consistency mechanism and the degree of consistency employed by a CDN depends on the nature of the cached data. Consequently, a CDN should ensure the consistency of replicas with the content provider by employing suitable mechanisms.

The problem of consistency maintenance has been well studied in the context of proxy servers. Particularly, in proxy servers the Time to Live (TTL) concept is widely used [26]. According to this, the content provider, when serving a cacheable object to the proxy, supplies an explicit TTL value. Then, the proxy considers that object valid during its TTL period. In the context of a CDN, the TTL concept should be employed in each individual surrogate server. In such a case, each surrogate server is responsible for maintaining consistency of data stored in its cache. Therefore, each one interacts with the content provider to do so independently of the other surrogate servers. However, this approach is impractical/unfeasible to be implemented in a large-scale network infrastructure. Considering that a typical CDN usually consists of a large number of surrogate servers (i.e. the Akamai - the leading CDN provider - has more than 25,000 surrogate servers around the world), the content provider will need to individually interact with a large number of surrogate servers. Thus, such an approach is not scalable from the perspective of the content providers.

To formally define the degree of consistency that a CDN can support, let CP_k^t and S_k^t denote the version of the object k at the content provider and the surrogate server respectively at time t. In this context, an object k is said to be:

- *Strongly consistent* with that at the content provider if the version at the surrogate server is always up-to-date with the content provider. That is, $\forall$t, $CP_k^t = S_k^t$. Strong consistency ignores network delays incurred in propagating updates to the surrogate server.
- *Delta consistent* with that at the content provider if the version at the surrogate server is identical for Δ time units, where Δ is a configurable parameter. That is, $\forall$t, $\exists\tau$ $0 \leq \tau \leq \Delta$ such that $CP_k^{t-\tau} = S_k^t$.
- *Weak consistent* with that at the content provider if the version at the surrogate server is not always up-to-date with the content provider.

A consistency degree may also be defined for multiple objects; it is known as *mutual consistency*. To formally define this degree of consistency, consider two objects a and b that are related to each other. Cached versions of objects a and b at time t (S_a^t and S_b^t respectively) are defined to be mutually consistent in the time domain (M_t-consistent) if the following condition holds: If $CP_a^t = S_a^{t_1}$ and $CP_b^t = S_b^{t_2}$ then $|t_1 - t_2| \leq \delta$, where δ is the tolerance on the consistency guarantees. For $\delta = 0$, it requires that the objects should have simultaneously existed on the content provider at some point in the past.

There exist a wide range of mechanisms [16, 17, 31] that have been used to provide efficient cache consistency in CDNs. These can be broadly categorized as follows:

- Server-driven consistency (also referred to as *server-based invalidation*): the content provider notifies the surrogate servers when the content changes. This approach substantially reduces the number of control messages exchanged between the content provider and the surrogate server since messages are sent only when an object is modified. However, this results in inefficient usage of the distribution network for content delivery and inefficiency in managing consistency at surrogate servers, since the content provider should maintain a list of all surrogate servers that cache the object. Several new protocols have been proposed recently to provide consistency using server-based invalidation. Web cache invalidation protocol (WCIP) [14] is one such proposal for propagating server invalidation using application-level multicast. Web Content Distribution protocol (WCDP) [31] is another proposal that enables server-driven consistency. Using the WCDP, the content provider can dynamically control the propagation and visibility of an object update. WCDP supports different levels of consistency (i.e. strong, delta, weak, and mutual).
- Client-driven consistency (also referred to as *client polling*): the updated version of a Web object is delivered to all the surrogate servers whenever a change is made to the object at the content provider. The advantage is that it does not require any list to be maintained at the content provider. However, such an approach may generate significant levels of unnecessary traffic if the objects are updated more frequently than accessed. Mikhailov and Wills [16] proposed a client-driven consistency approach, called MONARCH (Management of Objects in a Network using Assembly, Relationships and Change cHaracteristics). The MONARCH guarantees the cache consistency by collecting snapshots of content from sites of interest. This content is then used as input to a simulator to evaluate several cache consistency policies over a range of access patterns.
- Leases approach: Consistency is achieved by associating leases with each object that get replicated to surrogate servers. Specifically, lease is a time period where its duration denotes the interval of time during which the content provider agrees to notify the surrogate server if the object is modified. After the expiration of the lease, the surrogate server must send a message requesting renewal of the lease. This approach is a combina-

tion of server-driven and client-driven consistency. If the lease duration is zero, the cache consistency scheme degenerates into pure client-driven consistency. On the other hand, if the lease duration is infinite, the cache consistency scheme degenerates into a pure server-driven consistency. The concept of a lease was first proposed in the context of cache consistency in distributed file systems [10]. The use of leases for Web proxy caches was first presented in [15]. Duvvuri et al. in [8] present extensions to the HTTP protocol in order to incorporate leases. Ninan et al. [17] presented a variation of the lease approach for CDNs, called cooperative leases, by using Δ-consistency semantics. Specifically, Δ-consistency requires that a cached version of an object is never out-of-date by more than Δ time units with its server version. The value of Δ determines the nature of the provided guarantee. Therefore, the larger the value of Δ is, the weaker the consistency is.

4 Caching Techniques on CDNsim

This section is focused on introducing cache replacement policies in CDNs by using an actual CDN simulator. For this purpose we use CDNsim [1] as the main paradigm. First of all, the necessity of such a simulated environment is investigated, along with other simulation solutions. Then the requirements of a simulated cache, in terms of scientific issues and resource requirements, are defined. Finally, several issues related to the actual development of such caching framework are discussed.

4.1 The Need for CDN Simulated Environments

There have been several attempts to create private academic CDNs such as CoDeeN [18], Coral [9], and Globule [22] for research and every day purposes. However, the reproducibility of a given experiment is impossible since we are dealing with real networks. Moreover, it is hard to implement and evaluate new policies due to the required large scale alterations of the whole system. Consequently, the necessity of a simulation environment for performing experiments, still remains. Towards this direction, there have been several implementations of a simulated CDN [4, 7, 12, 34] which fit the individual needs of each research work. Most of them do not take several critical factors into account, such as the bottlenecks that are likely to occur in the network, the number of sessions that each network element can serve (e.g. router, surrogate server) and ignore the TCP / IP protocol. Finally, the most impor-

[1] http://oswinds.csd.auth.gr/∼cdnsim

tant disadvantage is the unavailability of a platform for examining caching techniques in CDNs.

Filling this gap, CDNsim is developed as a general purpose CDN simulator. It is extensible and open source, written in C++ using the OMNET++ and INET libraries [2]. It is a parallel discrete event trace driven network simulation package that provides libraries, utilities, and interfaces for content delivery on the Web. CDNsim models a CDN including basic network components such as users, surrogate servers, content providers, and routers. It takes the characteristics of Internet infrastructure into account by simulating the TCP/IP. CDNsim has been designed to support research in broad-coverage CDN services. It has also the ability to simulate Peer-to-Peer (P2P) services as well as various internetwork configurations. The experience gained from the development of such a tool is reported in the following paragraphs.

4.2 CDNsim's Caching Framework Requirements

This subsection includes the specifications of the surrogate servers' caches in CDNsim which are taken into account at design time, and both research and performance issues are addressed. A requirement is to support the integrated caching schemes as reported in these works [2, 20, 29, 30]. Cache consistency mechanisms must also be supported. Moreover, it is required to support complex content types such as video, and treat dynamic content by fragmentation. Finally, support for non cacheable content should be enabled.

The diversity of cache replacement algorithms leads to confusing branches of implementation cases. For instance, LFU and SIZE use different attributes for replacing objects. Therefore the detection of a common denominator is necessary. More specifically, given a set of primitive generic operations and content types one should be able to implement any flavor of the mentioned methodologies. Therefore, a strict requirement is to prepare a set of interfaces that can be used as building blocks of the various caching schemes. An appropriate exposure of the cache content to the network should be considered. This would enable both user access for downloading content and CDN access for management.

The execution of a CDN simulation includes high activity in the surrogate servers' caches. A typical simulation scenario involves a set of users performing requests for objects (Web pages, multimedia content, etc) to the CDN. The surrogate servers manage the content of their caches and attempt to satisfy the requests. By increasing the number of requests, the required CPU time (of the host running the simulation) is increased as well. Moreover, an increment to the caches' capacity leads to more objects being stored and thus to higher RAM requirements. It is evident that depending on the various

[2] http://www.omnetpp.org/

simulation scenarios and configurations the caches may become performance bottlenecks.

The primary performance concern is to optimize the cache function in terms of CPU requirements. In order to define a satisfactory performance threshold, the available operations of a cache need to be identified:

- *Search*: This operation involves the procedure of browsing though the cache's contents until a specified object is found. For instance, a user requests a Web page and the surrogate server examines the cache to check whether it is stored or not. The performance penalty of such operation depends on the organization of the content. Generally, the *search* complexity at an average case should always be better than $O(n)$, where n refers to the number of objects residing in cache. This is critical since the *search* operation may be executed several million times during a simulation, involving caches with several thousands of objects.
- *Insertion*: This operation refers to the procedure of inserting a new object in cache. It is responsible to maintain the proper organization of the cache and update other attributes such as the remaining storage space. Likewise, it must perform better that $O(n)$. Every cache replacement policy includes *insertion* operations as part of their algorithm. It is expected to be executed many times during a simulation and therefore it is an essential optimization parameter.
- *Deletion*: It is the procedure to remove a specific object from the cache. Every cache replacement algorithm replaces objects by performing *deletion* operations to free up storage space. Therefore, $O(n)$ performance should be upper bound.
- *Update*: The case of an object's update can be expressed as a combination of *deletion* of the previous version object and *insertion* of the updated object. The *update*. operation takes place when cache consistency is applied.

Summarizing, the cache speed is closely related to the content organization in the RAM of the host. Most of the cache replacement algorithms include the *Search-Insertion-Deletion-Update (SIDU)* operations. Therefore, the optimization requirement is the design of efficient content manipulation operations.

The secondary performance concern is the optimization of the memory footprint. It is important that the simulation's execution environment fits in RAM. Absolutely no memory must be swapped to the hard drive of the host, or else the execution performance will be reduced. Therefore, it is required to adopt a conservative design that saves memory. Simulating large networks with many surrogate servers, caches, and objects require several gigabytes of RAM. However, the memory optimization is usually a secondary requirement because the aforementioned problem can be easily solved with sufficient RAM.

4.3 CDNsim's Cache Architecture

In order to meet the previously discussed requirements, we define an architectural design. This design is used for the actual implementation of CDNsim. It is an effort to shape an abstraction to the real entities in a CDN. More specifically we model the cache organization of a CDN as a 3-level architectural component, depicted in Fig. 4.

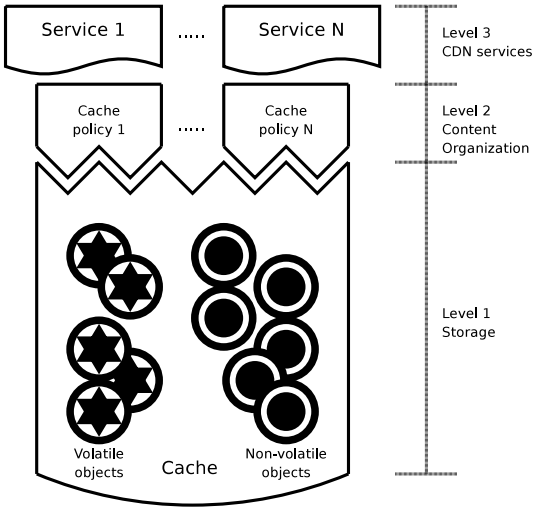

Fig. 4 3-level cache organization.

The first level deals with the notion of content by ignoring all the special characteristics that identifies it. *Cacheable content*, in general, is considered as raw *fragments*, namely objects. The objects are stored into the cache, which is merely a storage medium that keeps up-to-date information such as available storage capacity and provides the SIDU interface, as discussed previously. The objects can be classified into two categories: a) volatile objects and b)non-volatile objects. The first term refers to the objects that are stored inside the cache suggesting static caching, while the later defines the objects devoted for cache replacement algorithms. By flagging the objects, the integrated static and dynamic caching algorithms described in [2, 20, 29, 30] can be implemented.

One level up, we deal with the organization of the objects. The specific characteristics of each object are ignored. In this context, a set of cache replacement policies is defined, managing the content of the cache. Each policy maintains an ordering of the objects according to a set of attributes. The attributes may refer to objects' characteristics such as size, TTL, and last access time. For instance, the LRU cache policy should maintain a buffer

that keeps the objects sorted according to last access time, enabling the replacement of the objects. The upper level uses the SIDU interface which is provided by the cache replacement policies. At this point, we may introduce the concept of cache partitioning and especially *Static cache partition* and *Dynamic cache partition*.

The third level, defines a higher level logic of content management. In contrast to the lower levels, we are interested in the actual content type and special characteristics of the objects. Each content type effectively can be expressed as a group of objects (fragments) forming a hierarchy. For instance, a dynamic page can be expressed as a set of objects, representing identified cacheable page fragments. Therefore, the object abstraction of the lower levels provides a unified approach for dealing with the content diversity. Another high level construct is the service, which represents any operation at CDN level. The services can be of internal use only (e.g. surrogate server cooperation) and are available only to the CDN. Otherwise, the services are public, such as dynamic Web pages manifestation and serving to the users. Therefore, cache consistency can be addressed by implementing appropriate services that manage the cache content. Additionally, *uncacheable content (on-demand-update)* can be handled by implementing services capable of composing content on-the-fly.

Caching issue	**Architectural component**	**CDNsim default**
Static/Dynamic cache partitioning	Cache partition, volatile / non-volatile	Generic support/LRU
Strong/Delta/ Weak/Mutual consistency	Service	Strong
Complex content	Object hierarchy	Video, Web pages, etc
Unpredictable/periodic - cacheable dynamic content	Object	Generic support
On-demand-update - uncacheable dynamic content	Service	Unspecified

Table 2 Mapping of caching issues to CDNsim's architectural components.

Table 2 summarizes the mapping between various caching issues and the architectural components in CDNsim. Specifically, CDNsim supports directly the static, dynamic, and integrated caching scheme. Each can be modeled as partition of the cache. CDNsim offers generic input for any kind of static replica placement algorithm, while by default it supports the LRU dynamic caching. Cache consistency is managed by a set of services that signal the various content updates and commit the changes. Cache consistency lays at level-3. By default CDNsim implements strong cache consistency. Complex content types such as audio, video, Web pages, and streaming content are supported in the form of objects' hierarchies. Each type can be represented

by a set of objects that can be cached. Combined with cache consistency services we can enable the caching of dynamic content updates . Uncacheable content is dealt separately by implementing a set of specialized services at level-3.

4.4 Implementation Considerations

This subsection covers the actual implementation of a caching scheme in CDNsim, which can be of use to an actual software practitioner. Specifically, we use a representative example where the surrogate servers contain caches with partitioned storage capacity for static and dynamic caching, as reported by Stamos et al. [30]. The static part of the cache is filled using a replica placement algorithm while the dynamic part of the cache obeys to the LRU. We select LRU as it is a well known and easy to follow algorithm. Our goal is to implement such a cache by following the described architecture, while keeping up with the performance requirements.

Level 1 The primary concern in this level is to implement the classes object and cache. Since we plan to create a caching scheme that incorporates dynamic and static caching, a class of volatile objects, and a class for non-volatile objects is defined. We consider an object belonging to the non-volatile class, to be stored in the static part of the cache. Specifying an object as volatile leads to be stored at runtime in the dynamic part and potentially be removed by a cache replacement policy.

Low memory consumption is defined as a secondary requirement. An implementation approach needs to be followed that balances information compression and the ability to perform cache replacement:

- *Full compression - no information*: Bloom filters [13] are proposed as a method for simulating a cache. A bloom filter is a bitarray that packs information. In the context of a cache, this information refers to the ids of the objects. The operations permitted in the bitarray are: reading, enabling, and disabling bits. A set of different hash functions map each inserted id to a set of bits in the array. This approach has several advantages. The operation is fast, because it includes AND and OR operations, native to the CPU of the host and the memory consumption is low. However, the use of hash functions causes collisions. For different object ids the same bits may be suggested leading to inaccurate content description. Furthermore, the information related to each object is stripped. We cannot store attributes such as last access time and thus we are unable to implement cache replacement algorithms like LRU.
- *Partial compression - partial information*: As the name suggests, this approach makes partially use of the bloom filters technique and the full representation of the objects [13]. In full representation, each object is an actual C++ object that stores all the necessary attributes, like the size

and the last access time. However, the bloom filters result in information loss and thus LRU cannot be implemented.

- *No compression - full information*: The full representation is suitable for implementing LRU since all the objects' attributes are available. Consider the following example, we need 16 bytes (4 for the id, 8 for the last access time and 4 for the size, in a 32 bits environment) we still can store roughly about 130 million objects in 2 GB RAM. Therefore, despite the increased memory usage several millions of objects can be managed using a standard host. The use of lossless compression schemes is prohibited, because they involve time consuming decompression and re-compression leading to performance penalty. Therefore, the suggestion is to use a $1-1$ mapping of ids-objects. All the SIDU operations are $O(1)$.

Level 2 It manages the organization of the content in the form of cache policies. Two distinct cache policies are identified, the static caching and the LRU. The content of the static part of the cache remains unchanged by default, therefore, we are not required to maintain some kind of ordering in the stored objects. A $1-1$ mapping of the object ids to the objects will suffice and the SIDU performance is $O(1)$.

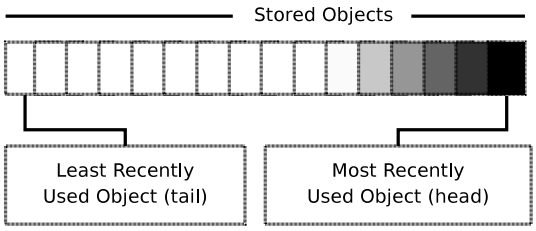

Fig. 5 The LRU buffer.

On the other hand, LRU requires special treatment. LRU removes the least recently used objects in favor of others most recently used. Therefore, It is necessary to enable an ordering of the objects according to the last access time. CDNsim uses a buffer containing objects sorted by the last access time, depicted in Fig. 5. This buffer does not store the actual objects, which is the responsibility of the cache itself. Instead, the buffer is used only as an ordering structure of object ids. The *search* operation requires $O(n)$ in the worst case, since we have to browse through the buffer to detect the required object. The *insertion* operation requires $O(1)$; the new object is inserted at the beginning of the buffer (head) and if necessary, several objects at the end of the buffer (tail) are being removed to free up storage capacity. The *deletion* operation requires $O(1)$ time. As long as the object for deletion is searched, we just crop the object from the buffer without the need for adjusting any ordering. Finally the *update* operation is also $O(1)$. Provided that we have searched the object, we just update without changing the ordering. It is evident that the *search*

operation is involved in most of the SIDU operations. The time complexity of the search in the worst case is $O(n)$ and may reduce the speed of the cache. However, this can be safely ignored for two reasons: a) this data structure can be cached out easily in the CPU cache, leading to small performance overhead and b) we tend to search for recently used objects, so only a few objects are being checked at the beginning of the buffer. Although in practice it gives satisfactory performance, we can further improve the *search* operation. This can be achieved by maintaining an extra index that points directly to the objects inside the buffer achieving $O(1)$ performance.

Another issue is the possible resource deadlocks and content inconsistency by accessing simultaneously the content. This is handled by creating private copies of the requested content. Therefore, the services of the upper level deal only with private copies of the content.

Level 3 At this level we are free from the caching issues, as they are handled by the lower level. The software practitioner is free to implement services that serve and distribute content.

4.5 Indicative Experimentation Results

The effectiveness of the storage partitioning scheme for Web caching and content replication is supported by a set of experiments conducted in this work [30]. In this subsection we demonstrate briefly a few results that capture the behavior of this scheme. The examined criteria are:

- *Mean response time*: This is the expected time for a request to be satisfied. It is the summation of all request times divided by their quantity. Low values denote that content is close to the end user.
- *Response time CDF*: The Cumulative Distribution Function (CDF) in our experiments denotes the probability of having a response times lower or equal to a given response time. The goal of a CDN is to increase the probability of having response times around the lower bound of response times.
- *Hit ratio*: It is defined as the fraction of cache hits to the total number of requests. A high hit ratio indicates an effective cache replacement policy and defines an increased user servicing, reducing the average latency.
- *Byte hit ratio*: It is the hit ratio expressed in bytes. It is defined as the fraction of the total number of bytes that were requested and existed in cache to the number of bytes that were requested. A high byte hit ratio improves the network performance (i.e. bandwidth savings, low congestion, etc.).

The tested caching schemes include the LRU, LFU, and SIZE algorithms at various levels of integration (r, c) with static replication. The (r, c), as already defined, represent the percentage of the storage capacity used for

static replication and Web caching respectively. The used static replication algorithm is il2p [19] which takes the server load into account. Specifically, il2p using two phases selects which object should be placed and where. During the first phase for each object the appropriate surrogate is selected minimizing network latency. Given the candidate pairs of (object, surrogate server), at the second phase, the one that yields the maximum utility value (depended on server load) is selected. This selection process is iterated until all caches are full. For completion, the cases of full mirroring (entire Web site is copied to the caches) and empty disks (simulating the absence of CDN) are included. Meeting the experimentation needs, the Stanford's Web site [3] is used. In this context a CDN was built using CDNsim, simulating several thousand users and network elements.

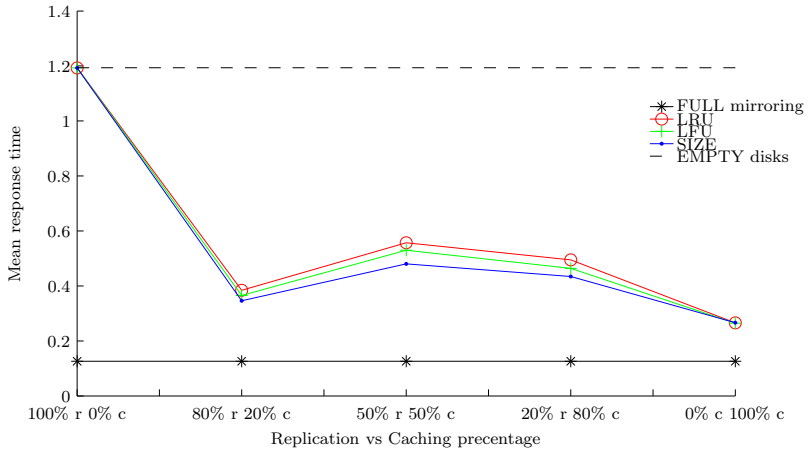

Fig. 6 Mean response time.

To begin with, Fig. 6 illustrates how the mean response time of the requests is affected by modifying the level of integration between static replication and caching. Using static replication only ($r = 100\%$) we receive the highest mean response time. This can be explained by the fact that a fixed replica placement cannot cope with the changing users' demands for content. Two distinct performance peaks can be identified; the first for $r = 80\%, c = 20\%$ and the second for $c = 100\%$. Increasing the dynamic partition of the cache only by 20% (first peak) leads to significant performance improvement. This behavior is logical since we keep a part of the cache open for new content to be stored, based on the users' requests, while maintaining a sufficient amount of static replicas suggested by il2p. As the percentage of the dynamic partition increases the good attributes of replication are gradually lost. This

[3] http://www.stanford.edu/~sdkamvar/research.html

is caused by the fact that the cache replacement policies may remove usefull content which otherwise would be strategically placed by il2p. The caching scheme (second peak) appears to perform slightly better than the integrated scheme (first peak). A possible explanation is that by letting the entire storage capacity to be used by cache replacement, we allow the Web caching scheme to adapt effectively to the user traffic patterns. Moreover, the fact that the Stanford's Web site contains mostly small objects, leads to low performance penalty during a cache miss. However, caching is not the choice, since the CDN is converted into a proxy server including all the scalability problems a proxy server imposes. Another important observation is that all the cache replacement algorithms follow the same behavior, with SIZE to be slightly better. SIZE's superiority can be explained by the fact that more room for new objects is available leading to better cache size utilization.

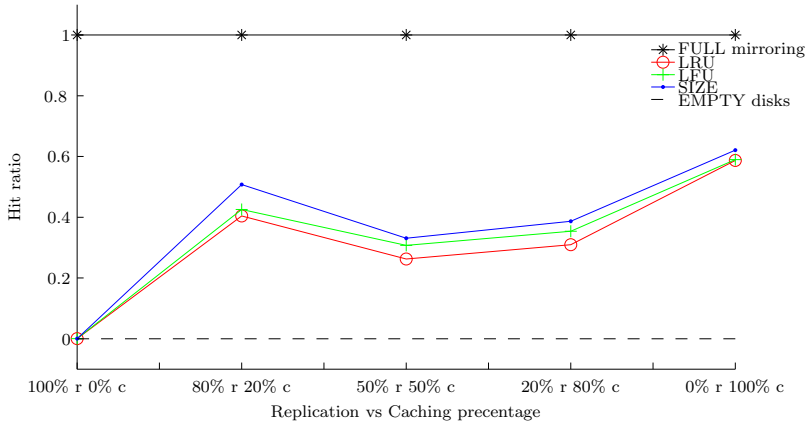

Fig. 7 Hit ratio.

In terms of hit ratio, depicted in Fig. 7, the same two peaks can be detected. Pure Web caching and the integrated schemes offer comparable results, while the static only replication does not keep up with. Fixed replica placement using the entire storage capacity suffers from low hit ratio since redundant content is outsourced and the placement is not optimal. The optimal placement cannot be achieved due to the changing users' requests pattern and the fact that it is a NP-complete problem. This reason also indicates why pure Web caching demonstrates slight performance superiority over the integrated scheme. The same behavior also exists in Fig. 8, illustrating the byte hit ratio.

A more accurate representation of the requests' satisfaction time distribution is presented in Fig. 9 for the first performance peak ($r = 80\%, c = 20\%$). Ignoring the ideal case of full mirroring, we observe that the integrated scheme (SIZE/il2p) outperforms (is the ceiling of all distributions) by achieving lower

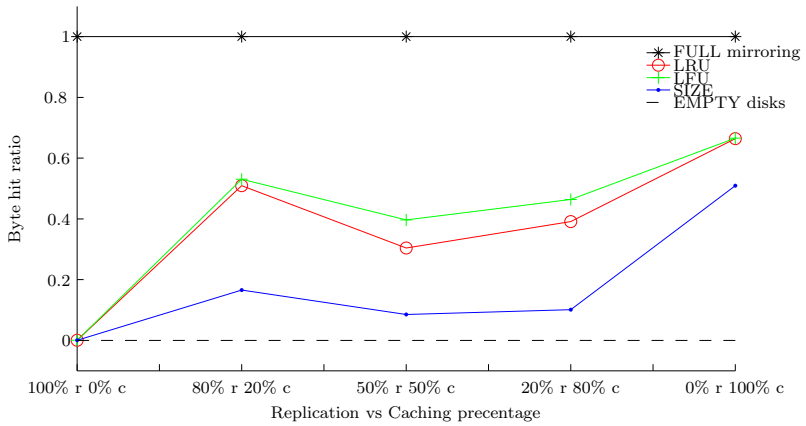

Fig. 8 Byte hit ratio.

response times than the other schemes. Here below we outline our observa-

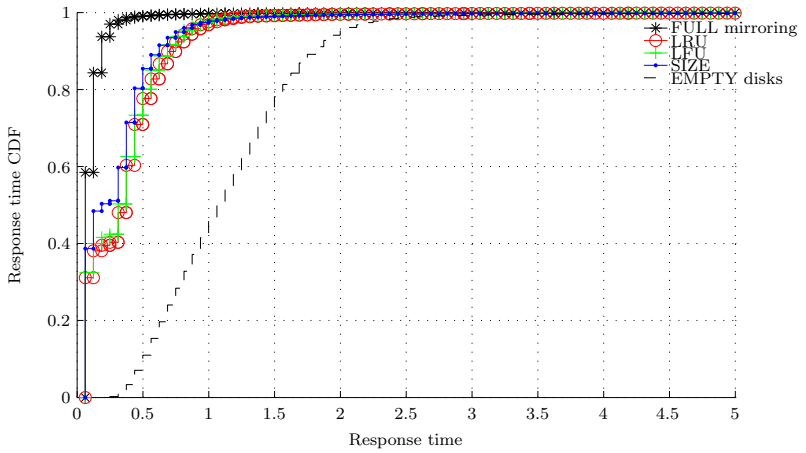

Fig. 9 Response Time CDF.

tions by summarizing the results:

- There exist at least one performance peak belonging to the integrated caching and replication scheme.
- The integrated scheme may demonstrate reduced performance because of an inefficient replica placement.

- In special caches the caching only scheme perform as good as the integrated scheme, but it is not recommended since it inherits all the disadvantages of the proxy servers.

The conclusion is that there is a realistic room for performance improvement by implementing Web caching characteristics in a CDN.

5 Visionary Thoughts for Practitioners

CDNsim is a free and open tool available both for research purposes and commercial use. Two main target groups can be identified where CDNsim could be of great use; the CDN developers and the software practitioners interested in CDNs. In this section, we discuss several visionary thoughts on how CDNsim can be used by these groups.

Dealing with the first target group, CDN providers are interested in maximizing the benefit of their network infrastructure. To achieve this, the CDN developers design proprietary algorithms that manage the content effectively. The natural derivative of such activity is the creation of a new product. In the context of CDNs, the product is a new content delivery service, like streaming video, large files delivery, etc. Although each service [4] may differ from the others in terms of functionality, a common set of periods in the life time of every service can be identified :

- *Before service release:* This period includes the development process of the service before its release to the users. CDNsim could be of use at the early development stages. It can be used to design and implement prototypes giving shape to the initial product ideas. Once the prototyping is done, it can be used to perform an *in vitro* evaluation of the performance and behavior under various network configurations and traffic patterns. CDNsim could significantly reduce the infrastructure investments during the stages of testing and prototyping until a certain level of maturity is reached. Then, evaluation is performed at the real CDN infrastructure. A real world example of the concept of prototyping and testing that could potentially be performed by CDNsim is the recent High Definition Video streaming by Akamai [5].
- *After service release:* The service, by the time of its release to the wider public, should have passed a set of testing suites. Additionally, there is a set of documented conclusions about its behavior and performance. However, as the product is being used under untested circumstances, the behavior may divert from the initial conclusions. CDNsim may be used to reproduce a problematic or unexpected situation aiding the analysts to explain *why*

[4] Using the term *service* we refer to a content delivery service in general.

[5] http://www.akamai.com/

an observed behavior is reached. Therefore, CDNsim could be used for continuous evaluation without disrupting the deployment of the service. Since the environment where a service runs is not static, CDNsim might act as a preventing mechanism of unwanted situations before they happen. For instance, the necessity of behavior prediction and disaster prevention is apparent before a worldwide broadcast of soccer world championship by Limelight Networks [6].

- *Service evolution in time:* Eventually a service will reach a certain level of maturity, stability, and correctness. However, the service's "habitat" (network configurations, typical user populations, current technologies) is constantly evolving. A representative example is the increment of fast internet connections and the fact that IPv6 [11] will become a necessity since the available IP addresses are reducing. CDNsim could be used to perform a *what-if* analysis. How the service scales with larger user populations? Can the service and the existing infrastructure keep up with much faster connections currently not available? These questions could be addressed by CDNsim by setting up the respective network configurations. Failing to predict the long term evolution could result in loss of clients by not investing on upgraded infrastructure in time.

Dealing with the second target group, software practitioners are encouraged to extend the existing architecture to support the latest trend of algorithms. A visionary evolution of CDNsim could be a testbed that incorporates a complete suite of caching algorithms used for performance comparison and testing. Moreover, CDNsim is able to run in parallel environments. The high performance computing researchers could find a testbed for implementing parallel algorithms in the context of CDNs. Therefore, some ideas concern the design and implementation of caching algorithms that take advantage of the new multi-core processors and the appliance of new more efficient data structures. Further research directions are outlined in the following section.

6 Future Research Directions

CDNsim might offer new perspectives for future research directions in the area of content delivery. Some indicative applications where CDNsim would be used as a simulation testbed could be the following:

- **Content Delivery Practices**: Several issues are involved in CDNs since there are different decisions related to where to locate surrogate servers, which content to outsource, and which practice to use for (selected content) outsourcing. It is obvious that each decision for these issues results in different costs and constrains for CDN providers. In this framework,

[6] http://www.limelightnet.com/

CDNsim can be used to evaluate a wide range of policies as well as to explore the benefits of caching in a CDN infrastructure.

- **Pricing of CDNs Services**: Pricing of CDNs' services is a challenging problem faced by managers in CDN providers. Deployment of new services, such as *Edgesuite*, are accompanied with open questions regarding pricing and service adoption. Chapter 8 addresses some pricing issues and presents some pricing models the context of CDNs. CDNsim can be used in order to validate them.
- **Mobile CDNs**: Content delivery on the mobile wireless Web is a topic of emerging interest and importance in the academic and industrial communities. Considering the recent advances in mobile content networking (e.g. WAP, IPv6 etc.), the infrastructure of mobile CDNs may play a leading role in terms of exploiting the emerging technological advances in the wireless Web. Chapter 14 presents mobile CDNs in details. CDNsim may be used as a testbed in order to address new research pathways in the area of mobile CDNs.
- **Peering of CDNs**: Peering of CDNs is gaining popularity among researchers of the scientific community. Several approaches are being conducted for finding ways for peering CDNs. However, several critical issues (i.e. When to peer? How to peer? etc.) should be addressed. Chapter 16 discusses some of these issues in detail. CDNsim may be used to simulate the peering CDNs framework under realistic traffic, workload, and replication conditions. It can also be utilized to evaluate the best practices and new techniques for load measurement, request redirection and content replication in the proposed framework for peering CDNs.
- **Security in CDNs**: The rapid growth of business transactions conducted on the Internet has drawn much attention to the problem of data security in CDNs [35]. In this context, secure content delivery protocols should be proposed in order to maintain content integrity (the delivered content which is modified by unauthorized entities should not be accepted) and confidentiality (the delivered contents cannot be viewed by unauthorized entities, including unauthorized proxies, and other users besides the requester) in CDNs. The high extensibility of CDNsim allows researchers to adapt the proposed protocols (e.g. iDeliver [35]) under its infrastructure.
- **P2P and Grid Technologies in CDNs:** Since CDNs are complex large-scale distributed systems, their development may be supported by the new emerging technologies of P2P and Grid. The successful exploitation and integration of these paradigms and technologies under a CDN infrastructure would provide an efficient way to cope with the aforementioned issues and would contribute significantly to the development of more efficient CDNs. The CDNsim architecture can easily enhance the aforementioned emerging technologies.

7 Conclusions

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, the explosive growth of the Web has imposed a heavy demand on networking resources and Web content providers. Users often experience long and unpredictable delays when retrieving Web pages from remote sites. CDN infrastructure seems to address the issues of capacity and performance on the Web in an efficient way. More and more Web content providers rely their content to be distributed by CDNs. The key to satisfy these growing demands lies in managing the content which is replicated in CDNs. Specifically, the need of various Web data caching techniques and mechanisms on CDNs has become obligatory towards improving information delivery over the Web.

In this chapter, we have summarized the emerging caching techniques which can be applied on CDN simulated frameworks. We study how to integrate caching policies on CDN's infrastructure. We also provide a comprehensive survey of the cache consistency mechanisms that can be applied on CDNs. Furthermore, we present the caching techniques which have been applied under CDNs for delivering dynamic content. Finally, we study these techniques under an analytic simulation tool for CDNs, the CDNsim.

To sum up, CDNs are still in an early stage of development and their future evolution remains an open issue. It is essential to understand the existing practices involved in a CDN framework in order to propose or predict the evolutionary steps. In this regard, caching-related practices seem to offer an effective roadmap for the further evolution of CDNs.

References

1. Aioffi, W.M., Mateus, G.R., Almeida, J.M., Loureiro, A.A.F.: Dynamic content distribution for mobile enterprise networks. IEEE Journal on Selected Areas on Communication **23**(10) (2005)
2. Bakiras, S., Loukopoulos, T.: Increasing the performance of cdns using replication and caching: A hybrid approach. In: IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), p. 92.2. IEEE Computer Society, Washington, DC, USA (2005)
3. Bartolini, N., Presti, F.L., Petrioli, C.: Optimal dynamic replica placement in content delivery networks. In: 11th IEEE International Conference on Networks (ICON 2003), pp. 125–130. Sydney, Australia (2003)
4. Bent, L., Rabinovich, M., Voelker, G.M., Xiao, Z.: Characterization of a large web site population with implications for content delivery. World Wide Web **9**(4), 505–536 (2006)
5. Challenger, J., Dantzig, P., Iyengar, A., Witting, K.: A fragment-based approach for efficiently creating dynamic web content. ACM Trans. Inter. Tech. **5**(2), 359–389 (2005)

6. Chen, Y., Katz, R.H., Kubiatowicz, J.: Dynamic replica placement for scalable content delivery. In: IPTPS, pp. 306–318. Cambridge, USA (2002)
7. Chen, Y., Qiu, L., Chen, W., Nguyen, L., Katz, R.H.: Efficient and adaptive web replication using content clustering. IEEE Journal on Selected Areas in Communications **21**(6) (2003)
8. Duvvuri, V., Shenoy, P., Tewari, R.: Adaptive leases: A strong consistency mechanism for the world wide web. IEEE Transactions on Knowledge and Data Engineering **15**(5), 1266–1276 (2003)
9. Freedman, M.J., Freudenthal, E., Mazi'eres, D.: Democratizing content publication with coral. In: 1st USENIX/ACM Symposium, vol. 2004
10. Gray, C., Cheriton, D.: Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. In: SOSP '89: Proceedings of the twelfth ACM symposium on Operating systems principles, pp. 202–210. ACM, New York, NY, USA (1989). DOI http://doi.acm.org/10.1145/74850.74870
11. Huston, G.: Ipv4: How long do we have? The Internet Protocol Journal **6**(4) (2003)
12. Kangasharju, J., Roberts, J.W., Ross, K.W.: Object replication strategies in content distribution networks. Computer Communications **25**(4), 376–383 (2002)
13. Kulkarni, P., Shenoy, P.J., Gong, W.: Scalable techniques for memory-efficient cdn simulations. In: WWW, pp. 609–618 (2003)
14. Li, D., Cao, P., Dahlin, M.: Wcip:web cache invalidation protocol. IETF Internet Draft (2000)
15. Liu, C., Cao, P.: Maintaining strong cache consistency in the world-wide web. In: ICDCS '97: Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97), p. 12. IEEE Computer Society, Washington, DC, USA (1997)
16. Mikhailov, M., Wills, C.E.: Evaluating a new approach to strong web cache consistency with snapshots of collected content. In: WWW '03: Proceedings of the 12th international conference on World Wide Web, pp. 599–608. ACM, New York, NY, USA (2003)
17. Ninan, A., Kulkarni, P., Shenoy, P., Ramamritham, K., Tewari, R.: Cooperative leases: scalable consistency maintenance in content distribution networks. In: WWW '02: Proceedings of the 11th international conference on World Wide Web, pp. 1–12. ACM Press, New York, NY, USA (2002)
18. Pai, V.S., Wang, L., Park, K., Pang, R., Peterson, L.: Codeen. In: Second Workshop on Hot Topics in Net-working (HotNets-II) (2003)
19. Pallis, G., Stamos, K., Vakali, A., Katsaros, D., Sidiropoulos, A.: Replication based on objects load under a content distribution network. In: ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06). IEEE Computer Society, Atlanta, USA (2006)
20. Pallis, G., Thomos, C., Stamos, K., Vakali, A., Andreadis, G.: Content classification for caching under cdns. In: Innovation on Information Technology. IEEE Computer Society, Dubai, United Arab Emirates (2007)
21. Pallis, G., Vakali, A., Stamos, K., Sidiropoulos, A., Katsaros, D., Manolopoulos, Y.: A latency-based object placement approach in content distribution networks. In: Third Latin American Web Congress (LA-Web 2005), pp. 140–147. Buenos Aires, Argentina (2005)
22. Pierre, G., van Steen, M.: Globule: a collaborative content delivery network. IEEE Communications Magazine **44**(8), 127–133 (2006)
23. Podlipnig, S., Böszörmenyi, L.: A survey of web cache replacement strategies. ACM Comput. Surv. **35**(4), 374–398 (2003)
24. Presti, F.L., Petrioli, C., Vicari, C.: Dynamic replica placement in content delivery networks. In: 13th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005), pp. 357–360. Atlanta, GA, USA (2005)

25. Rabinovich, M., Spatscheck, O.: Web Caching and Replication. Addison Wesley (2002)
26. Rabinovich, M., Xiao, Z., Douglis, F., Kalmanek, C.R.: Moving edge-side includes to the real edge - the clients. In: USENIX Symposium on Internet Technologies and Systems. Seattle, Washington, USA (2003)
27. Ramaswamy, L., Iyengar, A., Liu, L., Douglis, F.: Automatic fragment detection in dynamic web pages and its impact on caching. IEEE Trans. Knowl. Data Eng. **17**(6), 859–874 (2005)
28. 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)
29. Stamos, K., Pallis, G., Thomos, C., Vakali, A.: A similarity based approach for integrated web caching and content replication in cdns. In: Tenth International Database Engineering and Applications Symposium (IDEAS 2006), pp. 239–242. Delhi, India (2006)
30. Stamos, K., Pallis, G., Vakali, A.: Integrating caching techniques on a content distribution network. In: Advances in Databases and Information Systems, 10th East European Conference, ADBIS 2006, pp. 200–215. Thessaloniki, Greece (2006)
31. Tewari, R., Niranjan, T., Ramamurthy, S.: Wcdp: Web content distribution protocol. IETF Internet Draft (2002)
32. Tse, S.S.H.: Approximate algorithms for document placement in distributed web servers. IEEE Trans. Parallel Distrib. Syst. **16**(6), 489–496 (2005)
33. Vakali, A., Pallis, G.: Content delivery networks: Status and trends. IEEE Internet Computing **7**(6), 68–74 (2003)
34. Wang, L., Pai, V.S., Peterson, L.L.: The effectiveness of request redirection on cdn robustness. In: 5th Symposium on Operating System Design and Implementation (OSDI 2002)
35. Yao, D., Koglin, Y., Bertino, E., Tamassia, R.: Decentralized authorization and data security in web content delivery. In: SAC '07: Proceedings of the 2007 ACM symposium on Applied computing, pp. 1654–1661. ACM, New York, NY, USA (2007)
36. Yin, J., Alvisi, L., Dahlin, M., Iyengar, A.: Engineering web cache consistency. ACM Trans. Inter. Tech. **2**(3), 224–259 (2002)
37. Zhuo, L., Wang, C.L., Lau, F.C.M.: Load balancing in distributed web server systems with partial document replication. In: 31st International Conference on Parallel Processing (ICPP), p. 305. Vancouver, Canada (2002)