Chapter 4
Content Delivery and Management

Claudia Canali, Valeria Cardellini, Michele Colajanni and Riccardo Lancellotti

4.1 Introduction

The Web has evolved in the last decade from a mean to distribute content with marginal interest to a major communication media, where critical content and services are delivered to the users. This success was mainly driven by the concerns of content providers about the user-perceived performance of content delivery. When high availability, scalability, and performance are required, a common solution for content providers is to exploit third-party services to improve the performance of content and service delivery. The technical goal of Content Delivery Network (CDN) providers is to guarantee adequate delivery performance even when the incoming request load is overwhelming for the content provider alone.

CDNs have been originally proposed to primarily distribute static Web content and some limited streaming audio/video content over the Internet. First-generation CDNs were designed primarily to ease network congestion caused by the delivery of static Web pages through congested public peering points. However, the current context for content delivery is very different from that of the inception of these infrastructures, which date back to almost 10 years ago. Indeed, the Web scenario is currently characterized by an increased sophistication and complexity in delivered content. Modern Web contents are often dynamically generated and personalized according to the user preferences and needs. Traditional content delivery technologies designed for static content are not able to meet the new needs, as there are inherent challenges and limitations in the delivery of dynamic content that need to

Claudia Canali
University of Modena and Reggio Emilia, 41100 Modena, Italy, e-mail: claudia.canali@unimore.it

Valeria Cardellini
University of Roma "Tor Vergata", 00133 Roma, Italy, e-mail: cardellini@ing.uniroma2.it

Michele Colajanni
University of Modena and Reggio Emilia, 41100 Modena, Italy,
e-mail: michele.colajanni@unimore.it

Riccardo Lancellotti
University of Modena and Reggio Emilia, 41100 Modena, Italy,
e-mail: riccardo.lancellotti@unimore.it

R. Buyya et al. (eds.), *Content Delivery Networks*,

be overcome. In the last years, CDN started to evolve towards the support for the delivery of dynamically generated content, allowing the content providers to exploit the benefits of CDNs for modern Web applications and services.

This chapter explores the issues of content delivery through CDNs, with a special focus on the delivery of dynamically generated and personalized content. We describe the main functions of a modern Web system and we discuss how the delivery performance and scalability can be improved by replicating the functions of a typical multi-tier Web system over the nodes of a CDN. For each solution, we present the state of the art in the research literature, as well as the available industry-standard products adopting the solution. Furthermore, we discuss the pros and cons of each CDN-based replication solution, pointing out the scenarios that provide the best benefits and the cases where it is detrimental to performance.

The rest of this chapter is organized as follows. Section 4.2 provides some background material, presenting the logical layers of a Web system for delivering dynamic and personalized content and classifying the main caching and replication solutions for the logical layers. Sections 4.3, 4.4, and 4.5 discuss in details how the logical layers can be mapped over the nodes of a CDN in order to accelerate the delivery of static and dynamic resources. Section 4.6 discusses the issues related to the management and replication of user profile information over a distributed delivery infrastructure. Finally, Sect. 4.7 concludes the chapter with some final remarks and outlines the open research directions.

4.2 Systems for Web Content Delivery

In this section, we provide some background material on the architecture of the systems employed to generate and deliver Web content. We first review in Sect. 4.2.1 the logical layers of a multi-tier Web system; we then describe in Sect. 4.2.2 the architecture and the main functionalities of a CDN. Finally, in Sect. 4.2.3 we classify the caching and replication approaches for the logical layers of a Web system that can be employed by CDNs to accelerate the generation and delivery of dynamic and personalized content.

4.2.1 Logical Layers of a Web System

The large majority of modern Web systems is based on a multi-tier logical architecture, that separates the HTTP interface, the application (or business) logic, the data repository and, when existing, the user-related information for authentication and content personalization. These logical architecture layers are often referred to as *front-end*, *application*, *back-end*, and *user profile* layers [8]. Figure 4.1 shows the structure of a typical system providing Web-based services. We recognize the logical components of the system and the fundamental interactions between the layers.

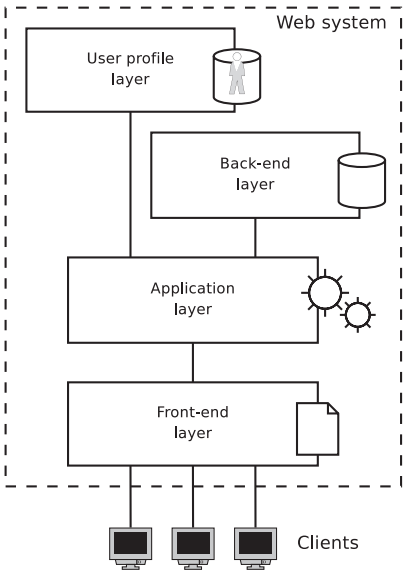

Fig. 4.1 Logical layers of a Web system

The front-end layer is the interface of the Web-based service. It accepts HTTP connection requests from the clients, serves static content from the file system, and represents an interface towards the application logic of the middle layer. The delivery of static content is a straightforward operation. A static Web content is typically stored in a file system and client requests for this type of content are managed by the HTTP server, which retrieves the resources from the file system and sends them back to the client in HTTP responses.

Examples of static content that are handled by the front-end layer are:

- *Web objects embedded in a Web page*. Typical embedded objects are images, style sheets, and active components such as flash animations, Java applets, and ActiveX controls.
- *Multimedia content*. Audio and video streams are static content handled by the front-end layer. To allow a smooth consumption of multimedia content by the client, the common approach is to rely on *HTTP streaming*, that is to divide the resources into chunks of data that are delivered in sequence to the client. The client can start the playback as soon as the first data chunk has arrived, without waiting for the delivery of the whole resource [26].
- *Page fragments*. They are portions of a Web page with a distinct theme or functionality [14]. Each fragment is considered as an independent information entity. For example, the page of a Web portal is typically described as composed by fragments such as latest news, feature articles, link bars, and advertisements. The use of fragments in the management of static content aims to improve the re-usability of Web content, because some fragments are common to multiple pages. However, when fragment-based management of static content is used, the front-end layer is also responsible for the *assembly* of fragments to build the Web page prior to its delivery to the user.

The application layer is at the heart of a Web-based service: it handles all the business logic and computes the information which is used to build responses with dynamically generated content. Content generation often requires interactions with the back-end and user profile layers: hence the application layer must be capable of interfacing the application logic with the data storage of the back-end and must be able to access the user profile when personalized content needs to be generated. Dynamic content is generated on-the-fly as a response to client requests. Examples of dynamic content generated by the application layer are:

- Answers retrieved from an organized source of information, such as the shopping cart page or searches in an e-commerce site.
- Web content generated dynamically to separate the content from its representation. For example, content management systems[1] or XML-based technologies [47] provide mechanisms for separating structure and representation details of a Web document. In these systems, the content (even when its lifecycle is relatively long) is generated dynamically from a template on-the-fly. The burden of dynamic data generation, that requires a computational effort due to data retrieval from databases, (optional) information processing and construction of the HTTP output, is outweighted by the convenience of handling data through software specifically designed to this aim, such as a DBMS.
- Web content generated by user social behavior. For example, the pages of forums or blogs provide an exchange place for messages written by the Web users.

The back-end layer manages the main information repository of a Web-based service. It typically consists of a database server and storage of critical information that is a source for generating dynamic content. If we refer to the examples of dynamic content generation from the application layer, we can identify the following data repositories:

- In the case of an e-commerce site, we use a database for storing the product lists, accessed for searching in the product catalog. Furthermore, also the user interactions are managed using a database for shopping cart status or list of purchases.
- In the case of a content management system, the dynamic generation of content accesses the database to retrieve both the Web page templates and the actual contents during the generation of Web resources.
- For Web sites such as blogs or forums, articles, comments, and posts are typically stored in a database.

The user profile layer stores information on the user preferences and context [16]. This information is accessed in the generation of dynamic content to provide personalized content. The information stored in the user profile may be originated from multiple sources such as:

- Information supplied by the user, that is usually provided through a fill-in form to add/edit user preferences. This profile communication may occur when the user registers itself for the access to a Web-based service or may be filled/modified later.

[1] For a list of the most popular CMS software, the reader may refer to http://www.cmsmatrix.org.

- Information inferred from the analysis of user behavior that is typically obtained from data mining of Web logs [20, 21, 27]. Typical examples of Web-based services that rely on information gathered through data mining are the recommendation systems for e-commerce [27] or the advertisements tailored on the user preferences.

4.2.2 A Simplified CDN Architecture

A CDN's architecture aims to achieve high performance and scalability by leveraging the principle of replicating the system resources (that is, the Web servers) to guarantee a high level of performance in the service of a huge amount of content. Replication occurs both at local and geographic level. In the case of local replication of system resources, the servers used for the service of user requests are tightly connected. They are placed on the same LAN and usually share a single upstream link connecting the system to the rest of the Internet. The common term to describe such a system is *cluster*. Servers within a cluster provide increased computing power thanks to the replication of system resources. They can interact in a fast and effective way [11]. Moreover, the replication may improve fault tolerance because a faulty node can be easily bypassed.

LAN-based systems have many pros, but they have scalability problems related to efficient generation and delivery of resources when the Web site is highly popular. The first problem that affects replication on a local scale is the so called first mile, i.e. the network link connecting the cluster to the Internet. This link can represent the system bottleneck for the end-to-end performance; moreover, it is a potential single point of failure. Traffic on the Web-based cluster zone, failures on an external router, and Denial-of-Service (DoS) attacks may cause the service to be unreachable independently of the computational power of the cluster platform. When better scalability and performance are needed, it is useful to replicate some elements of the infrastructure over a geographic scale.

A simplified view of a CDN's geographically distributed architecture is shown in Fig. 4.2. We distinguish two types of servers in a typical CDN, namely, *edge servers* and the *core server* [18, 36]. Edge servers are replicated on the so-called *network edge*, which means that these servers are as close as possible to the clients, typically in the Points of Presence (POP) of multiple Interner Service Providers (ISPs), and are mainly responsible for the interaction with clients. Client requests are typically diverted from the origin server to the edge server by means of DNS-based redirection [12, 36]. This approach is based on modified DNS servers that resolve queries for the site hostname with the IP address of a suitable edge server (the algorithm used to detect the most suitable edge server is usually complex and takes into account geographic and network distance, network link and edge server status).

The core server is a logical entity that handles the functions that are related to the management of the infrastructure, coordination of request distribution policies,

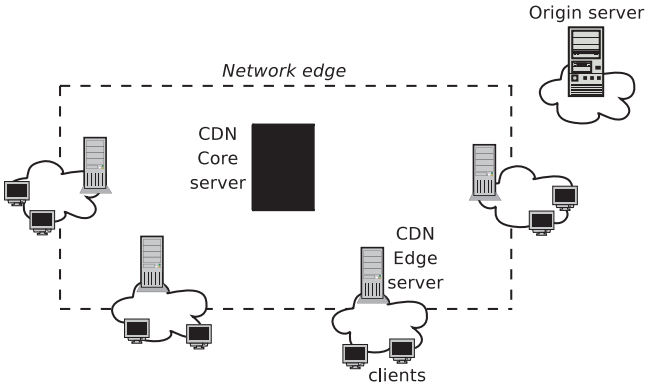

Fig. 4.2 A simplified CDN architecture

and billing. It can be implemented as a single powerful server or, more often, as a multi-cluster, that is a set of clusters that cooperate and behave like a single virtual computer with high availability and computational power.

4.2.3 Accelerating Content Generation and Delivery

The trend of Web evolution towards an ever increasing demand of scalable and high performance content delivery requires the content provider to rely on CDNs. On the other hand, CDNs should develop techniques to accelerate the delivery of content on behalf of the content provider.

To analyze how a CDN can accelerate the delivery of Web content and applications, we focus our attention on the origin server and on the edge servers, that are the elements of the Web infrastructure most involved in the content delivery process. The directions to address scalability and performance issues are the classical two: caching and replication. Indeed, CDNs replicate some logical layers of the origin server on their edge servers. Since we have four logical levels in the Web system, we envision four mapping approaches, as illustrated in Fig. 4.3.

- **Replication of front-end layer**. The edge server is responsible only for the management of static content. This approach is typical of the first generation of CDNs, where the edge servers, called *surrogate* servers, behave like reverse proxies to accelerate the delivery of content that can be stored at the file system level [36, 49]. The replicated Web content may be whole Web objects, for example when a CDN is used for delivering embedded objects or multimedia resources, or the replication may consider a more fine-grained approach, replicating Web fragments [14].
- **Replication of application layer**. A CDN is used to improve the delivery performance of dynamically generated content. This approach, called *edge computing*

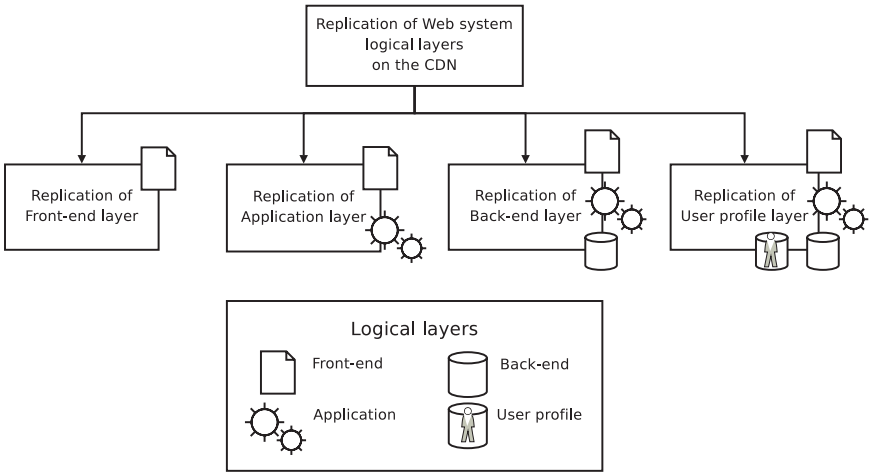

Fig. 4.3 Possible mapping of Web system logical layers on a CDN infrastructure

[44], moves Web application programs or components directly on the edge server [18, 37] with the aim of generating dynamic Web content close to the clients.

- **Replication of back-end layer**. The edge server provides both the functions for generating dynamic content and hosts data involved in the content generation. The origin server (or the core server of the CDN) is only responsible for the management of the infrastructure and acts as a master copy of the data.
- **Replication of user profile layer**. The edge server hosts also the data repository used for the generation of personalized content [42].

4.3 Front-End Layer Replication

The replication of the front-end layer aims to improve performance and scalability in the delivery of static content, as shown in Fig. 4.4. Such content is *cached* on the CDN edge servers. Moving the delivery of static content on the network edge addresses scalability issues, because it avoids the risk of network congestion in peering points and WAN links, that provides a major contribution to network-related delays [36].

Accelerating the delivery of static content using a third-party infrastructure is a common approach for improving the performance of content delivery, and dates back to the first generation of CDNs, such as the Adero CDN or the Akamai media delivery service [2]. However, delivery of this content is still a critical task, due to the growing amount of rich-media content [50] that is becoming a significant fraction of Web traffic. Moving the delivery of such media content close to the clients may have an important benefit for two reasons. First, due to the large size

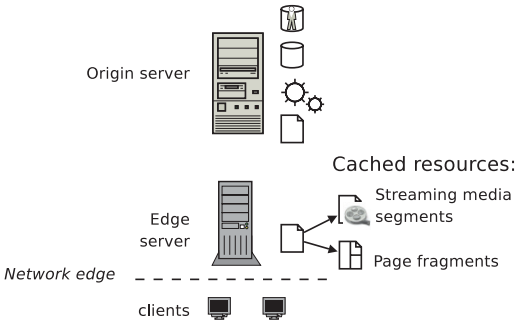

Fig. 4.4 Replication of the front-end layer on the edge server

of these content, network-related delays at peering points may have a significant impact on the user-perceived performance. Second, due to the techniques of HTTP streaming, which is commonly used, reducing the variance in delivery time results in smoother playback [26].

Due to the large size of multimedia content, it is common to cache on the edge server only the most popular fraction of each multimedia content instead of storing the whole resource (this is usually referred as *segment caching*) [15, 25], as shown in Fig. 4.4. The popularity of each fragment within a multimedia resource depends on the user access patterns. In the case of sequential access, the common approach is to rely on *sequential caching*, that is storing the first part of the resource to reduce buffering time. On the other hand, when the access patterns involve a significant amount of seek operation within the media, different caching techniques, such as *interleaved caching*, may be more effective [25].

The approach of dividing streaming content into segments has been proposed also for Web resources, in the case of the delivery of Web content assembled from *fragments* (represented among the cached resources in Fig. 4.4). This solution requires more effort from the edge server, because the front-end layer must include the functions for the separate caching and the assembly of fragments. Being an independent information entity, each fragment can have its own cacheability profile, which describes the ability to cache it and its Time-To-Live (TTL), thus allowing to manage the content freshness and lifetime at a fragment granularity rather than at the Web page level.

Upon a user requests the Web page (template), the edge server examines its cache for the included page fragments and assembles the page on-the-fly. Only those fragments either stale or deemed impossible to cache are fetched from the origin server. Therefore, using fragment-based caching and dynamic assembly on the edge servers, the origin server obtains two advantages: first, it does not have to assemble the page; second, it is typically required to deliver only a small fraction of the page, corresponding to stale or non-cacheable fragments. As regards the user-perceived performance, fragment-based caching has been proved to be effective in improving response time by serving most of the resources that comprise a dynamically generated page at the edge of the Internet, close to the end user [38, 51]. Furthermore, fragment-based caching has also beneficial effects on the edge servers. Indeed, it improves the disk space utilization because fragments that are shared across different

Web pages need to be stored only once; furthermore, it reduces the amount of invalidation at the edge server, because only those parts of the Web page that expire need to be invalidated.

The common standard for fragment-based caching is represented by Edge Side Includes (ESI) [19], which is an XML-based markup language that enables to distinguish via XML cacheable and non-cacheable content. The content provider designs and develops the business logic to form and assemble the pages by using the ESI specification within its development environment. Besides the primary functionality for including fragments within a page (even in a conditional way), the other key functionalities provided by ESI include the support for handling exceptions due to fragments unavailability and the support for explicit invalidation of cached fragments in such a way that it guarantees a stronger consistency than that provided by a TTL-based mechanism [19, 29].[2]

Fragment-based publishing and caching of Web pages have been adopted by companies and commercial products, including the EdgeSuite network of Akamai [2] based on the ESI specification and IBM's WebSphere Edge Server [28]. A large-scale deployment of a Web publishing system based on the fragment-based approach and compatible with ESI has been presented by Challenger et al. in [14]. This system is able to construct complex objects from fragments and has been developed to handle major sporting events at Web sites hosted by IBM. The authors also addressed the problem of detecting and updating all Web pages affected by one or more fragment changes. The proposal is to adopt different algorithms based on graph traversal that can be used according to the consistency requirements. A comparative study of four different offloading and caching strategies at the edge servers has been conducted by Yuan et al. in [51] using a representative e-commerce benchmark. Their results show that a simple strategy of offloading the functionality of composing Web pages from fragments can be very effective in terms of latency and server load reduction.

Most edge servers that support fragment-based caching do not provide any support for cooperation among the individual edge caches, i.e. these are treated as completely independent entities. This limitation does not allow to take full advantage of the potential caching capabilities of the edge servers that can be exploited through cooperation. Some effort toward this direction has been taken in the Akamai EdgeSuite network, which however includes only a limited cooperation among its edge servers. A recent work by Ramaswamy et al. [39] has addressed some significant challenges in designing a large-scale cooperative network of edge servers. Their proposal presents low-cost cooperative techniques based on dynamic hashing-based document lookup and update protocols and considers also how to provide failure resilience of individual edge servers.

The major drawbacks of the fragment-based solution are related to its applicability with respect to the type of dynamic content being delivered and to the task of fragmenting a Web page. Indeed, fragment-based caching can be effectively applied if the stream of requests is characterized by a high locality and if updates in

[2] For an analysis of cache consistency mechanisms in CDNs, the reader may refer to Chap. 5 of this book.

the content of the origin server are not frequent. This condition ensures that the fragment cacheability profiles are sufficient for managing the content freshness, thus relieving the origin server from the task to explicitly invalidate cached fragments. Furthermore, this technique suffers from lack of transparency, since caching, fragmentation, and assembling should be implemented on a per-application basis. For example, ESI requires a complete revision of the Web page code, because ESI code must be added over the original code, and its performance is dependent on the page structure. This manual identification and markup of page fragments is also hardly manageable for edge servers which deliver content from multiple providers. To overcome the manual fragmentation of Web pages, in [38] Ramaswamy et al. have proposed a scheme to automatically detect fragments in a Web page. Their approach depends upon a careful analysis of the dynamic Web pages with respect to their information sharing behavior and change patterns.

4.4 Application Layer Replication

A performance bottleneck in CDNs that replicate only the front-end layer is represented by the application layer in the origin server, which is responsible for the generation of dynamic content according to the Web application logic. Replication of application layer, commonly known as *edge computing* [18, 45], aims to improve the delivery of dynamically generated content by offloading this task from the origin server. The application code is replicated at multiple edge servers, while the data layer is still kept centralized. The computation is pushed to the edge of the network, as illustrated in Fig. 4.5.

In edge computing, each edge server has a full copy of the application code while the back-end layer is still centralized in the origin server, i.e. the edge servers

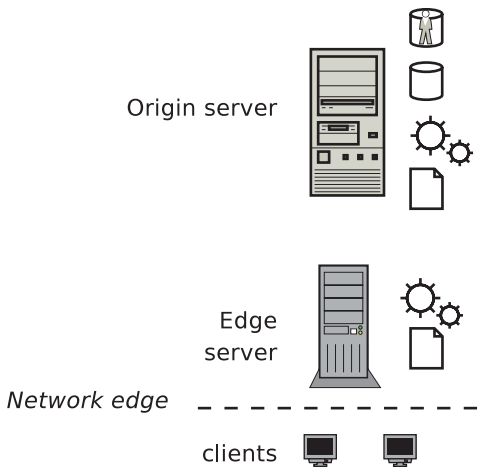

Fig. 4.5 Replication of the application layer on the edge server

continue to share a centralized database. By pushing the computation task to the edge of the network, the load on the origin server can be reduced and the CDN can achieve better efficiency and performance and higher availability with respect to the front-end only replication approach, where the application and data layers are centrally managed.

We can identify two architectural solutions depending on the ability of the edge server to distinguish between transactional and non-transactional requests. A transactional request is an atomic set of database operations that usually involve lock on part of the database and perform some update to the database records, while non-transactional requests have a read-only behavior on the data. If the edge server cannot distinguish the type of request, the Web server at the edge server forwards all requests to its local application layer, where they are executed. The local application logic then makes calls for database access to the centralized data layer located in the CDN core. Otherwise, if the edge server is able to distinguish between transactional and non-transactional requests, the edge server redirects only non-transactional requests to the local application layer, while transactional requests are directly forwarded to the application layer at the origin server, that then executes the transaction and accesses the centralized database.

In the application replication approach, the CDN core typically plays a coordinator role, being in charge for migrating and/or replicating the applications on the edge servers and keeping track of the replicas. It can be also responsible for maintaining the application replicas consistent with the primary copy. The CDN core may accomplish this functionality using a simple server-based invalidation that is, updating the application on the edge servers when the developer changes the primary copy.

Edge computing has been proposed and applied in a variety of commercial products and academic projects. For example, it is the heart of the EdgeComputing product from Akamai [3], which hosts customer-supplied J2EE components on edge-side application servers. Akamai EdgeComputing employs a two-level model for replicating the application layer: JSPs and servlets that contain the presentation logic are deployed on the edge servers of the Akamai network, while the business tier components that are tightly coupled with back-end applications or a database typically remain in the CDN core at the origin server.

Process migration issues have been addressed by many years of research; an example of complex system that could be employed in the CDN context is vMatrix [9], which migrates the entire dynamic state of the application from one server to another. However, Web applications do not require a real application migration at an arbitrary time but only at request boundaries [37]. Therefore, a significant simplification applicable in the CDN context is the automatic deployment of the application at the edge servers. ACDN (where the acronym stands for Application CDN) by Rabinovich et al. [37] is an application distribution framework that exploits this concept of automatic deployment; the application is dynamically replicated by the central coordinator on the basis of the observed demand. The framework implementation is based on a meta-file, which contains the list of the files comprising the application and an initialization script.

The DotSlash framework by Zhao and Schulzrinne [52, 53] is another academic project that adopts a dynamic script replication technique to manage dynamic content. DotSlash was not designed for large-scale CDNs, but it rather provides a system to handle sudden load spikes that affect the performance of Web sites through the dynamic provisioning of rescue servers which act as caching proxies.

The application layer replication is characterized for enabling the customization of concrete and specific applications. The application replication approach is neither generic nor transparent. Indeed, it requires customization on a per-application basis, because a manual configuration is needed to choose the components to be offloaded and where to deploy applications. For example, in ACDN [37], applications can be deployed and re-deployed dynamically, but manual administration is still involved, such as creating the meta-file for each application that has to be replicated. This application customization increases substantially the total cost of ownership, and it is prone to codification errors. Some effort for automatically deciding how to replicate Web applications has been proposed in [33]. However, these studies are mainly focused on a small scale scenario and may be not suitable for a large scale CDN, with tens of thousands of edge servers.

Further disadvantages of the application layer replication approach steam from keeping the data centralized at the origin server. This architectural choice determines two drawbacks. First, if the edge servers are located worldwide, as in large-scale CDNs, then each data access incurs a WAN delay; second, the centralized database may quickly become a performance bottleneck, as it needs to serve all database requests from the whole system. Therefore, the application replication solution is suitable only for those Web sites that require few accesses to the centralized database in order to generate the requested content.

The remaining approaches discussed in the next section aim to mitigate the centralized data layer bottleneck, which limits the overall CDN scalability. Therefore, the further steps in offloading the functionalities of the origin server to the edge servers exploit caching and replication techniques for the data layer.

4.5 Back-End Layer Replication

The edge computing approach may not solve every scalability problem, since in some Web applications the bottleneck lies in the back-end layer [13] instead of the application layer. In this case, scalability issues can be addressed by assigning to a third party (i.e. a CDN) the management of application data. A CDN provides answers to the queries of the application layer hosted by the edge servers on behalf of the back-end tier of the origin server.

The available solutions for replicating a data storage have been widely studied in the context of databases [23]. In this chapter, we will limit the scope of our analysis to the replication of data in the back-end layer of a Web system. In this scenario, the available approaches are summarized in [43]: the replication of the data stored in the back-end layer may be complete or partial, as illustrated in Fig. 4.6. The

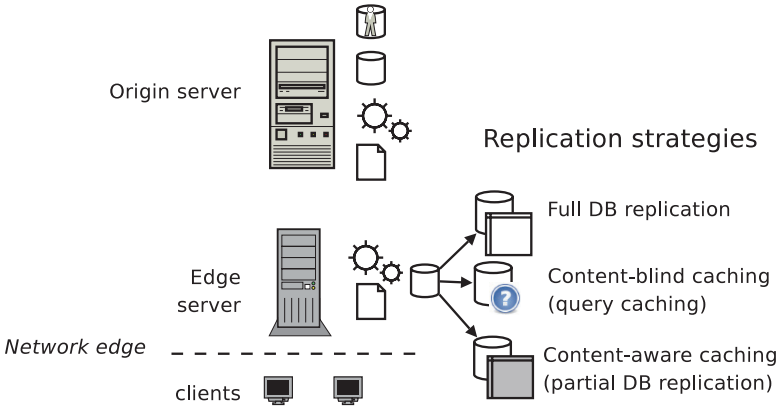

Fig. 4.6 Replication of the back-end layer on the edge server

partial replication of data can be obtained by exploiting a caching mechanism of the most popular queries to the data storage (*Content-Blind Caching*) or by actively replicating portions of the back-end data, selected on the basis of usage patterns, network, and infrastructure status [44] (*Content-Aware Caching*).

We anticipate that there is no clear winner among these alternatives, due to the different access patterns of the Web application to the database. Indeed, a work by Gao et al. [22] propose different replication strategies depending on the nature of the Web applications considered.

4.5.1 Content Blind Caching

When content-blind caching is adopted, edge servers cache the results of previous queries to the database. In such a way, the server may process locally future identical or similar queries, thus improving performance and relieving the load on the origin server back-end layer.

The approach of caching query results to replicate the back-end layer is highly popular. For example, the GlobeCGC [40] system explicitly aims to cache queries on the edge servers of a geographically distributed systems such as a CDN. Recently, the idea of dynamically replicating the back-end tier using a query cache to improve scalability has been proposed. For example, the QCache module of the DotSlash framework [53] proposes an agreement of cooperating Web sites that can temporarily enable a distributed query cache facility to alleviate the overload conditions in case of unexpected traffic surges.

The effectiveness of the query results caching depends on the achievable cache hit rate. To improve the amount of queries that can be serviced accessing the query cache, the characteristic of content blindness of the caching mechanism may be relaxed. To this aim, sophisticated query matching engines can be used so that a new

query can be answered using a union of already cached queries instead of contacting the origin server. Support for this enhanced query matching engine is a distinctive feature of DBproxy [4]. An efficient way to merge cached queries has been proposed in [30], where each query contributes to populate an (initially empty) copy of the original back-end database. DBCache [10] supports database caching at the level of tables, allowing to cache at the edge node either the entire content or a subset of the tables of the centralized database server.

Caching mechanisms should guarantee consistency of the cached data. Since a traditional TTL-based approach is not suitable for every Web application, some specific consistency enforcing mechanism has been proposed. For example, Olston et al. rely on a multicast messaging system to ensure that invalidation messages are sent to every query cache [32], while Tolia et al. [48] use hash functions to guarantee that no stale data are served from the cache.

The query-caching support on the edge server is an important feature that is available in multiple commercial products, including BEA WebLogic and IBM WebSphere. In particular, IBM WebSphere supports query caching through the so-called *Materialized Query Tables*. A materialized query table (MQT) is a table that materializes the pre-computed result of a query involving one or more tables. After the MQT is created and populated, an arbitrary subsequent query may be satisfied by the MQT, if the MQT matches all or a part of the query. A similar feature is provided by BEA WebLogic by means of EJBs.

Even if some consistency enforcing mechanism is adopted, the network layers on the geographic infrastructure can lead to hosting data at the edge servers that are stale with respect to the current state at the centralized data layer. This might not be a problem for read-mostly scenarios, where the Web applications do not need transactional semantics. However, for an important class of applications (e.g. when payment operations are involved) transactional semantics is a must and database updates are frequent. In these cases, query caching may not be a viable option. Furthermore, database caching techniques are suited only for those applications which repeatedly issue the same queries to the data layer. For applications which do not exhibit this temporal locality, it can be more efficient to replicate partially or entirely the data layer at the edge servers.

4.5.2 Content Aware Caching

In the case of content-aware caching, each edge server runs its own database server, which contains a partial view of the centralized database. The typical approach for partial data replication is to push section of the database close to the edge, according to access patterns. Since the aim is to improve the response time perceived by the end user, the algorithms for replica placement (such as HotZone) usually include network latency in the performance model [46].

A significant example of replication mechanism is provided by GlobeDB [43], that uses partially replicated databases based on data partition to reduce update traffic. However, this solution relies on one special server, which holds the full

database, to execute complex queries. Thereby, it may suffer from scalability because of the new throughput bottleneck represented by the special server. GlobeTB [24] improves the approach of GlobeDB with the goal of not only reducing the latency but also to increase the throughput of the replicated database. To this aim, GlobeTP relaxes the need for a single centralized master database, thus avoiding the risk of bottleneck in the origin server back-end.

As in the case of query caching, also partial database replication may suffer from consistency problems. Ganymed [35] explicitly addresses the issue of how to guarantee data consistency when the replicated back-end tiers are subject to changes (i.e. when update, delete or insert queries are issued). To this aim, Ganymed separates updates from read-only transactions, and routes updates to a main centralized database server and queries to read-only database copies.

The support for partial replication of databases is also available in multiple commercial products. For example, the mySQL DBMS supports a scheme for partitioning data among multiple replicas. Similar features have been also introduced into IBM DB2 and Oracle. However, in most cases partial replication schemes in databases are designed to manage a local replication of the resources (i.e. database clustering), and require a centralized manager that handles and distributes queries and transactions over the database partitions. This approach cannot be directly applied to the context of large-scale geographical replications, because the presence of a centralized manager would hinder the scalability of the system. For this reason, most commercial products rely more on query caching rather than on database replication schemes.

4.5.3 Full Database Replication

Full database replication maintains identical copies of the database at multiple locations. By moving a copy of the database to the edge servers and keeping the database copies coordinated among them, it becomes possible to completely deliver dynamic content at the edge of the network, without the need to modify each deployed application. However, the management of database replication introduces severe consistency management problems, that are particularly critical to solve when the client requests trigger frequent updates on persistent data. This is a well know issue that the database community has being addressed for a long time.

Traditionally, data replication is achieved through either *lazy* or *eager* write update propagation [23]. In the eager (or synchronous) approach, the coordination among the replicas occurs before the transaction commits, while in the lazy approach updates are only propagated after the transaction commits. The eager approach favors fault-tolerance and offers the same correctness guarantees as a single database. However, it suffers from severe limitations regarding performance and scalability that may render it impractical [23]. On the other hand, the lazy approach favors performance and scales very well; therefore, commercial replication products typically use it. However, the lazy approach introduces new problems, because transactions can read stale data and conflicts between updating transactions can be detected late, introducing the need for conflict resolution.

The simplest solution to manage database replication in Web environments is based on a centralized primary copy at the origin server and replicated secondary copies at the edge servers. Read-only transactions can be executed completely at the edge by accessing the local secondary database copy. However, for transactions that require updating operations (as in write-mostly scenarios), all database accesses are redirected to the database primary copy located at the centralized origin server. The primary database propagates any update to the secondary databases on a regular basis. A first drawback of this approach is that the edge servers must be aware of the application semantics, because it has to know whether a request triggers an update or a read-only transaction. Moreover, in this solution the consistency of the replicated data is maintained through a lazy update propagation scheme, which presents two negative effects. First, the data at the edge servers might be stale. Second, a crash might cause a data loss.

The exploitation of full database replication in the Web environment poses a number of challenging problems. Indeed, most database replication techniques proposed up to now assume that the database replicas are interconnected through a LAN. In recent years, the database community has proposed many replication protocols that provide both data consistency and good performance in LANs. As we focus on Web environments, we only mention some works that have addressed database replication in the context of locally distributed Web systems. The interested reader may also refer to [31] for a more comprehensive analysis on database replication systems based on group communications. A lazy replication solution that provides serializability and throughput scaling through the reduction of the number of conflicts has been proposed by Amza et al. in [5]; this earlier work has been improved through the introduction of distributed versioning, which provides strong consistency and avoids deadlock problems [6]. A recent work by the same authors investigates how to combine query result caching and cluster replication solutions [7]. A middleware tool that supports consistent and scalable data replication has been presented in [34].

In a CDN the database replicas are geographically spread in a WAN. If the Web application generates a significant number of database updates, a large amount of traffic may overload the wide-area network and impact negatively on performance, because each update needs to be propagated to all the other replicas to maintain the consistency of the replicated data. A performance analysis of data replication techniques that provide strong consistency in wide-area networks through group communications has been presented in [31]. However, the scalability analysis performed in this work is limited to eight replicas. Therefore, we can conclude that scalability and performance for database replication in WANs are largely an open issue that call for further research efforts.

4.6 User Profile Layer Replication

The user profile layer relies on a database for data storage, like the back-end layer. Hence, the possible solutions for replicating the user profile correspond to that

already described in Sect. 4.5. However, the access patterns for this layer are quite different if compared to the back-end layer.

In particular, the user typically interacts with only one edge server, hence the profile of a given user is accessed by one edge server for the whole duration of a user session. This access pattern has a significant impact on consistency and replication policies. Indeed, the whole dataset of user profiles can be partitioned and distributed over the edge nodes depending on the user access patterns. Since no replication is needed, consistency issues are limited to guarantee that the user profiles on the edge servers are consistent with the data on the origin server. The main approaches to manage the user profiles are therefore restricted to content blind or content-aware data caching, because full database replication is clearly unnecessary.

However, it is worth to note that, even if the user accesses only one edge server for the whole duration of its session, user migration among multiple edge servers may occur between consecutive session. Therefore, it is necessary to guarantee that the user profile data migrates following the user, as shown in Fig. 4.7. The support for this behavior is not explicitly optimized in most replication strategies for back-end data. Some proposals to handle this profile migration have emerged in the last years. CONCA [41] is a generic data caching framework that aims to support user mobility by allowing data to follow the user. This framework has been extended by the same authors to explicitly support the presence of personal data in Tuxedo [42].

Besides the replication of user-related information, a further critical operation that must be carried out by the user profile layer is the actual creation and update of such information. Currently, the user profile is either manually updated by the user through Web-based forms or is automatically updated by the Web system on the basis of the user behavior. The information stored in the user profile and the way to collect them depends on the Web-based services that are to be deployed. We present and discuss some significant examples of personalized content generation.

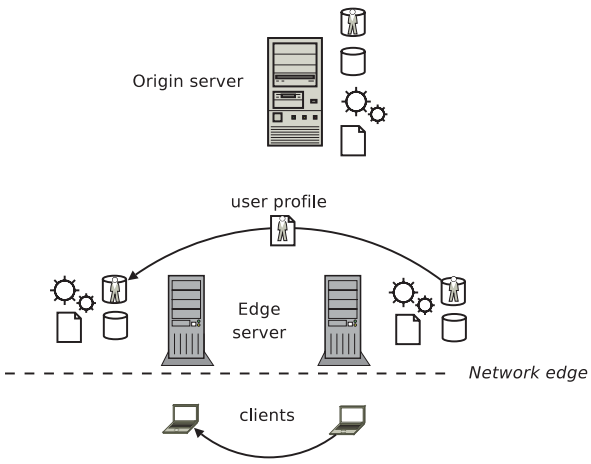

Fig. 4.7 Replication of the user profile layer on the edge server

- *Generation of personalized content through aggregation of external data sources*. This service is common to multiple personalized portals (e.g. myYahoo, iGoogle) and provides the user with a personalized news feed, typically retrieved from heterogeneous sources by means of XML-RSS feeds. The user profile contains information on which feeds are of interest for the users and about how the personalized content is to be presented. The users provide information about the subscription to news feeds and on the preferred presentation layout through filled-in forms during the registration to the personalized portal.
- *Collaborative filtering*. This type of service supports the interaction of users providing feedback on other users or topics. This type of personalized content generation is often used in recommendation systems that provide suggestions on goods to purchase, based on analysis of similar user behaviors, or to rank the reputation of a user in a social network. Information about the user is collected both through explicit user inputs (e.g. in the case where the user reputation is based on feedback from other users) or through implicit information collection, for example by mining the user purchases to cluster the user population according to pre-defined profiles [27].
- *Location and surrounding-based services*. These services generate personalized content on the basis of the user geographic location. The user position is determined through the analysis of data-related information or is explicitly supplied by the user when accessing the service. The user location is compared with geographic data and the generation and delivery of static and dynamic content (e.g. queries) is carried out according to the user location and surrounding, possibly combined with user preferences.

These examples show that, even if some information may be provided explicitly from the users, a significant fraction of the data stored in the user profiles are inferred through data mining of log files, cookies, and user click history. With the available information collection technologies, it is possible to extract interesting information related to the users including sensitive data, such as political, physical, and sexual features. Furthermore, most techniques are almost transparent to the users which are often completely unaware. Unauthorized users information collection occurred in the last years, for example, by the doubleclick.com commercial advertisement service. Several commercial services, including search engines, were associated with doubleclick.com. The commercial sites used cookies to monitor their visitors activities, and any information collected were stored in doubleclick.com databases. These user profiles were then used by doubleclick.com to select the advertisement banners more suitable for the users.

The examples of misusing personal information raised the interest towards the issues of whether and how to inform users about personal data collection. Concerns about privacy due to log data mining and cookie analysis [1] motivate the efforts of defining novel mechanisms to negotiate what information can be derived from user behavior and how they are to be used. The Platform for Privacy Preferences (P3P) [17] is an example of a proposal aiming to address this issue: each site compliant with the P3P standard must provide an XML-encoded description of which data are collected, how they are managed, and where they are stored. Full compliance

with the P3P standard imposes some restriction to the automatic replication of user profiles, because we must ensure that the adequate level of privacy is guaranteed for every replica of the user profile.

4.7 Conclusions and Open Issues

The delivery of static and dynamically generated content can be accelerated through a third party, i.e. a CDN infrastructure, that replicates some of the layers of a Web system. Throughout this chapter we have analyzed the replication of every logical layer composing a Web system. For each layer, we have discussed the research proposals in the field of content delivery and we have illustrated how the CDN industry is leveraging the replication to improve the effectiveness of content delivery. In particular, our analysis shows that replication of the front-end layer is suitable when the content provider aims to accelerate the delivery of static, possibly multimedia, content. When the CDN is used to accelerate the delivery of dynamic content, replication of the application layer is required. The achievable performance gain from this approach depends on the access patterns to the data, that may still determine a bottleneck in the back-end layer for some Web applications, thus forcing the replication of this latter layer also.

The research field in content delivery presents several open issues that are yet to be addressed. Indeed, even if some proposals to accelerate the delivery of dynamically generated content have been made and adopted by the industry, the effectiveness of the proposed solutions is still highly dependent on the access patterns of the applications. In particular, the risk of creating a bottleneck in the back-end layer is still one of the main issues that hinder the scalability of dynamic Web content delivery. This problem is likely to remain a major issue even in the next years, due to the evolution of Web content and applications. The Web 2.0 is shifting the Web towards two main trends: an ever-increasing amount of personalization, and the new Web usage patterns with large upload streams. Personalized (and uncacheable) content and high frequency of content refresh reduce the effectiveness of caching mechanisms and determine a growth in the overhead due to data consistency protocols. Furthermore, the presence of personal user information introduces bounds in the possibility of user profile replication, because the content provider must preserve the privacy of user sensitive information. The complexity of the scenario is further increased by the convergence of Web 2.0 with user mobility, that disrupts access locality due to the migration of users among the edge nodes. We believe that coping with this evolution will be the next challenge for CDN operators and researchers studying solutions for content delivery.

References

1. Agostini, A., Bettini, C., Riboni, D.: Loosely coupling ontological reasoning with an efficient middleware for context-awareness. In: Proc. of Mobiquitous 2005. San Diego, CA (2005)
2. Akamai: (2007). `http://www.akamai.com/`

3. Akamai EdgeComputing: (2007). http://www.akamai.com/html/technology/edgecomputing.html
4. Amiri, K., Park, S., Tewari, R., Padmanabhan, S.: DBProxy: A dynamic data cache for Web applications. In: Proc. of 19th IEEE Int'l Conf. on Data Engineering, pp. 821–831. Bangalore, India (2003)
5. Amza, C., Cox, A., Zwaenepoel, W.: Conflict-aware scheduling for dynamic content applications. In: Proc. of 4th USENIX Symp. on Internet Technologies and Systems (2003)
6. Amza, C., Cox, A., Zwaenepoel, W.: Distributed versioning: Consistent replication for scaling back-end databases of dynamic content web sites. In: Proc. of ACM/IFIP/Usenix Middleware Conf. (2003)
7. Amza, C., Cox, A., Zwaenepoel, W.: A comparative evaluation of transparent scaling techniques for dynamic content servers. In: Proc. of IEEE Int'l Conf. on Data Engineering (2005)
8. Andreolini, M., Colajanni, M., Mazzoni, F., Lancellotti, R.: Fine grain performance evaluation of e-commerce sites. ACM Performance Evaluation Review **32**(3) (2004)
9. Awadallah, A., Rosenblum, M.: The vMatrix: A network of virtual machine monitors for dynamic content distribution. In: Proc. of 7th Int'l Workshop on Web Content Caching and Distribution (2002)
10. Bornhovd, C., Altinel, M., Mohan, C., Pirahesh, H., Reinwald, B.: Adaptive database caching with DBCache. IEEE Data Engineering Bulletin **27**(2), 11–18 (2004)
11. Cardellini, V., Casalicchio, E., Colajanni, M., Yu, P.S.: The state of the art in locally distributed web-server systems. ACM Computing Surveys **34**(2) (2002)
12. Cardellini, V., Colajanni, M., Yu, P.: Request redirection algorithms for distributed web systems. IEEE Tran. on Parallel and Distributed Systems **14**(5) (2003)
13. Cecchet, E., Chanda, A., Elnikety, S., Marguerite, J., Zwaenepoel, W.: Performance comparison of middleware architectures for generating dynamic Web content. In: Proc. of 4th ACM/IFIP/USENIX Middleware (2003)
14. Challenger, J., Dantzig, P., Iyengar, A., Witting, K.: A fragment-based approach for efficiently creating dynamic Web content. ACM Transactions on Internet Technology **5**(2), 359–389 (2005)
15. Chen, S., Shen, B., Wee, S., Zhang, X.: Adaptive and lazy segmentation based proxy caching for streaming media. In: Proc. of ACM NOSSDAV (2003)
16. Colajanni, M., Lancellotti, R., Yu, P.S.: Scalable architectures and services for ubiquitous web access. In: Tutorial notes in 2006 World Wide Web Conf. (2006)
17. Cranor, L.: Web Privacy with P3P. O'Reilly (2002)
18. Davis, A., Parikh, J., Weihl, B.: EdgeComputing: Extending enterprise applications to the edge of the Internet. In: Proc. of 2004 World Wide Web Conf. (2004)
19. Edge Side Includes: (2007). http://www.esi.org/
20. Eiriniaki, M., Vazirgiannis, M.: Web mining for Web personalization. ACM Transactions on Internet Technology **3**(1) (2003)
21. Flesca, S., Greco, S., Tagarelli, A., Zumpano, E.: Mining user preferences, page content and usage to personalize Website navigation. World Wide Web **8**(3), 317–345 (2005)
22. Gao, L., Dahlin, M., Nayate, A., Zheng, J., Iyengar, A.: Improving availability and performance with application-specific data replication. IEEE Transactions on Knowledge and Data Engineering **6**(1), 106–120 (2005)
23. Gray, J., Helland, P., O'Neil, P., Shasha, D.: The dangers of replication and a solution. In: Proc. of ACM SIGMOD Int'l Conf. on Management of Data, pp. 173–182 (1996)
24. Groothuyse, T., Sivasubramanian, S., Pierre, G.: GlobeTP: Template-based database replication for scalable Web applications. In: Proc. of 2007 World Wide Web Conf. (2007)
25. Guo, H., Chen, S., Xiao, Z., Zhang, X.: DISC: Dynamic interleaved segment caching for interactive streaming. In: Proc. of the 25th International Conference on Distributed Computing Systems (2005)
26. Guo, L., Chen, S., Xiao, Z., Zhang, X.: Analysis of multimedia workloads with implications for Internet streaming. In: Proc. of 14th Int'l World Wide Web Conf. (2005)
27. Ho Ha, S.: Helping online customers decide through Web personalization. IEEE Intelligent systems **17**(6) (2002)

28. IBM WebSphere Edge Server: (2007). http://www-3.ibm.com/software/Webservers/edgeserver/
29. Iyengar, A., Ramasvamy, L., Schroeder, B.: Techniques for efficiently serving and caching dynamic Web content. In: S. Chanson, X. Tang, J. Xu (eds.) Web Content Delivery. Springer (2005)
30. Larson, P., Goldstein, J., Guo, H., Zhou, J.: MTCache: Mid-tier database caching for SQL server. IEEE Data Engineering Bulletin **27**(2), 35–40 (2004)
31. Lin, Y., Kemme, B., Patiño-Martínez, M., Jiménez-Peris, R.: Consistent data replication: Is it feasible in WANs? In: Proc. of Europar Conf. (2005)
32. Olston, C., Manjhi, A., Garrod, C., Ailamaki, A., Maggs, B., Mowry, T.: A scalability service for dynamic Web applications. In: Proc. of Innovative Data Systems Research, pp. 56–69. Asilomar, CA (2005)
33. Pacifici, G., Spreitzer, M., Tantawi, A., Youssef, A.: Performance management of cluster based Web services. IEEE Journal on Selected Areas in Communications **23**, 2333–2343 (2005)
34. Patiño-Martínez, M., Jiménez-Peris, R., Kemme, B., Alonso, G.: Consistent database replication at the middleware level. ACM Transactions on Computer Systems **23**(4), 1–49 (2005)
35. Plattner, C., Alonso, G.: Ganymed: Scalable replication for transactional Web applications. In: Proc. of ACM/IFIP/USENIX Int'l Middleware Conf. Toronto, Canada (2004)
36. Rabinovich, M., Spatscheck, O.: Web Caching and Replication. Addison Wesley (2002)
37. Rabinovich, M., Xiao, Z., Aggarwal, A.: Computing on the edge: A platform for replicating Internet applications. In: Proc. of 8th Int'l Workshop on Web Content Caching and Distribution. Hawthorne, NY (2003)
38. Ramaswamy, L., Iyengar, A., Liu, L., Douglis, F.: Automatic fragment detection in dynamic Web pages and its impact on caching. IEEE Transactions on Knowledge and Data Engineering **17**(6), 859–874 (2005)
39. Ramaswamy, L., Liu, L., Iyengar, A.: Scalable delivery of dynamic content using a cooperative edge cache grid. IEEE Transactions on Knowledge and Data Engineering **19**(5), 614–630 (2007)
40. Rilling, L., Sivasubramanian, S., Pierre, G.: High availability and scalability support for Web applications. In: Proc. of 2007 IEEE/JSP Int'l Symp. on Applications and the Internet. Washington, DC (2007)
41. Shi, W., Karamcheti, V.: Conca: An architecture for consistent nomadic content access. In: Proc. of Workshop on Caching, Coherence, and Consistency. Sorrento, Italy (2001)
42. Shi, W., Shah, K., Mao, Y., Chaudhary, V.: Tuxedo: A peer-to-peer caching system. In: Proc. of 2003 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications (2003)
43. Sivasubramanian, S., Alonso, G., Pierre, G., van Steen, M.: GlobeDB: Autonomic data replication for Web applications. In: Proc. of 14th Int'l World Wide Web Conf. Chiba, Japan (2005)
44. 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)
45. Sivasubramanian, S., Szymaniak, M., Pierre, G., van Steen, M.: Replication for Web hosting systems. ACM Computing Surveys **36**(3) (2004)
46. Szymaniak, M., Pierre, G., van Steen, M.: Latency-driven replica placement. IPSJ **47**(8) (2006)
47. The Apache Cocoon project (2007). http://cocoon.apache.org/
48. Tolia, N., Satyanarayanan, M.: Consistency-preserving caching of dynamic database content. In: Proc. of 16th Int'l World Wide Web Conf., pp. 311–320 (2007)
49. Vakali, A., Pallis, G.: Content delivery networks: Status and trends. IEEE Internet Computing **7**(6) (2003)
50. Williams, A., Arlitt, M., Williamson, C., Barker, K.: Web workload characterization: Ten years later. In: S. Chanson, X. Tang, J. Xu (eds.) Web Content Delivery. Springer (2005)
51. Yuan, C., Chen, Y., Zhang, Z.: Evaluation of edge caching/offoading for dynamic content delivery. IEEE Transactions on Knowledge and Data Engineering **16**(11) (2004)

52. Zhao, W., Schulzrinne, H.: DotSlash: Handling Web hotspots at dynamic content Web sites. In: Proc. of IEEE Global Internet Symposium. Miami, FL (2005)
53. Zhao, W., Schulzrinne, H.: Enabling on-demand query result caching in DotSlash for handling Web hotspots effectively. In: Proc. of Int'l Workshop on Hot Topics in Web Systems and Technologies. Boston, MA (2006)