Cluster Comput (2010) 13: 1–17
DOI 10.1007/s10586-009-0094-y

A Two-level distributed architecture for the support of content adaptation and delivery services

Claudia Canali · Michele Colajanni · Riccardo Lancellotti

Received: 12 September 2008 / Accepted: 25 June 2009 / Published online: 22 July 2009

Abstract The growing demand for Web and multimedia content accessed through heterogeneous devices requires the providers to tailor resources to the device capabilities on-the-fly. Providing services for content adaptation and delivery opens two novel challenges to the present and future content provider architectures: content adaptation services are computationally expensive; the global storage requirements increase because multiple versions of the same resource may be generated for different client devices. We propose a novel two-level distributed architecture for the support of efficient content adaptation and delivery services. The nodes of the architecture are organized in two levels: thin edge nodes on the first level act as simple request gateways towards the nodes of the second level; fat interior clusters perform all the other tasks, such as content adaptation, caching and fetching. Several experimental results show that the Two-level architecture achieves better performance and scalability than that of existing flat or no cooperative architectures.

Keywords Content adaptation · Multimedia resources · Distributed architectures · Performance evaluation

C. Canali · M. Colajanni (✉) · R. Lancellotti
Department of Information Engineering, University of Modena and Reggio Emilia, Modena and Reggio Emilia, Italy
e-mail: michele.colajanni@unimore.it

C. Canali
e-mail: claudia.canali@unimore.it

R. Lancellotti
e-mail: riccardo.lancellotti@unimore.it

1 Introduction

During the past decade, Web contents have evolved from mostly text and small size images to increasing percentages of personalized and multimedia resources [21, 26]. Recent studies [20, 33] have also evidenced that the popularity of Web-logs (Blogs) has dramatically augmented the demand for large images and audio/video clips, that are mainly delivered through HTTP streaming [21, 43]. Accessing these resources through the Web places high requirements on the client platform, in terms of processing power, display size, storage resources, and network bandwidth. This issue is exacerbated by the growing popularity of Web-enabled mobile devices, such as handheld PCs, personal digital assistants (PDAs) and smart phones, that are characterized by more limited computational and display capabilities than a traditional computer. Providing users with contents that can be accessed from everywhere and through any device requires the content provider architecture to carry out on-the-fly adaptation that can tailor the resources to the capabilities of the client device (e.g., display size, memory), of the client interconnection (e.g., protocols, wired/wireless networking), and of the user preferences.

Content adaptation involves a large and highly heterogeneous set of transformations that are typically obtained through computationally expensive chains of operations [6]. Another consequence of adaptation is that from the original set of stored resources, multiple versions may be generated. This significantly increases the size of the working set that may augment of one order of magnitude with respect to the traditional Web. For these reasons, supporting efficient content adaptation and delivery services is a complex task that requires intervention at the software level, modification of existing protocols, and even accurate design of the server infrastructures that represent the focus of this paper. Any

content provider should address the high computational cost of the adaptation and the augmented storage requirements due to the presence of multiple resource versions, hence the most practicable solution is to rely on intermediary architectures consisting of multiple geographically distributed servers interposed in the path from the client to the origin server [5, 17, 23, 27]. This type of architectures opens the possibility of sharing the load of computationally expensive services, increases the storage capacity and improves content delivery.

The main contribution of this paper is the proposal of a novel architecture for the support of on-the-fly content adaptation and delivery services. The server nodes are organized in two specialized levels: the edge level of thin nodes is located on the network borders; the interior level consists of clusters that are located in well connected Internet regions. The differentiation of the node functionalities and the use of a content partitioning at the level of the interior clusters are two innovative features of the Two-level architecture. The specialization of the nodes simplifies the management of the architecture and improves load balancing by assigning the most expensive tasks (in terms of computation and storage) to the most powerful clusters of the interior level, while the edge level nodes are devoted to light tasks, such as gateway operations. Exploiting content partitioning allows the provider architecture to limit the increment of the working set size due to the presence of adapted resource versions. The motivation is that content partitioning has the effect of preserving request access locality and avoiding content replication. The proposed architecture is innovative because it addresses present and future issues for scalability and performance that are not solved by available solutions based on geographically distributed architectures [5]. We compare the Two-level architecture with alternative flat architectures through several experiments based on prototype systems. We demonstrate that our proposal achieves better performance and scalability than the flat architectures for different workloads and scenarios. Moreover, we show that the Two-level architecture guarantees robust performance and low sensitivity to network delays and to highly skewed resource popularity.

The rest of this paper is organized as follows. Section 2 discusses alternative designs of intermediary distributed architectures to support content adaptation and delivery services. Section 3 presents the proposed Two-level architecture, while Section 4 describes the flat architecture that is the best existing alternative. Section 5 introduces the performance metrics for the evaluation of the distributed architectures and describes the experimental setup. Section 6 compares the performance of the Two-level and flat architectures. Section 7 contains a sensitivity analysis with respect to network conditions and resource popularity skewness and analyzes the scalability of the architectures with respect to different workloads and architecture sizes. Section 8 discusses the related work and Section 9 concludes the paper with some final remarks.

2 Design of architectures for content adaptation and delivery services

The discussion on the design choices of systems for content adaptation and delivery is related to architectures composed by intermediary servers located between the origin servers and the clients. This choice is popular for two reasons. First, it allows to improve delivery performance by replicating the resources close to the user. Second, the deployment of the architecture can exploit existing intermediary servers, such as access points or proxies, interposed in the path from a (mobile) client to an origin server.

When we examine the main tasks that are necessary to support content adaptation and delivery services, we should consider that any distributed architecture can make a different choice about where to execute these tasks.

2.1 Main tasks

We identify five main tasks that are necessary to an intermediary distributed architecture to support content adaptation and delivery services. We refer to logical tasks that may be implemented by one or multiple processes and may be located on one or different nodes.

Edge (E). This is the front-end component of the intermediary architecture that is contacted by the client. A process receives the client request, identifies the client requirements and activates the discovery task. Once the requested resource is obtained from some node, the edge task delivers it to the client. (Just the edge task is visible to the client; the other tasks and the overall architecture are transparent to the client.)

Discovery (D). This task has the goal to find one or more nodes that may hold a valid copy of the requested resource and fetch it from that node.

Cache (C). This task should provide a multi-version caching service that is able to manage original and adapted resources.

Fetch (F). When the requested resource is not found in any node of the intermediary distributed architecture, the fetch task retrieves the resource from the origin server.

Adaptation (A). This task represents the core of the content adaptation mechanism. When needed, it transforms the original resources in a form that matches the client requirements. Other details are described below.

The adaptation task may involve different chains of transformations depending on the adaptation purpose and on the

type of resource (image, audio or video). For example, in order to decrease the download time of a client with limited available bandwidth, the content adaptation task can reduce the size of the original resource. Such transformations can be obtained by reducing the quality factor of an image or by recoding an audio/video stream at a different bit rate. Another transformation that may reduce the resource size is the conversion between different formats and compression standards (e.g., from BMP to JPEG for an image, from WAV to MP3 for an audio file, different parameters of MPEG encoding for a video).

Content adaptation can be required to match the limited computing and display capabilities of most current mobile devices which are not suitable for high quality decoding and displaying. These transformations may include different operations, such as reducing the frame size and color depth of images and video, or decreasing the frame rate of video streams. It is worth to note that adaptation oriented to the device characteristics has also the effect of reducing the size of the adapted resources [6].

From a computational point of view, content adaptation represents the most expensive task and different adaptations can involve response times of different orders of magnitude. For example, modifying the quality factor of a large JPG image may require hundreds of milliseconds, while the format conversion of a short video clip (few minutes of length) may take up to several seconds [6]. The type of transformation is not the only factor that determines the computational cost of the adaptation task, that depends also on the initial quality level of the resource. We should consider that multiple versions of the same resource may be generated through the adaptation task and cached in some nodes of the distributed architecture. Hence, the required resource can be obtained not only from the original version, but also from an already adapted resource that may be further transformed to obtain a less detailed (or lower resolution) version. In this paper, we assume that each version of a resource can be obtained from any higher quality version. However, our proposal may handle any relation scheme between the multiple versions of the same resource and the allowed transformations among them. These relation schemes are usually represented through an *adaptation graph*, as described in [27]. The presence of multiple versions of the same resource opens novel issues because it multiplies the working set, and forces the architecture and its caching mechanism to manage multi-versioning through some efficient solution, because, even in the case where each adapted resources is smaller than the original content, the overall working set size is multiplied by the number of versions that each resource may have.

2.2 Mapping tasks over multiple nodes

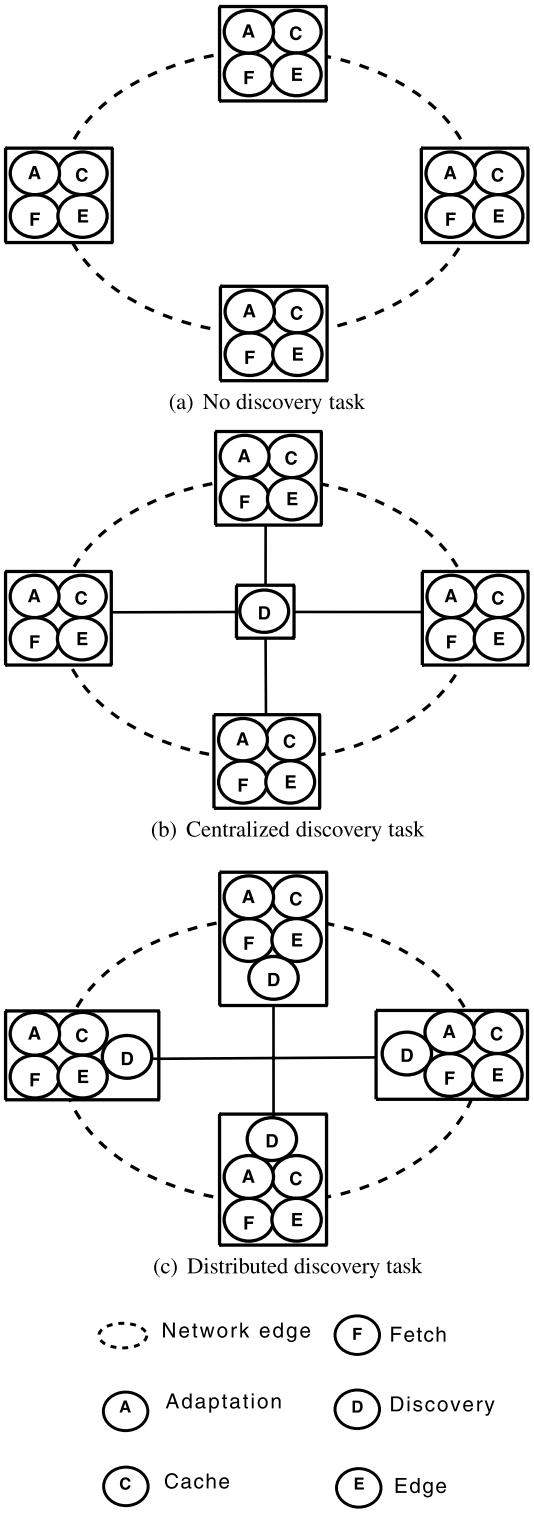
Once defined the main tasks, the design of an intermediary distributed architecture requires to determine how to map the tasks over the nodes of the system. In this section, we consider some mapping alternatives suggested by popular architectures for Web content delivery that have been proposed in literature [5, 18, 35, 39, 40] and then we present our alternative in Section 3. We can consider that the proposed solutions are based on architectures where multiple nodes are located on the network edge. Each node performs *edge*, *adaptation*, *cache* and *fetch* tasks. The main difference concerns the management of the *discovery* task, where we can identify three main schemes:

- a non cooperative architecture that does not provide any *discovery* task;
- a cooperative architecture with a centralized *discovery* task;
- a cooperative architecture with distributed *discovery* tasks.

Figure 1 outlines the three mapping alternatives. The boxes represent the nodes of the architecture, that are placed on the network edge. The circles indicate the tasks (each task is denoted by a capital letter) that are executed by a node. We analyze the pros and cons of each scheme to determine a convenient way to map the tasks over the nodes of a distributed architecture that should support efficient adaptation and delivery of Web and multimedia contents.

The simplest alternative is represented by a non cooperative architecture where the nodes operate individually for the generation, adaptation and delivery of contents. As shown in Fig. 1(a), every node performs the same four tasks, while the *discovery* task is not necessary. As there is no cooperation among the nodes, either the resource is found on the contacted edge node or is fetched from the origin server. Independent nodes located on the network edge have been widely used in the context of Web caching and content delivery [39, 40], since the user perceived response time can be reduced when the requests are served by nodes that are close to the clients. However, the lack of cooperation among the nodes leads to a limited cache hit rate and a limited scalability in the context of content adaptation and delivery services [39]. The high computational cost of the adaptation tasks, the explosion of the working set due to multiple resource versions and the non uniform distribution of the client requests among the edge nodes can easily exhaust the computational and storage capabilities of a stand-alone node, that cannot take advantage of already adapted resources on other nodes.

A first cooperation among the nodes exploits a centralized *discovery* task, as shown in Fig. 1(b). An example of this scheme has been proposed in the context of Web caching [18, 35]. This architecture introduces a first differentiation among the nodes: the edge nodes perform *edge*, *adaptation*, *cache* and *fetch* tasks; a central node performs the *discovery* task by storing a complete directory of all



(a) No discovery task

(b) Centralized discovery task

(c) Distributed discovery task

Fig. 1 Mapping tasks over the nodes of the distributed architecture

cached resources. The edge nodes cooperate through the central node that can perform an accurate resource discovery thanks to its complete knowledge of node caches. Unfortunately, this scheme presents some drawbacks if applied to a geographically distributed architecture for content adaptation and delivery services. A client request may be redirected to a close or very far node because this node is selected by the centralized *discovery* task on the basis of its cached resources. The immediate consequence is that the variance of the user-perceived response time augments significantly. A second issue is that the central node represents a system bottleneck in a highly replicated system with hundreds or thousands of nodes. The same proponents of this scheme observe that a similar architecture may limit in practice the geographical expansion of the system [18].

An alternative solution that preserves node cooperation and avoid bottlenecks is represented by architectures exploiting distributed *discovery* tasks. As an example, we consider an architecture where every node carries out all tasks (Fig. 1(c)). These systems where all nodes are identical peers are typically referred to as *flat* architectures. We describe some of their details in Section 4, because they represent the state of the art of intermediary distributed architectures for content adaptation and delivery of Web and multimedia contents.

3 Two-level architecture

In this section we describe the innovative *Two-level* architecture for content adaptation and delivery services. The design of the Two-level architecture takes advantage of some characteristics of the intermediary distributed architectures analyzed in Section 2.2 by exploiting:

– cooperation among the nodes of the architecture;
– a distributed *discovery* task.

Other features related to the node organization and content partition are original.

3.1 Organization of the nodes

The Two-level architecture organizes the nodes in two sets that provide different tasks: thin edge nodes and fat interior clusters. Figure 2 shows how the tasks of *adaptation*, *edge*, *discovery*, *cache* and *fetch* are mapped on the nodes of the architecture.

The edge nodes are located on the Internet borders. They are thin nodes executing *edge* and *discovery* tasks that are not intensive from a computational and storage point of view. Since an edge node is basically a gateway that does not require complex software and powerful hardware, it can be easily implemented by one server machine, at most two for fault tolerance reasons. Edge nodes may be easily replicated and spread around the Autonomous Systems of the most popular Internet Service Providers. In the version proposed in this paper, we assume that the edge nodes do not

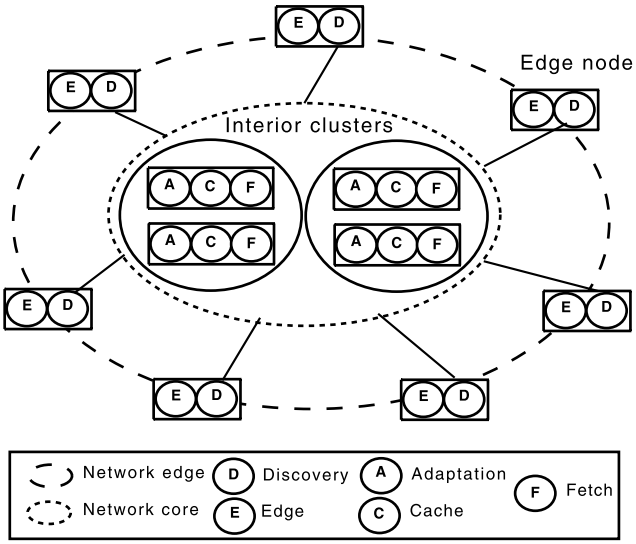


Fig. 2 Task organization in the Two-level architecture

provide any resource caching. This choice is motivated by a twofold goal. First, we aim to reduce storage costs; second, we avoid the issues of preserving cache content consistency, that would require a non negligible effort, especially in architectures with hundreds or thousands of edge nodes that may cache any resource.

The second level consists of geographically distributed clusters executing the tasks of *adaptation* and *caching* that are expensive from a computation and storage point of view, respectively. Furthermore, the clusters host also the *fetching* task that is strictly connected to the caching one. Each interior cluster consists of multiple physical servers with adequate power and storage capacities.

Clustering techniques guarantee several advantages in terms of system management. Any modification of the number of servers in an interior cluster is transparent to the clients, to the other nodes and, most importantly, to the discovery scheme and resource allocation. We can add, remove, replace some servers in the interior clusters without the need of reconfiguring system and data as long as the number of the interior clusters does not change. The number of interior clusters is one or two orders of magnitude lower than that of edge nodes and tends to be constant (in the order of some units, maximum 10). The limited number of interior clusters makes it feasible to place them in the best connected Autonomous Systems, that have the largest number of BGP peers with other Autonomous Systems. The strategic position of the interior clusters in well connected Internet locations limits the user response time variability, which is a potential problem of geographically distributed architectures [14, 38].

3.2 Content partitioning

The Two-level architecture exploits a content partitioning scheme of the global resource space so that each interior cluster manages a different partition of the original and adapted resources. We adopt a hash-based partitioning scheme on the URL of the original resource so that the original resource and all its adapted versions are kept in the same interior cluster. A resource x is associated to an interior cluster through a hash function $H(x)$. This function takes as input a resource ID (e.g., the URL) and returns a value k, where $k \in [1, \ldots, n]$, and n is the number of interior clusters. An example of $H(x)$ uses the MD5 algorithm to distribute the resources uniformly across n nodes:

$$H(\text{resourceID}) = (\text{MD5}(\text{resourceID}) \bmod n) + 1. \quad (1)$$

The deployment of a content partitioning technique on a Two-level architecture is a clear step ahead with respect to other architectural solutions in terms of response time variability and infrastructure management.

Applying hashing on a flat architecture would be unfeasible. For example, the operation of adding a new edge node to increase computational power would require a modification of the hash function (because the term n in Equation 1 changes) and would result in a re-configuration and redistribution of resources among the nodes. Although some solutions [38] can mitigate the most negative impacts, the management overhead is not acceptable when the architecture is large and one server may often join or leave the system also because of temporary system/network failures. A further problem preventing the use of hashing in a flat architecture and overlay networks, such as DHTs, is related to network issues [38, 42] because a request may be forwarded to a close or very far edge node, resulting in high variability of the user-perceived response time [14, 38].

On the other hand, the use of clusters in the hash-based partitioning schemes allows the management of the infrastructure by reducing the need of complex system reconfigurations. Adding a server within a cluster in the Two-level architecture is transparent with respect to the service of client requests and does not require the re-allocation of the resources. System-wide reconfiguration, with migration of resources on a geographic scale, is required only when a new interior cluster is added to the architecture. However, such modification occurs only in case of a significant drift of the client request patterns from the expected behavior obtained through capacity planning analyses, that is unlikely to occur more than once a year. Furthermore, the use of a limited number of large interior clusters, that is one-two orders of magnitude lower with respect to edge nodes, has also a benefit with respect to network-related issues. The Two-level architecture adopts hash partitioning only for few logical interior clusters that are located in the best connected

Autonomous Systems, so that the number of hops from any edge node to any interior cluster is similar and reduces the variability of user-perceived response time.

We can conclude that the Two-level architecture based on interior clusters allows to exploit all the advantages of hashing with no penalty for two reasons.

- It avoids replication of the same resources on different nodes, so that the system storage capabilities are utilized at their best. This is especially important in a scenario where the working set can easily become one order of magnitude larger than the original set because of adapted resources.
- It preserves in a simple way the access locality of the requests, because if multiple versions of the same resource exist, they are stored on the same interior cluster. The benefit on cache hit rate helps addressing the computational issues of content adaptation.

3.3 Request management

The client requests are managed by the Two-level architecture in a plain way, especially if compared with the request management carried out by some flat cooperative architectures. Figure 3 outlines the entire scheme. When an edge node receives a client request (step 1), it extracts the resource ID. Then, the edge node applies the hash function to the resource ID to identify the interior cluster that may hold a valid version of the requested resource, and it forwards the client request to that cluster (step 2). Hence, the *discovery* task of the Two-level architecture is a simple application of a hash function. If the selected interior cluster does not hold the requested resource, no other interior cluster does, hence there is no need for cooperative lookup.

Due to the presence of multiple versions of a resource, the caching semantics is more complex than that of traditional caching systems. We observe that any request to an interior cluster has three possible results.

Exact hit: if the cache contains the exact version of the requested resource, the interior cluster immediately sends it to the edge node (step 5).

Useful hit: if the cache contains a more detailed and adaptable version of the requested resource, it can be transformed to satisfy the client request (step (4)—the numbers between brackets in Fig. 3 specify steps that are not always executed). For example, a video clip with a low frame rate can be obtained through a content adaptation process from any version with a higher frame rate. After that the interior cluster sends the adapted resource to the edge node (step 5).

Miss: if the cache of the interior cluster does not contain any exact or adaptable version of the requested resource, the interior cluster must fetch the original version from the origin server (step (3)); if necessary, the interior cluster performs the required adaptation (step (4)), and sends the adapted resource back to the edge node (step 5).

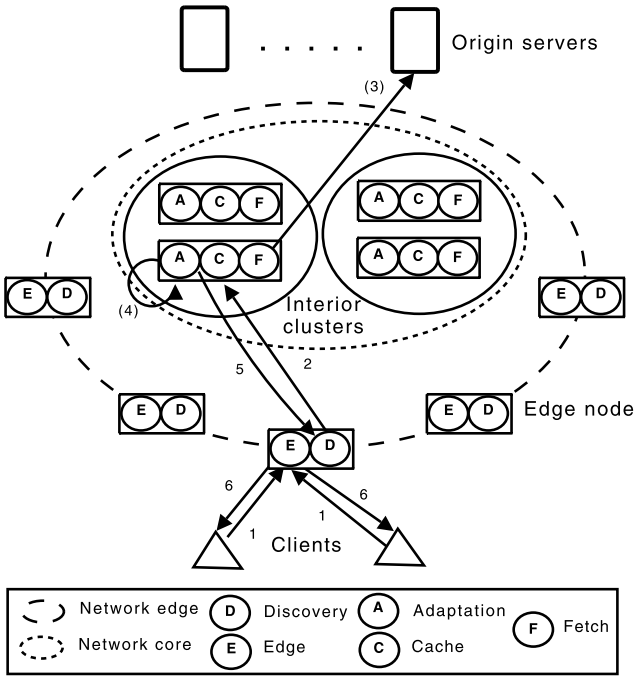


Fig. 3 Two-level architecture for content adaptation and delivery services

The final delivery of the object to the client (step 6) is always carried out by the edge node.

4 Flat cooperative architectures

For comparison purposes, in this section we consider one of the most popular *flat* cooperative architectures that implements content adaptation and delivery services. The flat architecture consists of peer nodes that are placed on the network edge. Each node provides all the necessary tasks for content adaptation and delivery: *edge*, *cache*, *discovery*, *fetch*, and *adaptation*, as shown in Fig. 4.

When the edge node receives the client request resulting in a miss on the local cache, that is when the edge node does not own an exact or adaptable version of the required resource, it can take advantage of cache contents of other edge nodes through a *discovery* task that may result in a *remote* useful or exact hit.

Several discovery mechanisms have been proposed in the context of cooperative Web caching [16, 35, 45]. In a previous study [5], the authors have adapted two popular discovery mechanisms, commonly known as *summary-based* and *query-based*, to handle the discovery of multi-version resources in a geographical distributed architecture.

In summary-based schemes, such as Cache Digests [16], each edge node obtains information about the cache contents

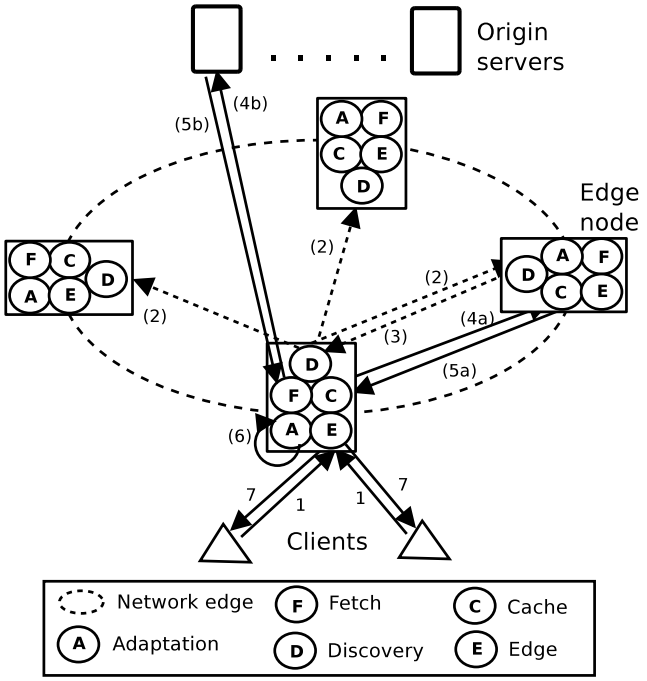


Fig. 4 Flat query-based architecture for content adaptation and delivery services

of the other peers through periodical exchanges of asynchronous messages and stores this information locally. In query-based schemes, that are typically based on the ICP protocol [45], a message exchange mechanism must be activated at runtime among the peers to identify the node that owns the required resource. The pros and cons of the two schemes are intuitive. Due to the synchronous nature of query-based interactions, this scheme provides always up-to-date information resulting in a higher cache hit rate than a summary-based scheme. The summary-based cooperation can be affected by obsolete information that hinders the effectiveness of resource discovery. On the other hand, the discovery operations of the query-based scheme are inherently slower and more expensive because they require a query/reply message exchange at discovery time. Moreover, while a remote hit is detected as soon as the local node receives an exact hit reply message, a miss may be expensive. Nevertheless, in [5] the authors found that the query-based scheme represents the best performing scheme for flat cooperative architectures based on content replication. The query-based mechanism relies on a modified version of the ICP protocol [45], where the resource discovery among the nodes is carried out through a query-response exchange activated by the node that has received the client request.

Figure 4 describes the operations carried out in the Flat query-based architecture when an edge node receives a client request from possibly heterogeneous devices (step 1). The first operation is a local cache lookup that can result in local exact hit, useful hit or miss. Exact and useful hits are handled locally by delivering exact hits immediately (step 7) and useful hits after content adaptation (step (6)). In the case of local miss, the *discovery* task starts a remote discovery issuing query messages for the requested resource to the other nodes (step (2)—the dashed lines represent query messages). Each remote node may reply with an exact hit, a useful hit, or a miss message (step (3)). Some nodes might also not reply at all; in this case, the local node assumes a miss after a certain timeout. The remote discovery may have multiple consequences: in the case of remote hit, the resource is fetched from the cache of the peer node (steps (4a) and (5a)); in the case of useful hit, the resource is also adapted to obtain the requested version (step (6)). Then the resource is delivered to the client (step 7). When no adaptable resource is found in any peer of the architecture (global miss), the node contacted by the client must get the original resource from the origin server (steps (4b) and (5b)). If necessary, the local node adapts the retrieved resource (step (6)) before sending it to the client (step 7).

5 Experimental setup

5.1 Metrics of interest

We compare the performance of the *Two-level* cooperative architecture against that of a *Flat* query-based architecture characterized by query-based cooperation. To provide a more complete comparison we also consider a non cooperative architecture (*No_Coop*) consisting of geographically distributed nodes that do not collaborate for adaptation, discovery, and delivery purposes. We evaluate the architectures on the basis of three main requirements that any architecture for efficient content adaptation and delivery services should address.

High performance. The architecture should guarantee the end-user with an adequate level of performance. We evaluate the architecture performance by considering the user response time, which is the main interest for the user.

Robust performance. The architecture should provide not only high peak performance, but also ensure good performance under not controllable external conditions, such as network traffic and resource popularity changes.

Scalability. The architecture should be scalable with respect to larger numbers of system nodes and increasing adaptation costs of the workload model, because it has to face the current evolution towards large distributed systems and multimedia-oriented workloads [13, 46].

The user response time is defined as the time between the client request and the arrival of the complete response measured on the client. Due to the heavy tail distributions of the response times in the considered systems [12], mean

values are not representative and we prefer to refer to the 90-percentile and cumulative distributions. We consider also the cache hit rate, that is a metric of interest for the system administrators. To guarantee performance robustness, we carry out a sensitivity analysis of the response time with respect to different network scenarios, workload characteristics and infrastructure size.

5.2 Workload and service models

To evaluate the performance of the different architectures, we build a prototype for content adaptation and delivery services that tailor Web and multimedia resources to the client requirements. To exercise the prototype, we define a workload WL1 that aims to represent a realistic Web scenario [46]. Table 1 shows the composition of the workload WL1: HTML pages, images, video and audio resources. HTML pages and images represent together the 90% of the resources, but they account for just the 42% of the bytes of the working set. Multimedia (audio and video files) represent the 10% of the resources, but they account for almost the 58% in terms of bytes. Images are in GIF and JPEG formats [6], while audio and video resources are MP3 and MPEG 4 files, respectively [21]. In this paper we consider static pre-stored multimedia resources, because in the Web almost the totality of audio and video clips are delivered through HTTP streaming [21], that follows a play-while-downloading approach without RTP-based streaming. We have introduced a popularity resource distribution by defining a set of hot resources (corresponding to 1% of the working set): 10% of the total number of the client requests refers to this set.

The prototype adapts images, video and audio resources on the basis of the client requirements. We have identified four classes of client devices that differ for their capabilities [7].

Table 1 Composition of workload WL1

Resource type	Resources (%)	Bytes (%)
HTML	22.2%	9.3%
Images	67.9%	32.8%
Video	8.0%	49.4%
Audio	1.9%	8.5%

PC: a full featured PC or a high-end laptop with a LAN or WiFi connection; this device can consume every resource in the original form.

Web Appliance: a set-top box that turns a TV into a Web browser and may be connected through a DSL modem; the main limits are related to the displaying capabilities of the TV screen.

Handheld PC: a device that is typically equipped with a UMTS connection; the main limits are represented by the small display size and the limited computational power.

Smartphone: a modern cellular phone; besides the limited display size and computational power, a significant limitation is also represented by the mobile connection, typically GPRS.

Table 2 summarizes the transformations that are carried out on images, video and audio resources to match the requirements of the considered device classes. It is important to note that we focus just on a subset of the possible transformations that can be performed to tailor resources to connection and device capabilities, however the main conclusions of this paper can be applied to any class of applications that require computationally intensive adaptation services.

For scaling and managing color depth of images we rely on the Imagemagick program. For video resources we carry out multiple transformations through the Transcode program. In particular, for Web Appliances and Handheld PC we change the frame rate and we compress the video data through the MPEG 4 compression, provided through the DivX codecs. For smartphones, we convert the video stream into a sequence of JPEG frames, at the rate of 1 frame per second, according to the Motion JPEG standard. Finally, audio clips are re-sampled at different frequencies through the Lame software.

Client requests are issued to the system through a client emulator according to synthetic traces with a request rate of 50 requests per second, for a total experiment duration of 2 hours. The requests are evenly distributed among the nodes providing the *edge* task. The probability that a request belongs to a given client device type is defined through a *device vector* D. In this paper, we consider $D_i = \{25\%, 25\%, 25\%, 25\%\}$ where $i \in \{\text{PC, Web appliance, Handheld PC, Smartphone}\}$. Although it is difficult to predict the trend of diffusion of future Web connected devices, in other not reported experiments we found that different mixes of client

Table 2 Resource transformations

Device	Images		Video		Audio
	Resolution	Color depth	Format	Frame rate	bit rate
PC	original	original	original	original	original
Web Appliance	640×480	16-bit color	MPEG 4	20 fps	56 kbps
Handheld PC	480×320	16-bit color	MPEG 4	10 fps	32 kbps
Smartphone	160×120	8-bit	Image set	–	8 kbps

device types within 10% of those considered here do not affect the main conclusions of this paper.

5.3 Experimental testbed

For a fair comparison, all the architectures are provided with the same amount of storage and computing elements. In our experiments all the architectures use 16 nodes, that are organized as follows: the Flat query-based cooperative architecture and the No_Coop architecture exploit 16 edge nodes, while in the Two-level architecture there are 2 interior clusters, each of which consists of 8 servers. Additional edge nodes in the Two-level architecture are simple gateways that do not contribute to storage and computing. The global storage size (that is, the sum of the sizes of all the storage elements) of each architecture covers 80% of the original working set. The servers are equipped with our prototype software and are connected through a fast Ethernet network.

For the performance evaluation and the sensitivity analyses we consider a controlled network environment that introduces network delays through the WAN emulator provided through special packet schedulers that are part of the 2.6 Linux kernel. We emulate packet loss and delay through the *netem* packet scheduler, while bandwidth limitation is obtained through the *token bucket filter* traffic shaper. Round trip delays, loss rates and bandwidth considered in the network scenarios are consistent with the datasets from real-world measurements in [48].

The WAN emulation scenario aims to represent a real geographic network, where the origin servers are placed in a remote location, connected through a geographic link with 14 hops in between, a mean round-trip time of 60 ms, and a maximum bandwidth of 16 Mb/s, while the intermediary nodes are on the same LAN and no WAN effects are introduced on these links.

6 Performance evaluation

The main performance comparison of the architectures is based on the user response time, that is the total elapsed time between the request and the arrival at the client of the last byte of the requested resource. Figure 5 shows the cumulative distribution of the response time for the Two-level, Flat query-based and No_Coop architectures in the case of workload WL1.

The performance of the Two-level architecture is clearly superior: about 80% of the requests are served in less than 1.2 s by the Two-level architecture, while for the Flat query-based and No_Coop architectures this percentage is below 60% and 50%, respectively. This result has two main motivations. The Two-level architecture has the most efficient discovery mechanism and a high cache hit rate that seem to

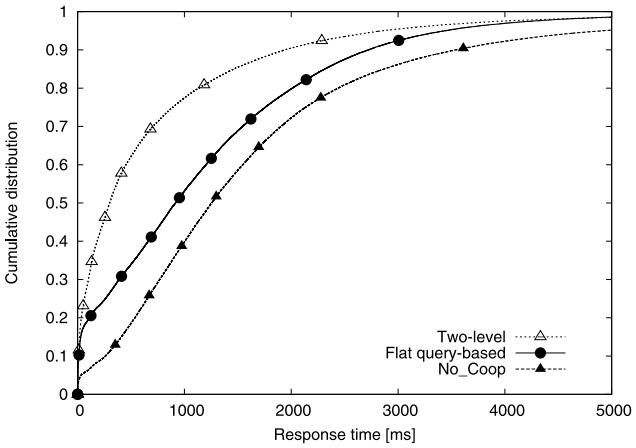


Fig. 5 Cumulative distribution of response times

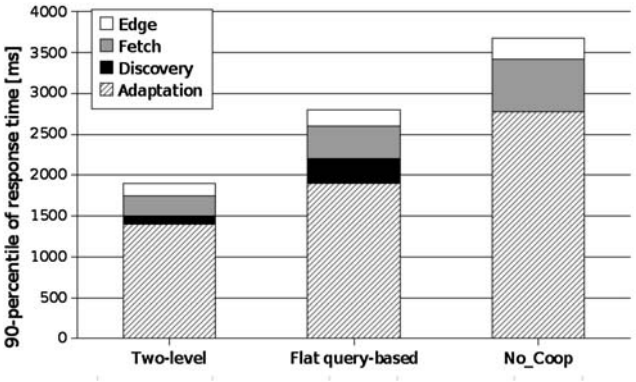


Fig. 6 Breakdown of the response times

counterbalance the need of contacting two servers for each request. To fully understand the architecture performance, let us analyze a breakdown of the response time.

Figure 6 shows the contributions to the response time related to the various tasks of a distributed system for the support of content adaptation and delivery services: *edge*, *fetch*, *discovery* and *adaptation*. The results are clearly in favor of the Two-level architecture: all the contributions of the Two-level architecture achieves better performance than those of the Flat query-based and No_Coop alternatives.

The lower *fetch* and *adaptation* times of the Two-level architecture are due to its higher cache hit rate. Indeed, a cache hit avoids to fetch the resource from the origin server; an exact hit avoids any adaptation; a useful hit requires a partial and less expensive adaptation.

The result for the *discovery* task is less intuitive. We recall that the No_Coop architecture does not exploit the *discovery* task. On the other hand, the Two-level architecture requires a double step lookup for every request, while the cooperative lookup of the Flat query-based architecture is activated only in the case of local miss.

Table 3 Cache hit rates

Architecture	Local		Remote		Global
	Exact	Useful	Exact	Useful	
No_Coop	8.3%	6.4%	n/a	n/a	**14.7%**
Flat query-based	8.0%	7.0%	25.1%	26.8%	**66.9%**
Two-level	n/a	n/a	60.2%	21.0%	**81.2%**

The lower discovery time of the Two-level architecture can be explained by considering the communication model of the two discovery processes. The Two-level architecture requires a one-to-one communication (from one edge node to one interior cluster), while the Flat query-based alternative adopts a one-to-many communication model that may cause high variability. Its discovery process can be very fast in the case of local hits, but it can be very time consuming in the case of misses because a miss is detected only after receiving the response from the slowest peer or after the expiration of a timeout. This has a major impact on the 90-percentile of the *discovery* time and explains the results of Fig. 6.

We now evaluate the cache hit rates, that have an important impact on the performance of the considered architectures. Table 3 reports the cache hit rates, that are divided in local, remote, exact, and useful for the workload WL1. The last column of this table shows the global cache hit rate, which is the sum of the previous hit rates. We do not report the local hits of the Two-level architecture, because the edge nodes act only as gateways and do not cache any resource. For the No_Coop architecture there are not remote hits because there is no lookup among the peers.

From the last column of Table 3 we observe that the global hit rates differ significantly depending on the architecture. This table indicates that cooperation can increase the cache hit rate: the Flat query-based architecture takes advantage of remote hits, thus increasing its global cache hit rate with respect to the No_Coop architecture. As expected, the Two-level architecture achieves a global hit rate considerably higher than that of the Flat query-based and No_Coop architectures. The best cache hit rate is due to the hash function that maximizes the access locality of the requests and avoids the presence of duplicated resources in the nodes. Avoiding cache content duplication allows the distributed caching system to contain a larger fraction of the working set and reduces the amount of cache content replacement operations.

A more detailed analysis of the cache hit rate components shows an important difference between the Flat query-based and Two-level architectures. The percentages of exact and useful hits for the Flat query-based architecture are similar, while the exact hits of the Two-level architecture are much higher than the useful hits. This result confirms that the choice of partitioning the resource space among the interior clusters and keeping all the versions of a resource on the same interior cluster allows the Two-level architecture to maximize locality and achieve the highest rates for the most precious (exact) cache hits.

7 Sensitivity analysis

In this section we evaluate how the considered architectures can satisfy the requirements of performance robustness and scalability. To this aim we evaluate how the architectures can cope with changes in the resource popularity and multiple network scenarios. Furthermore, we study the architecture scalability with respect to the infrastructure size and to highly multimedia-oriented workloads.

7.1 Experimental setup

For the evaluation of the performance robustness and of the scalability of the architectures we carry out experiments for different workloads, network scenarios and infrastructure sizes.

To evaluate the impact of resource popularity, we use the workload composition in WL1 and we create workloads with different Zipf popularity request distributions by varying the Zipf α parameter from 0 to 1. Higher values of the Zipf α parameter correspond to more skewed workloads, starting from a uniform workload for $\alpha = 0$. The popularity of each resource is independent of its URL (used for the computation of the hash function in the Two-level architecture), hence popular resources are spread across the nodes of the infrastructure randomly for both Flat query-based and Two-level architectures.

The sensitivity to network characteristics is evaluated considering two scenarios, namely *server scenario* and *intermediary scenario*. For each scenario we take into account two sets of links: the links between the intermediary architecture and the origin servers (the E-O and I-O links in Fig. 7) and the links among the nodes of the intermediary architecture (the E-E and the E-I links in Fig. 7). Since our experiments compare the intermediary architecture performance, the last mile problem is out the scope of this paper. Hence, we do not consider network effects among the clients and the nodes providing the edge task (that is, C-E links in Fig. 7), because they have the same impact on any architecture.

In the *server scenario* we consider different delay and bandwidth for the links between the intermediary architecture and the origin servers, while the characteristics of the links among the nodes of the intermediary architecture remains unchanged. In the *intermediary scenario* we change

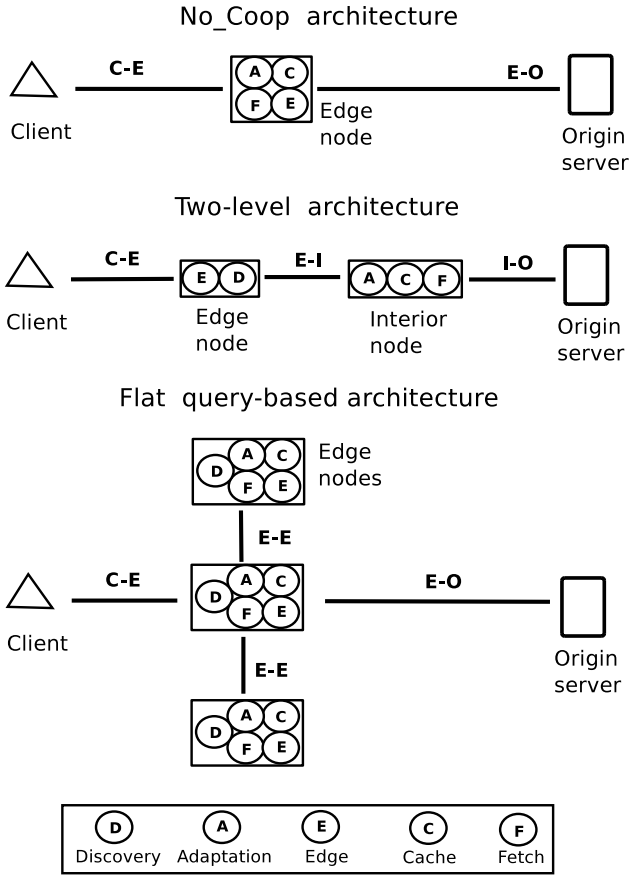


Fig. 7 Network scenario

Table 4 Network parameters

Link	Server scenario		Intermediary scenario	
	Bandwidth	Delay	Bandwidth	Delay
C-E	n/a	n/a	n/a	n/a
E-E, E-I	32 Mb/s	10 ms	8–32 Mb/s	0–20 ms
E-O, I-O	8–32 Mb/s	20–100 ms	16 Mb/s	20 ms

the characteristics of the links among the nodes of the intermediary architecture and we do not modify delay and bandwidth for the links between the intermediary architecture and the origin servers. Table 4 shows the network parameters for each considered scenario.

We are interested in evaluating whether and how the workload characteristics may affect the architecture performance. To this purpose, we consider the robustness of the performance with respect to the popularity distribution of the resources, and we evaluate how the architecture may cope with future workloads with different resource composition.

To evaluate the scalability of the architectures with respect to a growing demand for multimedia resources, we consider a workload, namely *WL2*, where the amount of multimedia resources is greatly increased with respect to the workload WL1 described in Section 5. WL2 represents a likely future scenario where universal access to audio and video clips is provided through the Web and is based on the traffic seen by popular resource sharing sites (e.g., YouTube [47]) and sites for multimedia downloading (e.g., MagnaTune [29]), where multimedia content accounts for the 30% of the resources and almost the 85% of the bytes composing the working set. HTML pages and images represent the 14% and 56% of the resources, respectively.

Finally, to study the scalability of the cooperative architectures with respect to their sizes, we evaluate the performance for three scenarios, namely S1, S2 and S3, corresponding to different system sizes. For the Flat query-based and No_Coop architectures we consider 8, 16, 32 edge nodes for the scenarios S1, S2 and S3, respectively. For the Two-level architecture, S1, S2 and S3 are characterized by 1, 2, 4 interior clusters, where each interior cluster consists of 8 servers. Since our focus is on scalability, for this analysis we increase the load offered to the system proportionally to the number of nodes so that the average number of requests per node remains constant.

7.2 Sensitivity to resource popularity

We now analyze the sensitivity of the Two-level architecture to the resource popularity skewness. Literature on distributed systems clearly shows that hashing can be sensitive to the so called *hot-spot* problem [38]. This may represent a drawback of the Two-level architecture, because the skewness of the resource popularity could trigger an overload condition on some interior cluster of the architecture. On the other hand, we should consider that a highly skewed workload may improve the cache hit rate with a consequent positive effect on the architecture performance. Hence, it is interesting to evaluate the trade-off due to the impact of the resource popularity.

To measure the degree of load balancing across the interior clusters of the Two-level architecture we consider the *load balance metric* (LBM) [3]. Let us define the load of an interior cluster i (of n clusters) at the jth observation (of m observation periods) as $load_{i,j}$ and $peak\_load_j$ as the highest load on any cluster at the jth observation. Since each interior cluster of the Two-level architecture consists of multiple servers, we consider the load of an interior cluster as the number of active connections at the Web switch level. The LBM is defined as follows:

$$LBM = \frac{\sum_{1 \le j \le m} peak\_load_j}{(\sum_{1 \le j \le m} \sum_{1 \le i \le n} load_{i,j})},$$

where the value of the LBM can range from $1/n$ to 1. Smaller values of the LBM indicate a better load balance than larger values.

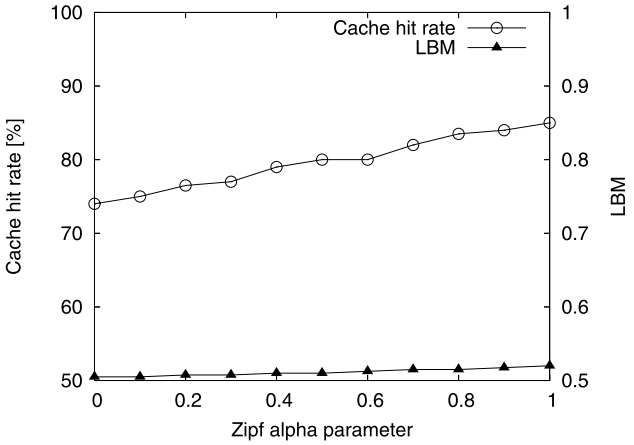


Fig. 8 Sensitivity to resource popularity skewness

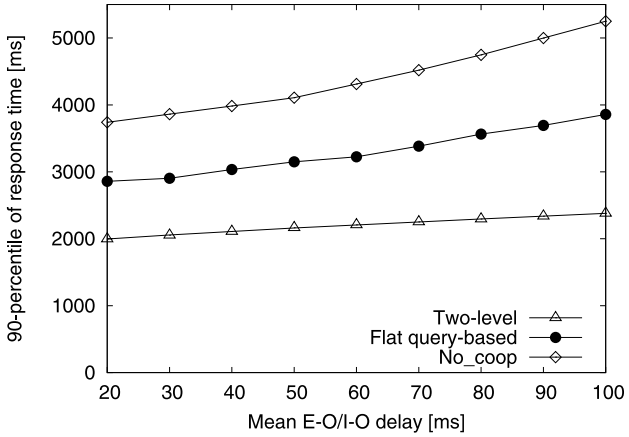


Fig. 9 Sensitivity to delay in link to origin servers (server scenario)

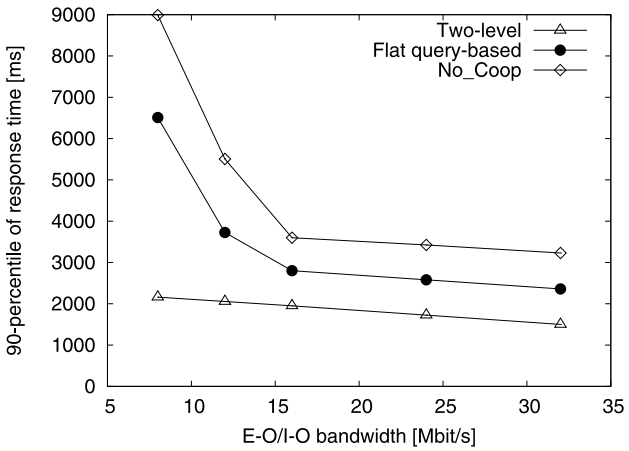


Fig. 10 Sensitivity to bandwidth in link to origin servers (server scenario)

Figure 8 shows the cache hit rate and the LBM values for the Two-level architecture as a function of the resource popularity skewness. It is worth to note that an LBM value equal to 0.5 indicates a perfect load balance among the interior clusters of the considered architecture ($n = 2$ in our experiments). We observe that the curve referring to the LBM metric remains very close to the value 0.5 for every resource popularity skewness, thus showing that the Two-level architecture achieves a good load balance and avoids hot-spots even in the case of highly skewed workloads. This result may be motivated by considering that the resource popularity skewness has a so positive impact on caching to eliminate the risk of unbalance. Indeed, the increment of the cache hit rate in Fig. 8 allows the Two-level architecture to avoid many content adaptation and fetches from the origin server, thus preventing overload conditions on the interior clusters responsible for the most popular resources. We can conclude that the caching effectiveness achieved through the hash-based content partitioning allows us to take full advantage from increased workload skewness and to obtain a balanced load among the interior clusters of the Two-level architecture.

7.3 Sensitivity to network parameters

We consider also important to evaluate the sensitivity to network parameters for the considered architectures. This analysis extends the study of the previous section to evaluate whether the Two-level architecture provides always better performance than that of the Flat query-based and No_Coop alternatives or if there are conditions where the Two-level architecture does not represent the preferable choice.

We first evaluate the impact of the network conditions on the links between the intermediary system and the origin servers (E-O/I-O links depending on the architecture). For this study we will refer to the *server scenario*. We can anticipate that the Flat query-based and No_Coop architectures should be more sensitive to the status of the links connecting the nodes to the origin servers because their lower cache hit rate tends to place a higher load on these network connections with respect to the Two-level architecture.

Figure 9 shows the sensitivity of the 90-percentile of the response time to the delay in network links connecting the nodes of the distributed architecture with the origin servers. We can see that the performance difference between the Two-level architecture and the other two alternatives is more evident as the delay grows: for a delay of 20 ms, the speedup of the Two-level architecture is 1.43 over the Flat query-based architecture and 1.87 over the No_Coop alternative, while for a delay of 100 ms the speedup grows to 1.60 and 2.20, respectively. This is a first confirmation that the Two-level architecture is less sensitive than the other distributed architectures to network delays on the connections to the origin servers. This conclusion is even more evident if we look at the bandwidth. In this case the most significant result from Fig. 10 is the poor performance of the Flat

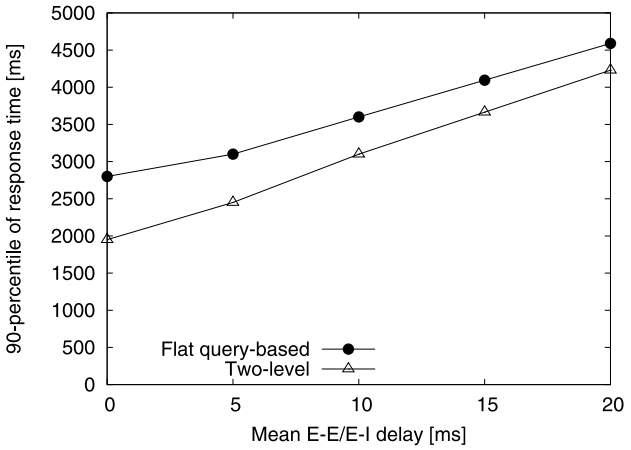


Fig. 11 Sensitivity to delay in link among intermediary nodes (intermediary scenario)

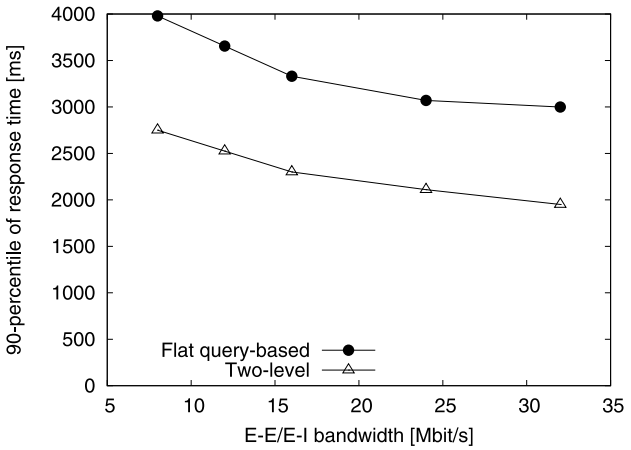


Fig. 12 Sensitivity to bandwidth in link among intermediary nodes (intermediary scenario)

query-based and No_Coop architectures when the available bandwidth is small. The lower cache hit rate achieved by these architectures causes a higher number of requests to the origin servers with respect to the case of the Two-level architecture. The high usage of the links to the origin servers significantly increases the response times for the Flat query-based and No_Coop architectures when the bandwidth is reduced to 8 Mbit/s because of network congestion on the E-O links.

The second experiment considers the *intermediary* scenario that focuses on the links among the nodes of the architectures (E-E and E-I). These links represent a possible bottlenecks of the Two-level architecture, because each client request has to pass through an edge node and an interior cluster. We want to evaluate whether the Two-level architecture can guarantee adequate performance even in the case of poor network connections among the nodes of the distributed architecture.

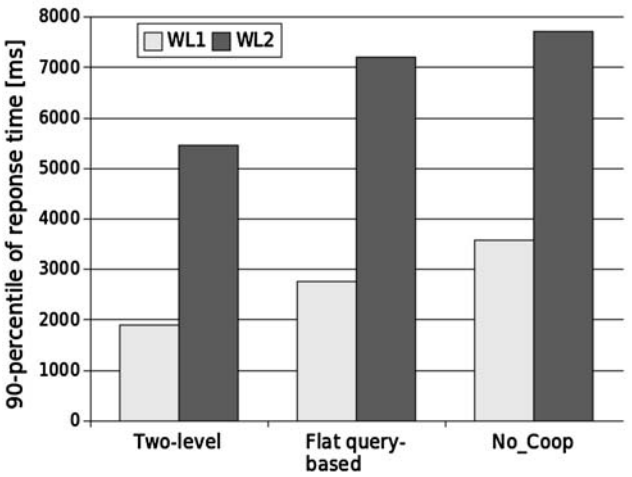


Fig. 13 Architecture performance for different workloads

Figures 11 and 12 confirm the not negligible impact on the response time of the network effects in the links among the nodes of the considered architectures. From Fig. 11, we see that the response times of the Two-level architecture tend to converge towards those of the Flat query-based architecture as the network delay grows. This confirms that the Two-level architecture is really sensitive to E-I delays. On the other hand, Fig. 12 reports the response times as a function of the network bandwidth. This parameter influences the performance of both architectures, but it does not penalize one with respect to the other.

7.4 Scalability to future workloads

In this analysis we evaluate the scalability of the architectures with respect to a growing demand for multimedia contents. To this purpose, we consider also the WL2 workload, that mainly consists of multimedia resources.

We now compare the performance of the architectures under the workloads WL1 and WL2. Figure 13 shows the 90-percentile of the response time for the considered architectures for the two workloads. We observe that the performance of the Two-level architecture is clearly superior to that of the Flat query-based and No_Coop architecture for every considered workload. However, to fully understand the causes of the different performance of the architectures, it is useful to analyze a breakdown of the response time according to the multiple tasks necessary to support content adaptation and delivery services. This study also allows us to evaluate how the content adaptation time changes as a function of the workload characteristics.

Figure 14 shows the contributions of the various tasks to the response time of the three architectures, from which we can observe that for every workload the main contribution to the response time is due to the *adaptation* task. It is the

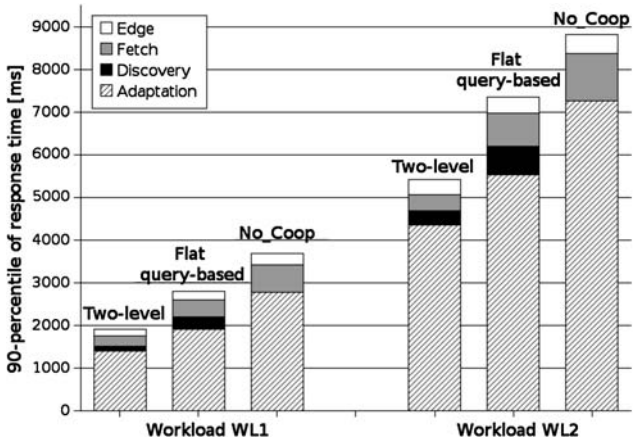


Fig. 14 Breakdown of the response time

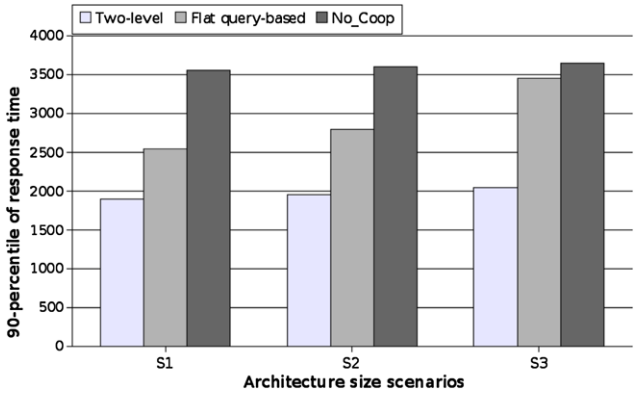


Fig. 15 System scalability to the architecture size

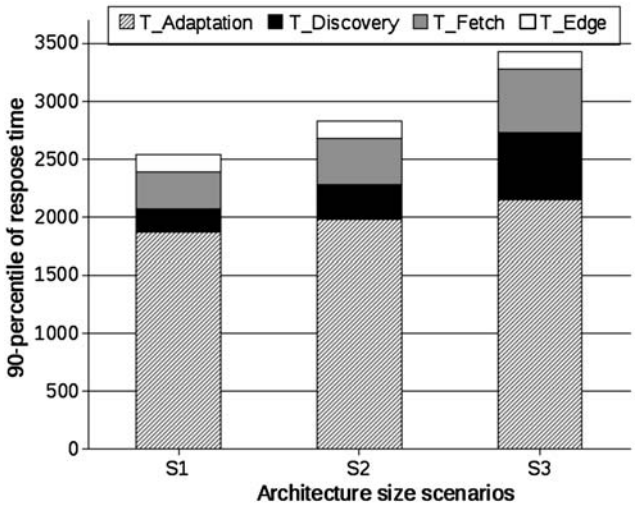


Fig. 16 Breakdown of response time for the Flat query-based architecture

growth of the adaptation time that causes a significant increase of the response time as we pass from the workload WL1 to the workload WL2. When the computational cost of content adaptation grows, the impact of the other network-related costs (discovery and fetch times) is reduced, even if the global response times augments because of the delivery of larger resources. We can thus infer that the network status will reduce its impact on performance in a near future for a twofold reason. First, the cost of content adaptation is likely to increase due to the growing amount of rich multimedia resources. Second, the network technology is evolving towards high bandwidth and low latency links, thus allowing to reduce the contribution of network-related delays on the response time. All these reasons indicate that the Two-level architecture is a preferable solution to support content adaptation and delivery services for present and future workloads.

7.5 Scalability to architecture size

To analyze whether the Two-level architecture can face the challenge of building a large scale system for supporting content adaptation and delivery services, we evaluate the performance of the distributed architectures under the workload WL1 for different system sizes. To this aim, we refer to the scenarios S1, S2 and S3 introduced in Section 7.1.

Figure 15 shows the 90-percentile of the response time as a function of the number of nodes. This graph shows a clear dependency of the performance of the Flat query-based architecture on the number of nodes, while the response times for the No_Coop and Two-level architectures do not significantly vary as the number of nodes increases. The motivation for this performance robustness is quite different. Every node of the No_Coop architecture operates as a stand alone node, hence the performance is poor independently from the architecture size. On the other hand, the Two-level architecture achieves the best performance and an almost perfect scalability even if the number of nodes quadruples thanks to the hash mechanism.

The poor scalability results of Flat query-based architecture deserve a further study. In Fig. 16 we show a breakdown of the response time for an increasing number of nodes from which we can conclude that the main motivation is the increment of the discovery and fetch times. A high number of nodes augments the loss or delay probability of a reply message, that in turn increases the number of cache misses. This effect has already been observed for query-based schemes utilized for Web caching [16, 25]. Even the *fetch* and *adaptation* times augment because each cache miss causes a retrieval from the origin server with a consequent possible adaptation of the fetched resource.

8 Related work

Content adaptation has attracted the interest of many researchers. The related efforts may be broadly divided into

two categories: proposals of middleware frameworks which enable innovative adaptation techniques and studies of architectures for supporting content adaptation and delivery services.

Most of the studies on middleware systems for content adaptation and delivery services evidence the flexibility of the proposed frameworks [11, 22, 28]. These studies aim to offer some programmability and configurability by providing a number of reusable and expandable software components and by giving guidelines on how these components can be programmed, configured, deployed, and can exchange information. Most of the previous studies on programmable frameworks pay scarce or no attention on how the software components may be suitably mapped over the nodes of a distributed architecture. Our study provides a detailed analysis of performance and scalability of the deployed services. Novel adaptation techniques to adapt contents to the client capabilities have been proposed in [1, 6, 8, 10, 24, 34], mostly based on image and video size scaling or cropping. In this paper we exploit some of the existing methods for content adaptation.

The focus on performance and system scalability has driven the proposal of many novel architectures and our study clearly fits in this research area. The systems for content adaptation and delivery services can be divided in three categories depending on the system component that performs the adaptation process [4, 23, 27]: *client-based*, *server-based*, and *intermediary-based* adaptation.

Client-based solutions propose the client as a platform for adaptation and delivery services [30, 37]. This scheme suffers from multiple drawbacks. Placing content adaptation on the client device requires the presence of dedicated software and adequate computational power on the client device. Furthermore, client-based solutions do not avoid the download of (potentially) large original resources over the last mile, which is often a narrow-band link. Our Two-level architecture assumes a thin client and focuses on the deployment of a platform that can adapt resources even for very simple devices with limited computational, display and network capabilities.

In the *server-based* approach, the functionalities of the content provider platform are enhanced with content adaptation [9, 23, 31]. Examples that address content adaptation at the server side are Apache Cocoon, that allows the automatic generation of files through the processing of statically or dynamically generated XML files, Oracle Application Server Wireless [32], and IBM WebSphere Transcoding Publisher [44]. When the resources need to be generated and adapted on-the-fly (hundreds of different device profiles already exist [41]), the drawback of the server-based approach is intuitive. Hardware capabilities of a single server platform can be easily exhausted by adaptation of multimedia resources, that have significant computational requirements. Thanks to multiple intermediary cooperative nodes, the proposed Two-level architecture may share the load of computationally expensive services and reduce the response time when requests are served by a node caching the requested version of the resource. Moreover, in the server-based approach the adaptation concerns just the resources of one content provider, while our proposal is oriented to the whole Web.

Intermediary nodes to support content adaptation and delivery services have been proposed in the context of Content Delivery Networks [13, 15, 36]. Projects such as ACDN [36] and Akamai EdgeComputing platform [13] propose to move application servers producing adapted contents close to the network edge to improve the user-perceived performance. In a similar way, the ESI technology [15] allows the architectures to assemble adapted contents on the network edge starting from Web page fragments, that are composed through specific rules that take into account user supplied information. Although all these architectures may support content adaptation and delivery services, these proposals imply a strict relationship between the content provider and the entity operating the CDN infrastructure. For this reason, the adaptation and delivery service concerns only a subset of Web sites, while we focus on a scenario where a third-party entity can provide adaptation and delivery services for the whole Web.

Research efforts on intermediary nodes operated by a third-party to combine content adaptation and caching have been proposed in [7, 39, 40]. All these solutions concern the enhancement of single nodes. On the other hand, we focus on distributed architectures to support content adaptation of multimedia resources because of the limited scalability of single intermediary nodes, that cannot address the high computational and storage requirements of these services.

Several studies have been proposed in literature concerning the use of a cluster of servers to support content adaptation and delivery services [2, 17, 19]. In particular, the study of Fox et al. [17] presents one of the first examples of cluster-based systems for image adaptation, while more recent proposals [2] are oriented to video streaming adaptation. The use of a single cluster of nodes may address computational and storage issues. However, it does not resolve the network constraint: the connection of the cluster to the Internet may become the system bottleneck and represents a single point of failure in the case of network congestion or power outage in the cluster area. The Two-level architecture is a step ahead, because it exploits the benefit of clustering, but it is designed to be distributed over a geographical area, thus preventing network bottlenecks.

In a previous study the authors proposed and compared flat distributed architectures for content adaptation and delivery services oriented to Web workloads consisting of HTML and small images [5]. However, flat architectures

still leave some open issues. These architectures may be affected by unfair load distribution among the nodes and this risk is exacerbated by the high computational costs due to multimedia content adaptations. Moreover, existing flat architectures waste global storage capabilities since they allow multimedia contents to be replicated among multiple nodes. The Two-level architecture represents a significant step further with respect to these proposals. It represents a novel architecture specifically designed to address the issues of adapting multimedia content, that exacerbates the requirements related to the computational cost and working set size. The Two-level architecture improves load balancing through the differentiation of the nodes functionalities and guarantees an optimized usage of the storage space thanks to a resource partitioning technique that avoids content replication.

9 Conclusions

In this paper we propose a novel distributed architecture to support adaptation and delivery services of Web and multimedia contents. The architecture is based on a two-level organization of the nodes: the edge nodes of the first level are simple gateways, while the interior nodes of the second level are based on clusters that execute the most expensive tasks, such as content adaptation, caching, and fetching.

We have evaluated through a prototype the performance of the two-level architecture against that of a flat query-based architecture and that of a non cooperative distributed architecture. The message from all our experiments and sensitivity analysis is clearly in favor of the proposed architecture. The two-level architecture is robust with respect to changes in network features and resource popularity skewness. Moreover, it guarantees high scalability with respect to future scenarios characterized by multi media-oriented workloads and architectures with a large number of nodes.

References

1. Ahmad, I., Wei, X., Sun, Y., Zhang, Y.Q.: Video transcoding: an overview of various techniques and research issues. IEEE Trans. Multimed. **7**(5), 793–804 (2005)
2. Bhuyan, L.N., Guo, J.: Load balancing in a cluster-based web server for multimedia applications. IEEE Trans. Parallel Distrib. Syst. **17**(11), 1321–1334 (2006)
3. Bunt, R.B., Eager, D.L., Oster, G.M., Williamson, C.L.: Achieving load balance and effective caching in clustered Web servers. In: Proceedings of the 4th International Web Caching Workshop, San Diego, CA, 1999
4. Butler, M., Giannetti, F., Gimson, R., Wiley, T.: Device independence and the Web. IEEE Internet Comput. **6**(5), 81–86 (2002)
5. Canali, C., Cardellini, V., Lancellotti, R.: Content adaptation architectures based on squid proxy server. World Wide Web J. **9**(1), 63–92 (2006)
6. Chandra, S.: Content adaptation and transcoding. In: Singh, M.P. (ed.) Practical Handbook of Internet Computing. Chapman Hall & CRC Press, London, Boca Raton (2004)
7. Chang, C.Y., Chen, M.S.: On exploring aggregate effect for efficient cache replacement in transcoding proxies. IEEE Trans. Parallel Distrib. Syst. **14**(6), 611–624 (2003)
8. Chang, S.F., Vetro, A.: Video adaptation: concepts, technologies, and open issues. Proc. IEEE **93**(1), 148–158 (2005)
9. Chen, J., Zhou, B., Shi, J., Zhang, H., Fengwu, Q.: Function-based object model towards Website adaptation. In: Proceeding of the 10th World Wide Web Conference (WWW'01), Hong Kong, 2001
10. Chen, Y., Xie, X., Ma, W.Y., Zhang, H.J.: Adapting Web pages for small-screen devices. IEEE Internet Comput. **9**(1), 50–56 (2005)
11. Chen, Y.F., Huang, H., Jana, R., Jim, T., Hiltunen, M., John, S., Jora, S., Muthumanickam, R., Wei, B.: Imobile ee: an enterprise mobile service platform. ACM J. Wirel. Netw. **9**(4), 283–297 (2003)
12. Crovella, M.: Performance evaluation with heavy tailed distributions. In: Proceedings of the 7th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'01), 2001
13. Davis, A., Parikh, J., Weihl, W.E.: EdgeComputing: extending enterprise applications to the edge of the Internet. In: WWW Alt. '04: Proceedings of the 13th international World Wide Web Conference on Alternate Track Papers & Posters, pp. 180–187, 2004
14. Dykes, S., Robbins, K.: A viability analysis of cooperative proxy caching. In: Proc. of IEEE Infocom 2001, Anchorage, AK, 2001
15. Edge Side Includes. http://www.esi.org (2002)
16. Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary cache: a scalable wide-area Web cache sharing protocol. IEEE/ACM Trans. Netw. **8**(3), 281–293 (2000)
17. Fox, A., Gribble, S.D., Chawathe, Y., Brewer, E.A., Gauthier, P.: Cluster-based scalable network services. In: Proc. of 16th ACM SOSP, pp. 78–91, 1997
18. Gadde, S., Chase, J., Rabinovich, M.: A taste of crispy squid. In: Proc. of Workshop on Internet Server Performance, 1998
19. Grieco, R., Malandrino, D., Scarano, V.: A scalable cluster-based infrastructure for edge-computing services. World Wide Web **9**(3), 317–341 (2006)
20. Gruhl, D., Guha, R., Liben-Nowell, D., Tomkins, A.: Information diffusion through blogspace. In: WWW '04: Proceedings of the 13th International Conference on World Wide Web, 2004
21. Guo, L., Chen, S., Xiao, Z., Zhang, X.: Analysis of multimedia workloads with implications for Internet streaming. In: WWW '05: Proceedings of the 14th International Conference on World Wide Web, 2005
22. He, J., Gao, T., Hao, W., Yen, I.L.: A flexible content adaptation system using a rule-based approach. IEEE Trans. Knowl. Data Eng. **19**(1), 127–140 (2007). Member-Farokh Bastani
23. Hwang, Y., Kim, J., Seo, E.: Structure-aware Web transcoding for mobile devices. IEEE Internet Comput. **7**(5), 14–21 (2003)
24. Ihde, S., Maglio, P.P., Meyer, J., Barrett, R.: Intermediary-based transcoding framework. IBM Syst. J. **40**(1), 179–192 (2001)
25. Lancellotti, R., Mazzoni, F., Colajanni, M.: Hybrid cooperative schemes for scalable and stable performance of Web content delivery. Comput. Netw. J. **49**(4) (2005)
26. Li, M., Claypool, M., Kinicki, R., Nichols, J.: Characteristics of streaming media stored on the Web. ACM Trans. Internet Technol. **5**(4), 601–626 (2005)
27. Lum, W.Y., Lau, F.C.: On balancing between transcoding overhead and spatial consumption in content adaptation. In: MobiCom '02: Proceedings of the 8th annual International Conference on Mobile Computing and Networking, 2002
28. Maglio, P., Barrett, R.: Intermediaries personalize information streams. Commun. ACM **43**(8) (2000)
29. MagnaTune: Magnatune—music downloads and licensing. http://www.magnatune.com/ (2007)

30. Marriott, K., Meyer, B., Tardif, L.: Fast and efficient client-side adaptivity for SVG. In: WWW '02: Proceedings of the 11th International Conference on World Wide Web, pp. 496–507, 2002
31. MediaLab: Web content adaptation. Tech. rep., TeliaSonera (2004)
32. Oracle Application Server Wireless. http://www.oracle.com/technology/tech/wireless/ (2008)
33. Parker, C., Pfeiffer, S.: Video blogging: content to the max. IEEE MultiMed. **12**(2), 4–8 (2005)
34. Pashtan, A., Kollipara, S., Pearce, M.: Adapting content for wireless Web services. IEEE Internet Comput. **7**(5), 79–85 (2003)
35. Rabinovich, M., Spatscheck, O.: Web Caching and Replication. Addison-Wesley, Reading (2002)
36. 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 and Distribution, Hawthorne, NY, 2003
37. Rabinovich, M., Xiao, Z., Douglis, F., Kamanek, C.: Moving edge side includes to the real edge—the clients. In: Proc. of USITS'03, 4th USENIX Symp. on Internet Technology and Systems, Seattle, WA, 2003
38. Ross, K.: Hash-routing for collections of shared Web caches. IEEE Netw. **11**(6), 37–44 (1997)
39. Shen, B., Lee, S.J., Basu, S.: Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks. IEEE Trans. Multimed. **6**(2), 375–386 (2004)
40. Singh, A., Trivedi, A., Ramamritham, K., Shenoy, P.: PTC: Proxies that transcode and cache in heterogeneous Web client environments. World Wide Web **7**(1), 7–28 (2004)
41. Singh, G.: Guest editor's introduction: content repurposing. IEEE Multimed. **11**(1), 20–21 (2004)
42. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. of the 2001 ACM SIGCOMM Conference, 2001
43. Wang, B., Kurose, J., Shenoy, P., Towsley, D.: Multimedia streaming via TCP: An analytic performance study. In: MULTIMEDIA '04: Proceedings of the 12th Annual ACM International Conference on Multimedia, 2004
44. IBM WebSphere Transcoding Publisher. http://www.ibm.com/software/pervasive/transcoding_publisher/ (2008)
45. Wessels, D., Claffy, K.: Internet Cache Protocol (ICP), version 2. RFC 2186 (1997)
46. Williams, A., Arlitt, M., Williamson, C., Barker, K.: Web workload characterization: ten years later. In: Tang, X., Xu, J., Chanson, S.T. (eds.) Web Content Delivery. Springer, Berlin (2005)
47. YouTube: Youtube—broadcast yourself. http://www.youtube.com/ (2007)
48. Zhang, R., Hu, C., Lin, X., Fahmy, S.: A hierarchical approach to Internet distance prediction. In: Proc. of the 26th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS'06), Washington, DC, USA, 2006

Claudia Canali is assistant professor at the University of Modena and Reggio Emilia since 2008. She got the Laurea degree summa cum laude in computer engineering from the same university in 2002, and the Ph.D. degree in computer engineering from the University of Parma in March 2006. During the Ph.D. she spent eight months as visiting researcher at the AT&T Research Labs in Florham Park, New Jersey. Her research interests include distributed architectures for Internet-based services, content delivery networks and mobile systems for mobile Web access. She is member of IEEE Computer Society.

Michele Colajanni is full professor in computer engineering at the University of Modena and Reggio Emilia since 2000. He received the Laurea degree in computer science from the University of Pisa in 1987, and the Ph.D. degree in computer engineering from the University of Roma Tor Vergata in 1991. His research activities on high performance Web systems began in 1996 during his sabbatical year spent at the IBM T.J. Watson Research Center, New York. His research interests include also distributed systems, security, performance and prediction models. He is member of IEEE Computer Society, ACM and Eurosys.

Riccardo Lancellotti is assistant professor at the University of Modena and Reggio Emilia since 2005. He received the Laurea Degree in computer engineering cum laude from the University of Modena and Reggio Emilia in 2001 and the Ph.D. in computer engineering from the University of Roma "Tor Vergata" in 2003. In 2003, he spent eight months at the IBM T.J. Watson Research Center as a visiting researcher. His research interests include privacy and security in distributed systems, scalable architectures for mobile Web access, peer-to-peer systems, performance evaluation and benchmarking. He is a member of the IEEE Computer Society and ACM.