Global Data Management
R. Baldoni, G. Cortese, F. Davide and A. Melpignano.
IOS Press, 2006

From Web Servers to Ubiquitous Content Delivery

Guillaume Pierre [1], Maarten van Steen,
Michał Szymaniak, Swaminathan Sivasubramanian
Vrije Universiteit, Amsterdam, The Netherlands.

Abstract. Hosting a Web site at a single server creates performance and reliability issues when request load increases, availability is at stake, and, in general, when quality-of-service demands rise. A common approach to these problems is making use of a content delivery network (CDN) that supports distribution and replication of (parts of) a Web site. The nodes of such networks are dispersed across the Internet, allowing clients to be redirected to a nearest copy of a requested document, or to balance access loads among several servers. Also, if documents are replicated, availability of a site increases. The design space for constructing a CDN is large and involves decisions concerning replica placement, client redirection policies, but also decentralization. We discuss the principles of various types of distributed Web hosting platforms and show where tradeoffs need to be made when it comes to supporting robustness, flexibility, and performance.

Keywords. Replication, Mirroring, Content delivery networks, Peer-to-peer.

1. Introduction

Thanks to the expansion of the Internet, an ever-growing number of businesses and end users decide to publish information using the World-Wide Web. These documents can range from a set of simple HTML pages to multi-tiered Web-based applications, in which pages are generated dynamically at the time of a request.

The Web was initially designed with the idea that each page would be hosted by one server machine [9]. The resulting system model is extremely simple: to obtain a document, a browser is given a URL which contains the name of the server to contact, and the document path to request. The browser initiates a TCP connection with the requested server and specifies the required document path, after which the server returns the document and closes the connection.

Although this approach is extremely simple, taking a closer look at it also quickly reveals a number of shortcomings. The network connection between the client and the server can have arbitrarily poor performance, potentially leading to long connection setup and transfer delays. The server cannot control the rate of incoming requests, so it always runs the risk of overload when subject to high request rates. This issue can also lead to increased document retrieval delays, or even to servers refusing incoming connections.

[1]Correspondence to: Guillaume Pierre, Department of Computer Science, Vrije Universiteit, de Boelelaan 1081, 1081 HV Amsterdam, The Netherlands; E-mail: gpierre@cs.vu.nl.

Finally, a centralized Web server constitutes a single point of failure: it is sufficient that a server, or the network connection leading to it, fails for the documents it hosts to become unreachable. This situation is clearly unacceptable to most Internet-based businesses or even to demanding end users.

The solution to these issues is to decouple a Web *site* (i.e., a collection of documents related to each other) from the one or more *server machines* used to host it. For example, the Google Web site is believed to be hosted by over dozens of thousands of servers [7]. A large number of systems have been built to this aim, using different techniques ranging from caching or mirroring Web documents, to building high-performance server clusters, and building worldwide replicated hosting platforms known as content delivery networks. More recently, worldwide content delivery has witnessed additional development, with the advent of collaborative and peer-to-peer content delivery networks.

Although distributed Web hosting platforms are very different in their architecture, most of them follow a common goal, namely to share the burden of delivering the contents among multiple machines by way of replication. Replication involves creating copies of a site's Web documents, and placing these copies at well-chosen locations. In addition, various measures are taken to ensure consistency when a replicated document is updated. Finally, effort is put into redirecting a client to a server hosting a document copy such that the client is optimally served. Replication can lead to reduced client latency and network traffic by redirecting client requests to a replica closest to that client. It can also improve the availability of the system, as the failure of one server does not result in entire service outage.

The original nonreplicated system model of the World-Wide Web also imposes a transparency constraint to any Web hosting system. Since no specific support for replication is included in the Web protocols, it is a desired property that replication be transparent to the Web browsers. Otherwise, the complexity of the system must be revealed to the end users by asking them to select a server where requests should be sent to, or by requiring them to use nonstandard replication-aware browsers to access the information.

This chapter will detail the different types of Web hosting architectures, and discuss their relative merits and drawbacks. We will show that, although decentralization is a desirable property, it makes the implementation of advanced features such as latency-driven client redirection and dynamic Web application hosting more complex, or even impossible.

2. Content Delivery Networks

The shortcomings of centralized Web site hosting can be addressed by creating copies of (or *replicating*) a site's Web documents at well-chosen locations and redirecting client requests to a server hosting a document copy such that the client is optimally served. Replication can lead to reduced client latency and network traffic by redirecting client requests to a replica closest to that client. It can also improve the availability of the system, as the failure of one replica does not result in entire service outage.

The simplest form of Web-site replication is mirroring. As discussed in Section 2.1, mirroring requires very little technical sophistication but presents a number of issues due to the almost total lack of centralized control over the system.

A number of infrastructures known as content delivery networks have been developed to address the issues of mirroring by providing worldwide distributed resources

that can be dynamically allocated to Web sites, and by advanced automation of system administration. Section 2.2 presents the general design of content delivery networks.

Many CDNs focus on replicating static Web pages only. However, an increasing fraction of Web contents are generated dynamically at the time of a request by applications that take, for example, individual user profile and request parameters into account when producing the content. Hosting such applications, and their associated databases, requires entirely different techniques than for hosting static contents. We present them in Section 2.3.

Finally, even though CDNs distribute Web contents at a worldwide scale, the management of these resources remains fundamentally centralized. Section 2.4 discusses the advantages and limitations of such an approach.

2.1. Mirroring vs. Content Delivery Networks

The simplest form of Web document replication is mirroring. Mirroring consists of creating copies of (a part of) a Web site at multiple servers. The servers hosting a mirror of a given site are chosen manually, usually by way of server administrators volunteering to mirror a popular Web site. Setting up replication is also done manually by periodically copying the files from the origin server to each of its mirrors. The origin server typically presents a list of mirror servers to its clients, who are expected to select "the best one" manually.

The popularity of mirroring techniques is certainly due to their extreme simplicity. However, this simplicity imposes many limitations in terms of server selection, consistency, availability guarantees, client redirection and administration.

The selection of servers hosting the content is constrained by the willingness of Web server administrators, rather than by the actual needs of the mirrored site. For example, a site may end up having many mirrors in an area where very few clients access the site, and on the other hand, lack mirrors in areas where it is most popular.

The consistency of document replicas is ensured by copying documents periodically, typically once every few hours or days. This limits the effectiveness of mirroring to sites with slow update rates, or which can tolerate the delivery of outdated content to its clients.

A mirrored site does very little to guarantee its availability in the presence of server or network failures. There is no automatic failover mechanism, so when a client notices the unavailability of one mirror, (s)he is expected to select another mirror manually. Furthermore, a failure of the origin server typically means that new clients cannot access the list of mirrors. The whole site then becomes unreachable, even though many mirrors may still work correctly.

Presenting a list of mirrors to a client requires the client to select "the best server" manually. Mirror lists are usually annotated with the geographical location, in the hope that a server geographically close to a client will deliver acceptable performance. However, it is notoriously difficult for a user to predict which server will deliver best performance, as geographical distance is a poor predictor of the performance of an Internet client-to-server network path [37]. In addition, geographical distance says nothing about the current load of the server. In some cases, it may be wise to (temporarily) switch to a farther, but better performing server.

Finally, the decentralization of control of a mirrored site can lead to a number of difficulties. Each mirror server is usually administrated by a different person, who needs to

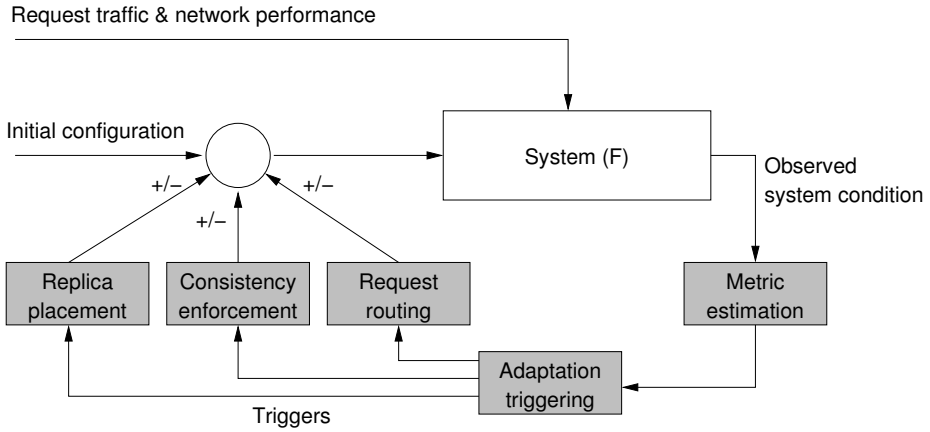


Figure 1. Abstract architecture of a content delivery network.

be contacted and convinced before any administrative task such as a change in configuration can take place. Similarly, each mirror maintains its own access log. Most mirrored sites do not require mirror servers to report their access logs to the origin server, which prevents the site owner from exploiting these logs for understanding the needs of clients or claiming revenue from advertisements embedded in the pages.

Most of the limitations of mirroring can be addressed by centralizing the control of replication, and automating a number of tasks. In particular, a popular Web site may buy servers, install them at well-chosen locations, and manage consistency maintenance itself across these servers. Additionally, a number of solutions can be implemented to automate the redirection of clients to one of the available servers. However, building such a worldwide system is very expensive, so from an economical point of view it often does not make sense to deploy a separate infrastructure for each Web site. This observation leads to the development of Content Delivery Networks (CDNs). A CDN provides many resources that can be dynamically allocated to the sites it is hosting, which allows to share the same infrastructure to host multiple unrelated sites. The next section describes the general design and architecture of most content delivery networks.

2.2. Content Delivery Networks Architecture

The best well-known commercial content delivery network is Akamai [4], but recent years have witnessed the development of many more [17]. Their detailed architecture is often considered a trade secret, but their general principles are known. Likewise, much academic research has been conducted in the domain [32,36].

The detailed architecture of content delivery networks is extremely diverse, but in essence every Web hosting system requires the same types of mechanisms. As illustrated in Figure 1, any content delivery network faces a continuously changing traffic of requests addressed by its clients and must deliver the requested documents via the Internet, whose performance also fluctuates. The goal of a CDN is to continuously adapt its configuration to provide a near-optimal quality of service at the lowest possible cost.

Such adaptation can affect the number and placement of replicas, the mechanisms used to maintain replicas consistent in the presence of updates, and the way client requests are directed to one of the replicas.

To take correct adaptation decisions, the system must monitor its own performance, such as the rate of requests addressed to each document and the location of clients. Although certain performance metrics are trivial to measure, others such as the inter-host network distance require specialized mechanisms [21,24,38].

Another issue is to decide when the system should adapt its configuration to maintain an acceptable level of performance. Adapting the configuration at regular time intervals allows the system to take into account long-term changes in the request traffic [30]. However, this technique does not allow a timely response to sudden changes in the request traffic. Large and abrupt changes in request rates do happen, and are known as flash crowds. To handle them, the system needs to quickly detect certain events which suggest that the situation is changing and that immediate adaptation is needed.

The actual system adaptation can take multiple forms, depending on the nature of the change in the request traffic and the network condition. The first possibility is to change the number or location of replicas. Multiple algorithms have been proposed to select replica placements that minimize the average client-to-replica distance, or to balance the load across replicas.

Another form of adaptation that a system can use is to change the way clients are redirected to replicas. Multiple mechanisms can be used to automatically redirect clients to a given replica, including DNS-based mechanisms [12,18], network-level packet redirection [33], and HTTP redirection. In addition, the system must define a policy to decide where each client should be redirected to. Many such policies have been defined [3,6,40].

Finally, the last type of adaptation is to change the way replicas are kept consistent in the presence of updates. We have shown that, for static Web documents, near-optimal performance can be attained by associating each document with the policy that suits it best [30]. Dynamic Web applications, which execute arbitrary code to generate documents upon each client request, clearly require different techniques, in particular when they access a backend database. Depending on the nature of an application and its client access pattern, different techniques such as fragment caching [13], database query result caching [11] and (partial) database replication [34] may be best employed.

Performance Evaluation Metrics

Evaluating the performance of a content delivery network is no easy task. In particular, there is no one metric that can capture the complexity of such a system. Instead, as shown in Table 1, there exists a wide range of metrics that can reflect the requirements of both the system's clients and the system's operator. For example, metrics related to latency, distance, and consistency can help evaluate the client-perceived performance. Similarly, metrics related to network usage and object hosting cost are required to control the overall system maintenance cost, which should remain within bounds defined by the system's operator.

Different metrics are by nature estimated in different manners. Certain metrics are trivial to measure in the CDN system itself, such as the amount of traffic generated over the network or the amount of storage resources currently in use. However, other relevant metrics such as the client-perceived download latency are much harder to evaluate from the point of view of the CDN and require dedicated measurement infrastructures. For

Table 1. Five different classes of metrics used to evaluate performance in content delivery networks.

Class	Description
Temporal	The metric reflects how long a certain action takes.
Spatial	The metric is expressed in terms of a distance that is related to the topology of the underlying network, or region in which the network lies.
Usage	The metric is expressed in terms of usage of resources of the underlying network, notably consumed bandwidth.
Financial	Financial metrics are expressed in terms of a monetary unit, reflecting the monetary costs of deploying or using services of the replica hosting system.
Consistency	The metrics express to what extent a replica's value may differ from the master copy.

example, metric estimation services are commonly used to measure client latency or network distance. The consistency-related metrics are not measured by a separate metric estimation service, but are usually measured by instrumenting client applications.

Replica Placement

The performance of a CDN depends to a large extent on its ability to identify the location of its clients and to place replicas close to them. In conjunction with an appropriate request redirection mechanism, this allows to optimize the latency and/or bandwidth of the client-to-replica network path. For example, the delivery performance of typical small Web objects is mostly constrained by the network latency [44]. We have shown that a dozen of well-placed replicas can decrease the median client-to-replica latency by a ratio around 3 depending on the site's client population [39], which demonstrates the potential gain of well-placed replicas.

For a content delivery network, replica placement can be divided into two subproblems. The first one is *server placement*, which strives to select a number of locations where servers should be placed. Setting up a new server in a remote location takes time and clearly involves a significant financial cost. As a consequence, server placement typically tries to forecast interesting server locations regardless of the short-term needs of the currently hosted sites. Such algorithms base their decisions on the general topology of the Internet and try to optimize some cost metric without taking the location of actual clients into account.

The second subproblem is *replica placement*, which consists of deciding which of the existing servers should be used to host a particular piece of content [27]. Unlike server placement algorithms, content placement algorithms take as input fine-grain information about the localization of clients who access the particular piece of content to be placed. The cost involved by creating or deleting content replicas in existing servers is relatively low, so content placement algorithms can be executed often to follow the variations of request load as closely as possible.

Request Redirection

Placing replicas carefully can be useful only if clients actually access the replica closest to them. However, manual replica selection falls short of this requirement while it creates an unnecessary burden to the users. Instead, content delivery networks use a variety of mechanisms to automatically redirect clients to one of the replicas.

Request redirection first requires a redirection mechanism to instruct Web browsers of the server where they should issue their requests. Two mechanisms are commonly

used. First, with HTTP redirection a redirector can decide on a per-page basis which replica server is to handle the request. To this end, the redirector returns the URL of the replicated page to the client. The drawback of HTTP redirection is the loss of transparency and control: as the client is effectively returned a modified URL, it can decide to cache that URL for future reference. As a consequence, removing or replacing a replica may render various cached URLs invalid. Alternatively, a second mechanism is DNS redirection. In this case, redirection is based entirely on a site's host name and the client's location. When the client resolves the site's host name, the redirector returns the IP address of the replica server closest to the client. In this case, redirection is done on a per-site basis as the DNS redirector has no means to differentiate between individual pages. On the other hand, DNS redirection is mostly transparent to the client, allowing for better control of replica placement. Most CDNs employ DNS redirection.

In addition to a redirection mechanism, one needs to define a redirection policy which decides where each client should be redirected to. Policies usually try to redirect clients to a replica server close to them. However, other criteria such as the respective load of replica servers may also be taken into account when taking this decision [19,26].

Consistency Maintenance

Creating copies of Web documents creates a new problem: when a document is updated, old copies need to be refreshed or destroyed so that no outdated information is delivered to the clients. A wealth of techniques have been developed to achieve this [42], but they can essentially be classified along three main dimensions.

The first dimension is the level of consistency that a specific consistency policy provides. Ideally, one would like no outdated document to be ever delivered to a client. Techniques to achieve this are however quite expensive in terms of necessary network traffic. To address this issue, many policies relax the consistency requirement by allowing some bounded level of inconsistency. Such inconsistency bounds are often expressed as a maximum time during which an outdated version is allowed to remain in the system, but they can also be expressed in terms of the number of outstanding updates or semantic distance between versions [43].

The second dimension is the nature of the update messages that are exchanged in the system. When a document is updated, the simplest form of update, called *state shipping*, consists of transferring the whole content of the new version. However, if only a few changes were applied to a long document, it might be more efficient to propagate those differences only, leading to *delta shipping*. Finally, *function shipping* carries the identity and parameters of an operation that must be applied to the outdated version to bring it up-to-date. Note that the latter two forms require that each replica server has a copy of the previous version available.

The third dimension is the direction in which updates are propagated. In some systems, the origin server *pushes* updates to its replicas. Other systems prefer replica servers to *pull* updates from their origin. Hybrid schemes also exist, and combine both approaches depending on the characteristics of the document [10].

An interesting form of document consistency was used in the Akamai CDN [28]. In this scheme, a hash of the content of a document is embedded in its URL. When a document is updated, the new version has a different hash value, so it is stored independently from the old version at a URL containing the new hash value. All other documents referring to it merely need to update their references to replace the old URL with the new

one. The old document version can coexist with the new one, as these are in fact implemented as two different documents. The old one, which ceases to be requested, is rapidly evicted from the system. Although this mechanism is very elegant, it has one drawback: a document's URL changes at each update, which makes this technique applicable only to the replication of embedded contents (images, videos, etc.). HTML documents cannot be easily replicated using this technique, which perhaps explains why Akamai apparently does not use this consistency policy any more.

Adaptation Triggering

As mentioned previously, the goal of a CDN is to continuously monitor its own performance so that it can adapt its configuration to changes in the traffic of requests it handles, and maintain near-optimal performance over time. This raises the question: when should a CDN update its configuration? Adaptations usually involve a cost (e.g., in terms of performance during the transition or cost of increased network traffic) so an adaptation should take place only if its anticipated benefits exceed the involved costs.

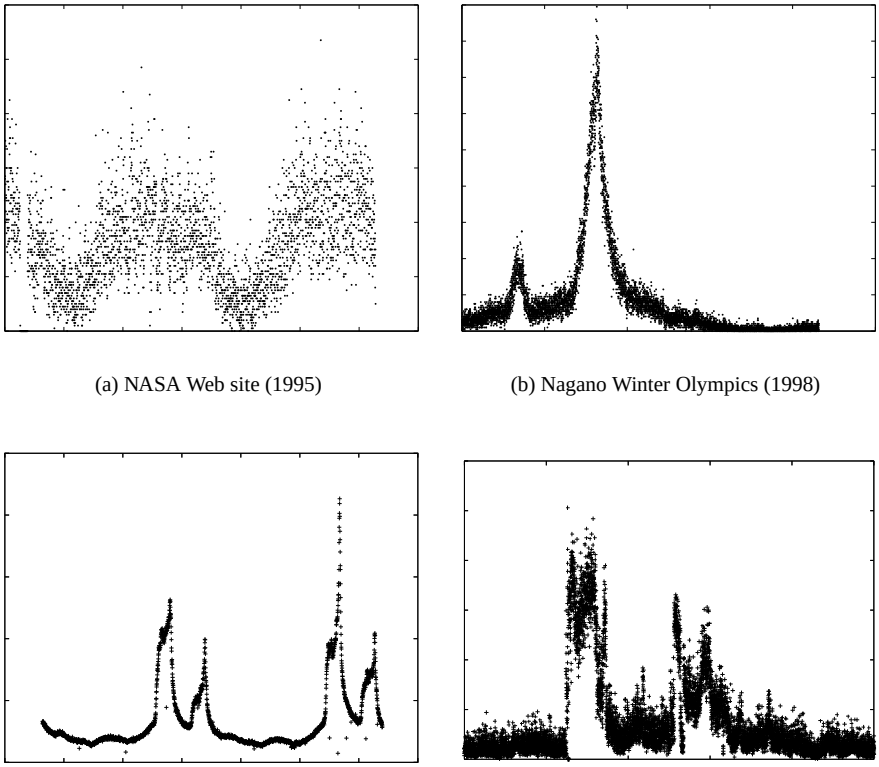
The simplest adaptation triggering scheme consists of adapting the system on a periodic basis. We have shown that, provided that the access patterns do not change too quickly, periodically re-evaluating the configuration of a CDN allows one to maintain near-optimal performance over time [30]. In such a scheme, the system periodically collects information about its own recent behavior, and evaluates whether a different configuration would have offered a better performance. If so, it then updates its own configuration accordingly.

A major drawback of such an approach is that it relies on the assumption that recent past access patterns allow one to predict the near future with a reasonable accuracy. Should access patterns change dramatically between two periodic adaptations, such as upon the occurrence of a flash crowd, the system would be unable to react in a timely manner. Due to a variety of reasons, a server's request load can increase by several orders of magnitude within minutes, and decrease back to normal only after several hours [2].

Figure 2 shows the variation of load of four different Web sites. Figure 2(a) shows normal variations of load according to a day-and-night pattern. Even though the load does vary to a large extent, its variations are predictable enough to be efficiently handled by periodic adaptation. On the other hand, Figures 2(b), 2(c) and 2(d) show abnormal behavior with harder-to-predict huge load peaks. In such situations, a system cannot rely on periodic adaptation any more. What is needed is to detect the flash crowd at its earliest stage, *predict* its near-future characteristics, and proactively adapt the system accordingly [8]. Adaptation often consists of increasing the number of replicas of the concerned documents. However, adapting the consistency and redirection policies may also help to a certain extent, for example to switch from a proximity-based redirection policy to a load-balancing-based one.

2.3. Dynamic Document Hosting

With the development of Web forums, e-commerce sites, blogs and many others, an increasing fraction of Web content is not delivered from a static file but generated dynamically each time a request is received. Dynamically generating Web contents allows servers to deliver personalized contents to each user, and to take action when specific requests are issued, such as ordering an item from an e-commerce site.



(a) NASA Web site (1995)

(b) Nagano Winter Olympics (1998)

(c) Soccer World Cup (1998)

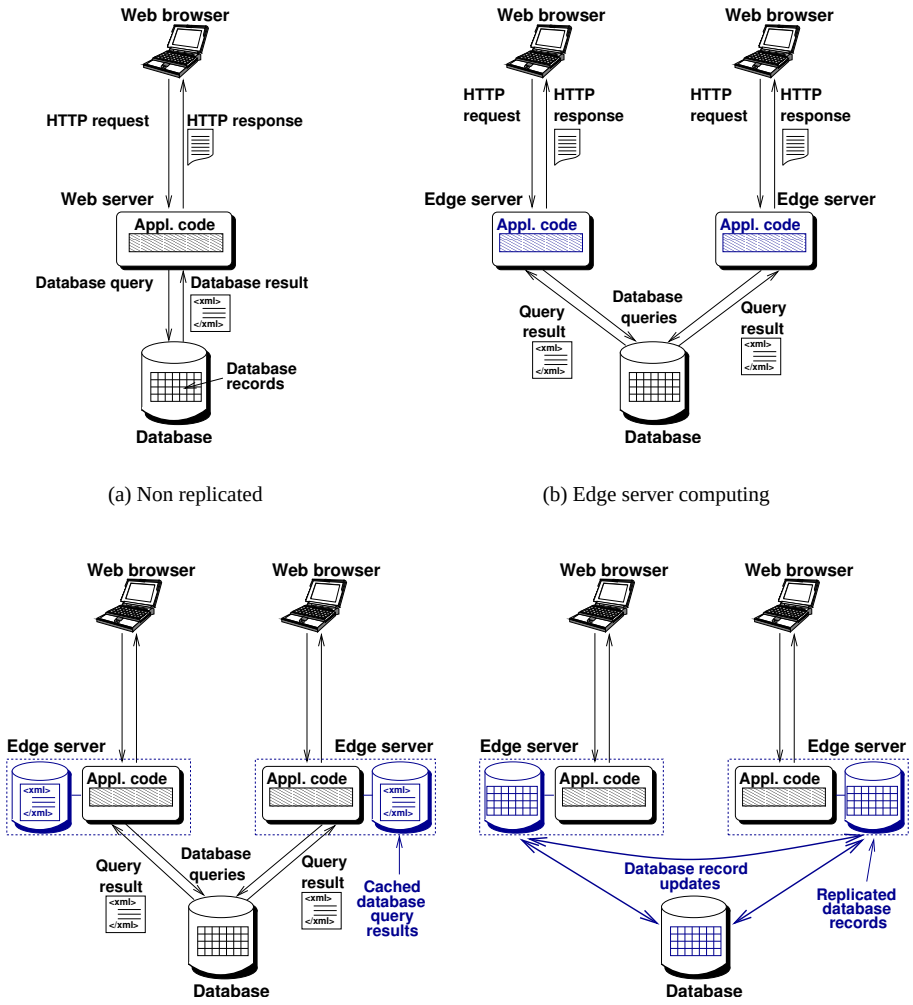
(d) Slashdotted site (2004)

Figure 2. One normal server load, and three different flash crowds (adapted from [8]).

Dynamic Web applications are often organized along a three-tiered architecture, as depicted in Figure 3(a). When a request is issued, the Web server invokes application-specific code, which generates the content to be delivered to the client. This application code, in turn, issues queries to a database where the application state is preserved.

From the point of view of a content delivery network, it can be tempting to host such Web applications using similar techniques as for static content. One can indeed ignore the fact that documents are dynamically generated, and cache the content as it is generated by the application. However, this technique, called fragment caching, offers poor performance as it is often unlikely that the exact same request will be issued again at the same server. Moreover, maintaining the consistency of dynamic document copies is hard because any update in the underlying database can potentially invalidate a copy.

An improved solution consists of duplicating the application code at all replica servers while the database remains centralized. This allows each server to execute the application in reaction to client requests (see Figure 3(b)). Edge server computing, as it is called, allows servers to generate contents tailored to the specificities of each client request while distributing the computational load [16]. On the other hand, the central-



(a) Non replicated

(b) Edge server computing

(c) Database query caching

(d) Database record replication

Figure 3. Various Web application hosting techniques (adapted from [29]).

ized database often constitutes a performance bottleneck, which limits the scale that such systems can reach.

To overcome these limitations, it is necessary to move the data to the edge servers, thereby reducing the load of the database. To this end, two types of systems can be distinguished. First, it is possible to cache the results of database queries at the edge servers (Figure 3(c)). Content-aware caching requires each edge server to run its own database server which contains a partial view of the centralized database [5,11]. Each query is subject to a so-called 'query containment check' to determine if it can be answered from the locally-available data. When this is not the case, the query is issued to the central

database. Results are subsequently inserted in the local database, before being returned to the application.

A second, simpler alternative to content-aware caching is content-blind caching, where the edge servers do not need to run a database server nor be aware of the database structure [35]. Instead, it stores query result structures independently from each other. This results in storing potentially redundant information at the edge servers. On the other hand, storing precomputed query results eliminates the database overhead of content-aware caching.

Finally, database caching techniques work well only for applications which repeatedly issue the same queries to their database. For applications which do not exhibit this behavior, it can be more efficient to replicate the whole database at the edge servers (Figure 3(d)). This guarantees that edge servers can always query their local database copy. On the other hand, database replication involves a lot of communication when the database is updated. One way to deal with this problem is to use partial database replication [34].

2.4. Discussion

As can be seen, content delivery networks are significantly more sophisticated than mirrored environments in terms of automation and control. Many issues that are typically handled manually in mirrored systems can in fact be best realized in a more automatic fashion, such as continuously evaluating the system's performance, selecting appropriate numbers and locations of replicas, redirecting requests and maintaining consistency. Also, only automatic systems are likely to deal with a flash crowd in a timely manner.

A different set of techniques must be used by CDNs to host dynamic Web applications. In comparison, mirroring techniques can only host very specific types of applications (such as an application containing only code, but no database).

CDNs also provide improved control over the system. While the administration of a mirrored system mostly relies on the good will of a multitude of administrators, CDNs offer more centralized control based on systematic performance evaluations and well-defined adaptation strategies. Such centralized control, however, is made possible mostly by the fact that a single entity (the CDN operator) owns and controls the whole server infrastructure, which may be dispersed across the Internet. Such an architecture restricts the operation of a CDN to large companies capable of investing the necessary funds, and which expect return on investments. In practice, this means that CDNs build commercial offers for the use of their infrastructure, which in turn limits the use of these technologies to a restricted class of Web site owners.

3. Collaborative Content Delivery Networks

Deploying CDN technology for only a single Web site is difficult as it requires that the owner has access to a large collection of machines placed at strategic locations in the Internet. Moreover, it is highly inefficient not to share this infrastructure as it effectively amounts to a gross overprovisioning of resources for just a single site.

On the other hand, it is not obvious why Web hosting for increased quality of service should be outsourced, as in many cases the necessary resources are already available

elsewhere in the system. Certain users or organizations, such as a supermarket chain, may have multiple computers online, so the extra resources needed are generally already available in-house. Alternatively, several users can decide to build their own CDN and join their respective resources in order to average everyone's resource needs over all the provided servers. Consider, for example, a retail chain. Assuming that each shop will always have at least one computer online (most of the time), the extra resources needed are probably already available in-house. As another example, worldwide nongovernmental organizations may be able to connect the computers of their local branches and team up to jointly host a single fully distributed Web site, making effective use of their own resources. A similar argument holds for other groups of which the members may want to jointly host a Web site, such as many virtual online communities. We thus come up with a collaborative model where independent organizations team up their resources for each other's benefit.

Collaborative content distribution networks are similar in architecture to noncooperative CDNs as described in the previous section: they need to evaluate performance, handle replica placement, do request redirection, maintain consistency, and trigger adaptations in order to keep the system's performance as close as possible to the optimum. We shall not detail these issues here, as they are mostly identical to those discussed in Section 2. However, the fact that CCDNs are operated by a group of organizations rather than a single entity creates a number of issues. In a system such as Globule [29] and DotSlash [45], each Web site ends up being replicated at a collection of servers which belong to different organizations and may not have the same goals and policies regarding the system. This raises a number of new issues regarding system management and security.

3.1. Availability

In a CCDN, resources are typically contributed by many organizations independent from each other, with very few guarantees regarding their availability. Servers may become unreachable due to voluntary disconnection from their owner, or because of a hardware, software or network failure. For these reasons, CCDNs should expect any server to be unreachable a significant fraction of the time. Moreover, when the number of servers taking part in hosting a given site increases, the probability that at least one server is unreachable grows quickly. A CCDN therefore needs to make sure that a site will remain functional even when a fraction of its hosting servers fail.

The first problem to address is the availability of the redirector subsystem at the time of a client request. When using DNS redirection, this issue is easily solved. A DNS redirector is simply a DNS server responsible for the site's host name, and which returns responses customized to each client. The DNS protocol allows multiple redundant servers to be registered for the same name; if one server fails, then the other ones are automatically queried instead.

The second issue is to make sure that at least one server is available at any time, and has a copy of the contents to be delivered. This advocates some form of full replication, where a site's contents is fully replicated at a number of servers. The Web site as a whole cannot experience a failure as long as one of these servers remains available. Note that this does not rule out all forms of caching or partial replication. Globule, for example, supports two forms of replication simultaneously: full replication across a few 'backup

servers' guarantees the site's availability, while partial replication across a potentially large number of 'replica servers' is in charge of optimizing the site's content delivery performance.

Finally, it is necessary that the redirector subsystem monitors the availability of the servers participating in hosting a given site. When one server fails, the redirector should redirect requests to a 'second best' server so that the failure is not perceived by the clients.

3.2. Brokerage

An important goal of a CCDN is to offer Web site owners suitable servers where to host their contents. However, in a CCDN servers may join or leave the system at any time. In such conditions, finding good servers where to host a site's content may prove difficult.

First, the definition of a 'good' server is more complex in a CCDN than in a commercial CDN. Clearly, criteria such as a server's location and network capacity are crucial. But other criteria such as the availability of specific dynamic document generation software, the identity of a server's owner and server-specific access right policies may also influence the choice. In particular, it is very important for server administrators to keep control over which site is hosted at their server.

One solution is to make administrators negotiate access rights manually, as done for example in DotSlash [45]. Such a choice is suitable for DotSlash, as this system is mostly concerned with handling flash crowds. In this system, peer servers are involved in delivering another site's content only upon a flash crowd. When this happens, the number and capacity of servers that take place in the 'flash-crowd rescue' are more important than their location.

However, when content replication is to be realized on a permanent basis and servers are expected to join and leave the system at will, such manual hosting negotiation is not practical any more. To address this issue, Globule proposes servers to register to a central repository so that queries can be issued to find suitable servers. Administrators can also specify policies to define who is authorized to host content at their server. Finally, they are proposed a number of servers with compatible access right policies to host replicas of their content.

3.3. Security

In a CCDN, a server will often host content that does not belong to its own administrator. In such situations, most administrators would demand guarantees that potentially malicious content cannot damage the server, by means of excessive resource usage, access to confidential information, or any other type of misbehavior. This threat is particularly present when hosting dynamic content, where arbitrary code can be executed to generate documents. This is a well-known problem, however, which is usually addressed by means of sandboxing techniques [1].

Another more difficult security issue is that a content owner expects guarantees that replica servers will actually perform their assigned task faithfully. A malicious replica server could, for example, reject incoming connections (creating a denial-of-service attack) or deliver modified versions of the original content. It is impossible for an origin server to check directly the content delivered by a replica server to a client without negating the benefits of replication. Instead, it is necessary to involve (some of) the clients

in checking the contents and reporting misbehaviors to the origin server. When the site is made of static content, the origin server can sign documents and expect clients or client-side proxies to check signatures. When dynamically-generated content is involved, however, more sophisticated techniques become necessary [31].

3.4. Discussion

Collaborative CDNs allow individually contributed servers to team up resources in the form of storage capacity, bandwidth and processing power. In return, their Web content is transparently and automatically replicated according to quality-of-service demands regarding performance, availability, and reliability. In this way, a collaborative CDN offers the same service as a commercial CDN, but at virtually no extra costs beyond what a member is already paying for Internet connectivity.

Most implementation techniques used in CDNs can be used in CCDNs as well. However, CCDNs experience new issues due to distributed management and security concerns that do not appear in CDNs. Current solutions to these issues introduce some additional burden to the administrators, who are not necessarily professionals in the domain. These constraints drive the need for simpler content distribution technologies, even at the cost of very restricted functionality.

4. Peer-to-Peer Content Delivery Networks

A completely different approach to distributing Web content in a decentralized fashion is the use of peer-to-peer technologies. Unlike traditional CDNs, peer-to-peer systems spread the request load across all their members hosts, which makes them extremely resilient to node failures and load surges.

Although very similar in features to a content delivery network, traditional peer-to-peer systems such as Gnutella [23] and BitTorrent [14] have not been specifically designed to host Web content. They are only focused at large-scale delivery of large, static and immutable files such as music or video files. Web content, on the other hand, is much harder to deliver using this type of overlays because it is made of many small documents which are potentially updated frequently or even generated dynamically. Moreover, these systems are designed to be access using specific client applications rather than standard Web browsers, which breaks the transparency requirement discussed in the introduction. Finally, the way they route requests through the overlay is often not designed to optimize document access latency but to maximize the throughput and the scalability of the system.

A number of peer-to-peer-based systems have been built specifically to host Web content. Systems such as Coral [22] and CoDeeN [41] are in fact made of a (potentially large) number of Web caches that cooperate with each other by way of peer-to-peer technologies. This architecture allows to handle large amounts of traffic with significantly better performance than noncooperative caches. It also enables regular Web browsers to access them using the standard HTTP protocol.

It must be noted that Web-oriented peer-to-peer CDNs do not involve the browsing users into the content delivery itself. Instead, both systems mentioned above are actually operated over a relatively limited number of servers, all of which remain under the con-

trol of their respective programmers [15]. Although their architecture does not clearly impose such centralized control, peer-to-peer CDNs will need to solve the same issues as collaborative CDNs before they can really be deployed in a fully decentralized fashion.

The architectures of Coral and CoDeeN as Web caches have another important consequence: the origin server does not actively participate in these systems. Building a CDN independent from the origin servers allows the replication of any Web site with no intervention of the site owner. However, it also prevents the CDN from hosting dynamically generated content as all techniques described in Section 2.3 except fragment caching require specific support at the origin server. Dynamically-generated content is typically considered not cacheable, which is the reason why current peer-to-peer CDNs are effective at hosting static content only.

It must be noted that, although existing peer-to-peer CDNs cannot host dynamically generated content efficiently, there is no fundamental reason why this would be impossible. However, one would need to host both application code and data in a peer-to-peer network, and provide a rich interface to access and modify the data. A number of research efforts is being conducted in this direction [20,25] which might in the future allow peer-to-peer CDNs to efficiently host dynamic Web applications.

5. Conclusion

Replicating a Web site over a collection of servers can improve the site's access performance and availability. The simplest form of Web replication is mirroring in which all replication-related issues are handled manually. However, mirroring falls short in terms of systematic performance improvement, availability guarantees and ease of administration.

Content delivery networks provide Web sites with advanced replication techniques. Most aspects of replication are handled automatically, such as replica placement, consistency maintenance, request redirection, etc. CDNs also continuously evaluate their own performance to automatically adapt their configuration upon changes in the request traffic.

Collaborative content distribution networks allow independent people or organizations to cooperate in order to build their own content delivery network. If done right, such a system can provide the same features as a commercial CDN, but at virtually no extra cost beyond what each member is already paying for Internet connectivity. CCDNs use similar techniques to CDNs, but they also face additional specific issues regarding management and trust, which make them harder to operate.

Finally, a next step toward decentralization is represented by peer-to-peer content delivery systems. These systems have the advantage of spreading the request load across all their members hosts, which makes them extremely resilient to node failures and load surges. On the other hand, their architecture currently limits them to hosting static Web content. More complex data types such as dynamic Web applications are currently beyond the reach of these systems.

Decentralized Web site replication is very appealing for reasons of cost, robustness and reactiveness to flash crowds. However, the more decentralized the hosting platform, the more difficult it is to provide rich features such as dynamic document replication. Very active research is being conducted in this area, so we can expect future Web hosting systems to approach the goal of a true ubiquitous content delivery infrastructure.

References

[1] M. Achour, F. Betz, A. Dovgal, N. Lopes, P. Olson, G. Richter, D. Seguy, J. Vrana, and several others. *PHP Manual*, chapter 42: Safe Mode. PHP Documentation Group, 2005. http://www.php.net/features.safe-mode.

[2] S. Adler. The Slashdot effect: An analysis of three Internet publications. http://ssadler.phy.bnl.gov/adler/SDE/SlashDotEffect.html.

[3] A. Aggarwal and M. Rabinovich. Performance of replication schemes for an Internet hosting service. Technical Report HA6177000-981030-01-TM, AT&T Research Labs, Florham Park, NJ, October 1998.

[4] Akamai. http://www.akamai.com/.

[5] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan. DBProxy: A dynamic data cache for Web applications. In *Proc. 19th Intl. Conf. on Data Engineering (ICDE)*, pages 821–831, Bangalore, India, March 2003.

[6] M. Andrews, B. Shepherd, A. Srinivasan, P. Winkler, and F. Zane. Clustering and server selection using passive monitoring. In *Proc. 21st INFOCOM Conference*, pages 1717–1725, New York, USA, June 2002.

[7] L.A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The Google cluster architecture. *IEEE Micro*, 23(2):22–28, March-April 2003.

[8] Y. Baryshnikov, E.G. Coffman, G. Pierre, D. Rubenstein, M. Squillante, and T. Yimwadsana. Predictability of web-server traffic congestion. In *Proc. 10th Intl. Workshop on Web Content Caching and Distribution*, pages 97–103, Sophia Antipolis, France, September 2005.

[9] T. Berners-Lee, R. Cailliau, A. Luotonen, H.F. Nielsen, and A. Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, August 1994.

[10] M. Bhide, P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive push-pull: Disseminating dynamic Web data. *IEEE Transactions on Computers*, 51(6):652–668, June 2002.

[11] C. Bornhövd, M. Altinel, C. Mohan, H. Pirahesh, and B. Reinwald. Adaptive database caching with DBCache. *Data Engineering*, 27(2):11–18, June 2004.

[12] V. Cardellini, M. Colajanni, and P.S. Yu. Request redirection algorithms for distributed web systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):355–368, April 2003.

[13] J. Challenger, P. Dantzig, A. Iyengar, and K. Witting. A fragment-based approach for efficiently creating dynamic Web content. *ACM Transactions on Internet Technologies*, 5(2):359–389, May 2005.

[14] B. Cohen. Incentives build robustness in BitTorrent. In *Proc. Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003.

[15] Coral frequently asked questions. Can I run a CoralCDN node? http://wiki.coralcdn.org/wiki.php/Main/FAQ#runnode.

[16] A. Davis, J. Parikh, and W.E. Weihl. EdgeComputing: Extending enterprise applications to the edge of the Internet. In *Proc. Intl. World Wide Web Conference*, pages 180–187, New York, USA, May 2004.

[17] B.D. Davison. Web caching and content delivery resources. http://www.web-caching.com/.

[18] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. *IEEE Internet Computing*, 6(5):50–58, September 2002.

[19] C. Ferdean and M. Makpangou. A response time-driven replica server selection substrate for application replica hosting systems. In *Proc. Intl. Symposium on Applications and the Internet*, Phoenix, Arizona, USA, January 2006.

[20] W. Fontijn and P. A. Boncz. AmbientDB: P2P data management middleware for ambient intelligence. In *Proc. Workshop on Middleware Support for Pervasive Computing*, pages 203–208, Orlando, FL, USA, March 2004.

[21] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A global Internet host distance estimation service. *IEEE/ACM Transaction on Networking*, 9(5):525–

540, October 2001.

[22] M.J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with Coral. In *Proc. 1st Symposium on Networked Systems Design and Implementation*, pages 239–252, San Francisco, CA, March 2004.

[23] Gnutella. http://www.gnutella.com/.

[24] K.P. Gummadi, S. Saroiu, and S.D. Gribble. King: Estimating latency between arbitrary Internet end hosts. In *Proc. 2nd SIGCOMM Internet Measurement Workshop*, pages 5–18, Marseille, France, November 2002.

[25] R. Huebsch, B. Chun, J.M. Hellerstein, B.T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A.R. Yumerefendi. The architecture of PIER: an Internet-scale query processor. In *Proc. Conference on Innovative Data Systems Research*, pages 28–43, Asilomar, CA, USA, January 2005.

[26] K.L. Johnson, J.F. Carr, M.S. Day, and M.F. Kaashoek. The measured performance of content distribution networks. *Computer Communications*, 24(2):202–206, February 2001.

[27] M. Karlsson and C. Karamanolis. Choosing replica placement heuristics for wide-area systems. In *Proc. Intl. Conference on Distributed Computing Systems*, pages 350–359, Tokyo, Japan, March 2004.

[28] F. Thomson Leighton and Daniel M. Lewis. Global hosting system. United States Patent, Number US6108703, August 2000.

[29] G. Pierre and M. van Steen. Globule: a collaborative content delivery network. Submitted for publication, November 2005.

[30] G. Pierre, M. van Steen, and A.S. Tanenbaum. Dynamically selecting optimal distribution strategies for Web documents. *IEEE Transactions on Computers*, 51(6):637–651, June 2002.

[31] B.C. Popescu, J. Sacha, M. van Steen, B. Crispo, A.S. Tanenbaum, and I. Kuz. Securely replicated web documents. In *Proc. 19th Intl. Parallel and Distributed Processing Symposium*, Denver, CO, USA, April 2005.

[32] M. Rabinovich and O. Spatscheck. *Web Caching and Replication*. Addison Wesley, Reading, MA, USA, 2002. ISBN: 0201615703.

[33] P. Rodriguez and S. Sibal. SPREAD: Scalable platform for reliable and efficient automated distribution. *Computer Network*, 33(1–6):33–46, 2000.

[34] S. Sivasubramanian, G. Alonso, G. Pierre, and M. van Steen. GlobeDB: Autonomic data replication for Web applications. In *Proc. 14th Intl. World-Wide Web Conference*, pages 33–42, Chiba, Japan, May 2005.

[35] S. Sivasubramanian, G. Pierre, M. van Steen, and G. Alonso. GlobeCBC: Content-blind result caching for dynamic Web applications. Submitted for publication, October 2005.

[36] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. van Steen. Replication for web hosting systems. *ACM Computing Surveys*, 36(3):291–334, 2004.

[37] L. Subramanian, V.N. Padmanabhan, and R.H. Katz. Geographic properties of Internet routing. In *Proc. Usenix Annual Technical Conference*, pages 243–259, Monterey, CA, USA, June 2002.

[38] M. Szymaniak, G. Pierre, and M. van Steen. Scalable cooperative latency estimation. In *Proc. 10th Intl. Conference on Parallel and Distributed Systems*, pages 367-376, Newport Beach, CA, USA, July 2004.

[39] M. Szymaniak, G. Pierre, and M. van Steen. Latency-driven replica placement. In *Proc. Intl. Symposium on Applications and the Internet*, pages 399-405, Trento, Italy, February 2005.

[40] L. Wang, V. Pai, and L. Peterson. The effectiveness of request redirection on CDN robustness. In *Proc. 5th Symposium on Operating System Design and Implementation*, pages 345–360, Boston, MA, December 2002.

[41] L. Wang, K. Park, R. Pang, V.S. Pai, and L. Peterson. Reliability and security in the CoDeeN content distribution network. In *Proc. Usenix Annual Technical Conference*, pages 171–184, Boston, MA, June 2004.

[42] J. Yin, L. Alvisi, M. Dahlin, and A. Iyengar. Engineering Web cache consistency. *ACM*

Transactions on Internet Technologies, 2(3):224–259, August 2002.

[43] H. Yu and A. Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Transactions on Computer Systems*, 20(3):239–282, August 2002.

[44] M. Zari, H. Saiedian, and M. Naeem. Understanding and reducing Web delays. *IEEE Computer*, 34(12):30–37, December 2001.

[45] W. Zhao and H. Schulzrinne. DotSlash: A self-configuring and scalable rescue system for handling web hotspots effectively. In *Proc. Intl. Workshop on Web Caching and Content Distribution*, pages 1–18, Beijing, China, October 2004.