Web Replica Hosting Systems Design

Swaminathan Sivasubramanian
Michał Szymaniak
Guillaume Pierre
Maarten van Steen
Dept. of Computer Science
Vrije Universiteit, Amsterdam, The Netherlands
{swami,michal,gpierre,steen}@cs.vu.nl

June 14, 2004

Internal Report IR-CS-001
(Revised May 2004)

Abstract

Replication is a well-known technique to improve the accessibility of Web sites. It generally offers reduced client latencies and increases a site's availability. However, applying replication techniques is not trivial, and various Content Delivery Networks (CDNs) have been created to facilitate replication for digital content providers. The success of these CDNs has triggered further research efforts into developing advanced *Web replica hosting systems*. These are systems that host the documents of a Web site and manage replication automatically. To identify the key issues in designing a wide-area replica hosting system, we present an architectural framework. The framework assists in characterizing different systems in a systematic manner. We categorize different research efforts and review their relative merits and demerits. As an important side-effect, this review and characterization shows that there a number of interesting research questions that have not received much attention yet, but which deserve exploration by the research community.

Vrije Universiteit
Department of Computer Science

Contents

1 Introduction

Replication is a technique that allows to improve the quality of distributed services. In the past few years, it has been increasingly applied to Web services, notably for hosting Web sites. In such cases, replication involves creating copies of a site's Web documents, and placing these document copies at well-chosen locations. In addition, various measures are taken to ensure (possibly different levels of) 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 replica does not result in entire service outage.

These advantages motivate many Web content providers to offer their services using systems that use replication techniques. We refer to systems providing such hosting services as *replica hosting systems*. The design space for replica hosting systems is big and seemingly complex. In this paper, we concentrate on organizing this design space and review several important research efforts concerning the development of Web replica hosting systems. A typical example of such a system is a Content Delivery Network (CDN) [Hull 2002; Rabinovich and Spastscheck 2002; Verma 2002].

There exists a wide range of articles discussing selected aspects of Web replication. However, to the best of our knowledge there is no single framework that aids in understanding, analyzing and comparing the efforts conducted in this area. In this paper, we provide a framework that covers the important issues that need to be addressed in the design of a Web replica hosting system. The framework is built around an *objective function* – a general method for evaluating the system performance. Using this objective function, we define the role of the different system components that address separate issues in building a replica hosting system.

The Web replica hosting systems we consider are scattered across a large geographical area, notably the Internet. When designing such a system, at least the following five issues need to be addressed:

1. How do we *select and estimate the metrics* for taking replication decisions?
2. *When* do we replicate a given Web document?
3. *Where* do we place the replicas of a given document?
4. How do we *ensure consistency* of all replicas of the same document?
5. How do we *route client requests* to appropriate replicas?

Each of these five issues is to a large extent independent from the others. Once grouped together, they address all the issues constituting a generalized framework of a Web replica hosting system. Given this framework, we compare and combine several existing research efforts, and identify problems that have not been addressed by the research community before.

Another issue that should also be addressed separately is *selecting the objects* to replicate. Object selection is directly related to the granularity of replication. In practice, whole Web sites are taken as the unit for replication, but Chen et al. [2002; 2003] show that grouping Web documents can considerably improve the performance of replication schemes at relatively low costs. However, as not much work has been done in this area, we have chosen to exclude object selection from our study.

We further note that Web caching is an area closely related to replication. In caching, whenever a client requests a document for the first time, the client process or the local server handling the request will fetch a copy from the document's server. Before passing it to the client, the document is stored locally in a cache. Whenever that document is requested again, it can be fetched from the cache locally. In replication, a document's server pro-actively places copies of document at various servers, anticipating that enough clients will make use of this copy. Caching and replication thus differ only in the method of creation of copies. Hence, we perceive caching infrastructures (like, for example, Akamai [Dilley et al. 2002]) also as replica hosting systems, as document distribution is initiated by the server. For more information on traditional Web caching, see [Wang 1999]. A survey on hierarchical and distributed Web caching can be found in [Rodriguez et al. 2001].

A complete design of a Web replica hosting system cannot restrict itself to addressing the above five issues, but should also consider other aspects. The two most important ones are *security* and *fault tolerance*. From a security perspective, Web replica hosting systems provide a solution to denial-of-service (DoS) attacks. By simply replicating content, it becomes much more difficult to prevent access to specific Web content. On the other hand, making a Web replica hosting system secure is currently done by using the same techniques as available for Web site security. Obviously, wide-spread Web content replication poses numerous security issues, but these have not been sufficiently studied to warrant inclusion in a survey such as this.

In contrast, when considering fault tolerance, we face problems that have been extensively studied in the past decades with many solutions that are now being incorporated into highly replicated systems such as those studied here. Notably the solutions for achieving high availability and fault tolerance of a single site are orthogonal to achieving higher performance and accessibility in Web replica hosting systems. These solutions have been extensively documented in the literature ([Schneider 1990; Jalote 1994; Pradhan 1996; Alvisi and Marzullo 1998; Elnozahy et al. 2002]), for which reason we do not explicitly address them in our current study.

1.1 Motivating example

To obtain a first impression of the size of the design space, let us consider a few existing systems. We adopt the following model. An *object* encapsulates a (partial) implementation of a service. Examples of an object are a collection of Web documents (such as HTML pages and images) and server scripts (such as ASPs, PHPs) along with their databases. An object can exist in multiple copies, called *replicas*. Replicas are stored on *replica servers*, which together form a *replica hosting system*. The replica hosting system delivers appropriate responses to its *clients*. The response usually comprises a set of documents generated by the replica, statically or dynamically. The time between a client issuing a request and receiving the corresponding response is defined as *client latency*. When the state of a replica is updated, then this update needs to propagate to other replicas so that no stale document is delivered to the clients. We refer to the process of ensuring that all replicas of an object have the same state amidst updates as *consistency enforcement*.

Now consider the following four different CDNs, which we discuss in more detail below. Akamai is one of the largest CDNs currently deployed, with tens of thousands of replica servers placed all over the Internet [Dilley et al. 2002]. To a large extent, Akamai uses well-known technology to replicate content, notably a combination of DNS-based redirection and proxy-caching techniques. A different approach is followed by Radar, a CDN developed at AT&T [Rabinovich and Aggarwal 1999]. A distinguishing feature of Radar is that it can also perform migration of content in addition to replication. SPREAD [Rodriguez and Sibal 2000] is a CDN also developed at AT&T that attempts to transparently replicate content through interception of network traffic. Its network-level approach leads to the application of different techniques. Finally, the Globule system is a research CDN aiming to support large-scale user-initiated content distribution [Pierre and van Steen 2001; Pierre and van

Table 1: Summary of strategies adopted by four different CDNs

Design issue	Akamai	Radar	SPREAD	Globule
Replica placement	Caching	Content is shipped to replica servers located closer to clients	Replicas created on paths between clients and the original document server	Selects the best replication strategy regularly upon evaluating different strategies
Consistency enforcement	Consistency based on replica versioning	Primary-copy approach	Different strategies, chosen on a per-document basis	Adaptive consistency policies
Adaptation Triggering	Primarily server-triggered	Server-triggered	Router-triggered	Server-triggered
Request Routing	Two-tier DNS redirection combined with URL rewriting, considering server load and network-related metrics	Proprietary redirection mechanism, considering server load and proximity	Packet-handoff, considering hop-based proximity	Single-tier DNS redirection, considering AS-based proximity

Steen 2003].

Table 1 gives an overview of how these four systems deal with the aforementioned five issues: replica placement, consistency enforcement, request routing, and adaptation triggering. Let us consider each of these entries in more detail.

When considering replica placement, it turns out that this is not really an issue for Akamai because it essentially follows a caching strategy. However, with this scheme, client redirection is an important issue as we shall see. Radar follows a strategy that assumes that servers are readily available. In this case, replica placement shifts to the problem of deciding what the best server is to place content. Radar simply replicates or migrates content close to where many clients are. SPREAD considers the network path between clients and the original document server and makes an attempt to create replicas on that path. Globule, finally, also assumes that servers are readily available, but performs a global evaluation taking into account, for example, that the number of replicas needs to be restricted. Moreover, it does such evaluations on a per-document basis.

Consistency enforcement deals with the techniques that are used to keep replicas consistent. Akamai uses an elegant versioning scheme by which the version is encoded in a document's name. This approach creates a cache miss in the replica server whenever a new version is created, so that the replica server will fetch it from the document's main server. Radar simply applies a primary-copy protocol to keep replicas consistent. SPREAD deploys different techniques, and has in common with Globule that different techniques can be used for different documents. Globule, finally, evaluates whether consistency requirements are continued to be met, and, if necessary, switches to a different technique for consistency enforcement.

Adaptation triggering concerns selecting the component that is responsible for monitoring system conditions, and, if necessary, initiating adaptations to, for example, consistency enforcement and replica placement. All systems except SPREAD put servers in control for monitoring and adaptation triggering. SPREAD uses routers for this purpose.

Request routing deals with redirecting a client to the best replica for handling its request. There are several ways to do this redirection. Akamai and Globule both use the Domain Name System (DNS [Mockapetris 1987a]) for this purpose, but in a different way. Akamai takes into account server load and various network metrics, whereas Globule currently measures only the number of autonomous systems that a request needs to pass through. Radar deploys a proprietary solution taking into account server load and a Radar-specific proximity measure. Finally, SPREAD deploys a network packet-handoff mechanism, actually using router hops as its distance metric.

Without having gone into the details of these systems, it can be observed that approaches to organizing a CDN differ widely. As we discuss in this paper, there are numerous solutions to the various design questions that need to be addressed when developing a replica hosting service. These questions and their solutions are the topic of this paper.

1.2 Contributions and organization

The main contributions of this paper are three-fold. First, we identify and enumerate the issues in building a Web replica hosting system. Second, we propose an architectural framework for such systems and review important research work in the context of the proposed framework. Third, we identify some of the problems that have not been addressed by the research community until now, but whose solutions can be of value in building a large-scale replica hosting system.

The rest of the paper is organized as follows. In Section 2 we present our framework of wide-area replica hosting systems. In Sections 3 to 7, we discuss each of the above mentioned five problems forming the framework. For each problem, we refer to some of the significant related research efforts, and show how the problem was tackled. We draw our conclusions in Section 8.

2 Framework

The goal of a replica hosting system is to provide its clients with the best available performance while consuming as little resources as possible. For example, hosting replicas on many servers spread throughout the Internet can decrease the client end-to-end latency, but is bound to increase the operational cost of the system. Replication can also introduce costs and difficulties in maintaining consistency among replicas, but the system should always continue to meet application-specific consistency constraints. The design of a replica hosting system is the result of compromises between performance, cost, and application requirements.

2.1 Objective function

In a sense, we are dealing with an optimization problem, which can be modeled by means of an abstract *objective function*, F_{ideal}, whose value λ is dependent on many input parameters:

$$\lambda = F_{ideal}(p_1, p_2, p_3, \ldots, p_n)$$

In our case, the objective function takes two types of input parameters. The first type consists of uncontrollable system parameters, which cannot be directly controlled by the replica hosting system.

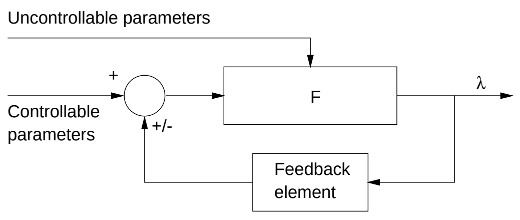

Figure 1: The feedback control loop for a replica hosting system.

Typical examples of such uncontrollable parameters are client request rates, update rates for Web documents, and available network bandwidth. The second type of input parameters are those whose value can be controlled by the system. Examples of such parameters include the number of replicas, the location of replicas, and the adopted consistency protocols.

One of the problems that replica hosting systems are confronted with, is that to achieve optimal performance, only the controllable parameters can be manipulated. As a result, continuous feedback is necessary, resulting in a traditional feedback control system as shown in Figure 1.

Unfortunately, the actual objective function F_{ideal} represents an ideal situation, in the sense that the function is generally only implicitly known. For example, the actual dimension of λ may be a complex combination of monetary revenues, network performance metrics, and so on. Moreover, the exact relationship between input parameters and the observed value λ may be impossible to derive. Therefore, a different approach is always followed by constructing an objective function F whose output λ is compared to an assumed optimal value λ^* of F_{ideal}. The closer λ is to λ^*, the better. In general, the system is considered to be in an *acceptable state*, if $|\lambda^* - \lambda| \leq \delta$, for some system-dependent value δ.

We perceive any large-scale Web replica hosting system to be constantly adjusting its controllable parameters to keep λ within the acceptable interval around λ^*. For example, during a flash crowd (a sudden and huge increase in the client request rate), a server's load increases, in turn increasing the time needed to service a client. These effects may result in λ falling out of the acceptable interval and that the system must adjust its controllable parameters to bring λ back to an acceptable value. The actions on controllable parameters can be such as increasing the number of replicas, or placing replicas close to the locations that generate most requests. The exact definition of the objective function F, its input parameters, the optimal value λ^*, and the value of δ are defined by the system designers and will generally be based on application requirements and constraints such as cost.

In this paper, we use this notion of an objective function to describe the different components of a replica hosting system, corresponding to the different parts of the system design. These components cooperate with each other to optimize λ. They operate on the controllable parameters of the objective function, or observe its uncontrollable parameters.

2.2 Framework elements

We identify five main issues that have to be considered during the design of a replica hosting system: metric determination, adaptation triggering, replica placement, consistency enforcement, and request routing. These issues can be treated as chronologically ordered steps that have to be taken when transforming a centralized service into a replicated one. Our proposed framework of a replica hosting system matches these five issues as depicted in Figure 2. Below we discuss the five issues and show how each of them is related to the objective function.

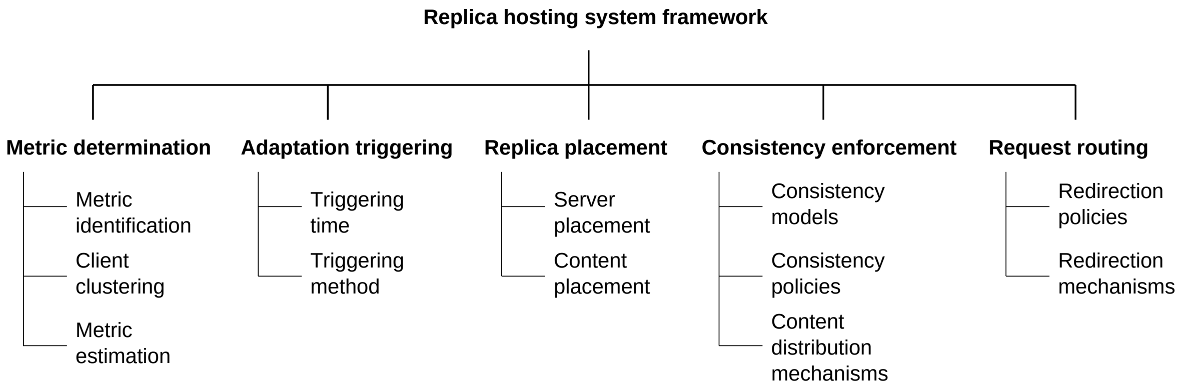

Figure 2: A framework for evaluating wide-area replica hosting systems.

In **metric determination**, we address the question how to find and estimate the metrics required by different components of the system. Metric determination is the problem of estimating the value of the objective function parameters. We discuss two important issues related to metric estimation that need to be addressed to build a good replica hosting system. The first issue is *metric identification*: the process of identifying the metrics that constitute the objective function the system aims to optimize. For example, a system might want to minimize client latency to attract more customers, or might want to minimize the cost of replication. The other important issue is the process of *metric estimation*. This involves the design of mechanisms and services related to estimation or measurement of metrics in a scalable manner. As a concrete example, measuring client latency to *every* client is generally not scalable. In this case, we need to group clients into clusters and measure client-related metrics on a per-cluster basis instead of on a per-client basis (we call this process of grouping clients *client clustering*). In general, the metric estimation component measures various metrics needed by other components of the replica hosting system.

Adaptation triggering addresses the question when to adjust or adapt the system configuration. In other words, we define when and how we can detect that λ has drifted too much from λ^*. Consider a flash crowd causing poor client latency. The system must identify such a situation and react, for example, by increasing the number of replicas to handle the increase in the number of requests. Similarly, congestion in a network where a replica is hosted can result in poor accessibility of that replica. The system must identify such a situation and possibly move that replica to another server. The adaptation-triggering mechanisms do not form an input parameter of the objective function. Instead, they form the heart of the feedback element in Figure 1, thus indirectly control λ and maintain the system in an acceptable state.

With **replica placement** we address the question where to place replicas. This issue mainly concerns two problems: selection of locations to install replica servers that can host replicas (*replica server placement*) and selection of replica servers to host replicas of a given object (*replica content placement*). The server placement problem must be addressed during the initial infrastructure installation and during the hosting infrastructure upgrading. The replica content placement algorithms are executed to ensure that content placement results in an acceptable value of λ, given a set of replica servers. Replica placement components use metric estimation services to get the value of metrics required by their placement algorithms. Both *replica server placement* and *replica content placement* form controllable input parameters of the objective function.

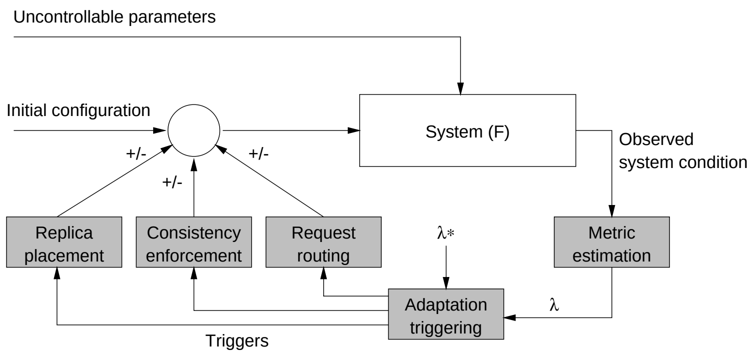

Figure 3: Interactions between different components of a wide-area replica hosting system.

With **consistency enforcement** we consider how to keep the replicas of a given object consistent. Maintaining consistency among replicas adds overhead to the system, particularly when the application requires strong consistency (meaning clients are intolerant to stale data) and the number of replicas is large. The problem of consistency enforcement is defined as follows. Given certain application consistency requirements, we must decide what *consistency models*, *consistency policies* and *content distribution mechanisms* can meet these requirements. A consistency model dictates the consistency-related properties of content delivered by the systems to its clients. These models define consistency properties of objects based on time, value, or the order of transactions executed on the object. A consistency model is usually adopted by consistency policies, which define how, when, and which content distribution mechanisms must be applied. The content distribution mechanisms specify the protocols by which replica servers exchange updates. For example, a system can adopt a time-based consistency model and employ a policy where it guarantees its clients that it will never serve a replica that is more than an hour older than the most recent state of the object. This policy can be enforced by different mechanisms.

Request routing is about deciding how to direct clients to the replicas they need. We choose from a variety of *redirection policies* and *redirection mechanisms*. Whereas the mechanisms provide a method for informing clients about replica locations, the policies are responsible for determining which replica must serve a client. The request routing problem is complementary to the placement problem, as the assumptions made when solving the latter are implemented by the former. For example, we can place replica servers close to our clients, assuming that the redirection policy directs the clients to their nearby replica servers. However, deliberately drifting away from these assumptions can sometimes help in optimizing the objective function. For example, we may decide to direct some client requests to more distant replica servers to offload the client-closest one. Therefore, we treat *request routing* as one of the (controllable) objective function parameters.

Each of the above design issues corresponds to a single *logical* system component. How each of them is actually realized can be very different. The five components together should form a scalable Web replica hosting system. The interaction between these components is depicted in Figure 3, which is a refinement of our initial feedback control system shown in Figure 1. We assume that λ^* is a function of the uncontrollable input parameters, that is:

$$\lambda^* = \min_{p_{k+1},\cdots,p_n} F(\underbrace{p_1,\ldots,p_k}_{\substack{\text{Uncontrollable}\\ \text{parameters}}},\underbrace{p_{k+1},\ldots,p_n}_{\substack{\text{Controllable}\\ \text{parameters}}})$$

Its value is used for adaptation triggering. If the difference with the computed value λ is too high, the triggering component initiates changes in one or more of the three *control components*: replica placement, consistency enforcement, or request routing. These different components strive to maintain λ close to λ^*. They manage the controllable parameters of the objective function, now represented by the actually built system. Of course, the system conditions are also influenced by the uncontrollable parameters. The system condition is measured by the metric estimation services. They produce the current system value λ, which is then passed to the adaptation triggering component for subsequent comparison. This process of adaptation continues throughout the system's lifetime.

Note that the metric estimation services are also being used by components for replica placement, consistency enforcement, and request routing, respectively, for deciding on the quality of their decisions. These interactions are not shown in the figure for sake of clarity.

3 Metric determination

All replica hosting systems need to adapt their configuration in an effort to maintain high performance while meeting application constraints at minimum cost. The metric determination component is required to measure the system condition, and thus allow the system to detect when the system quality drifts away from the acceptable interval so that the system can adapt its configuration if necessary.

Another purpose of the metric determination component is to provide each of the three control components with measurements of their input data. For example, replica placement algorithms may need latency measurements in order to generate a placement that is likely to minimize the average latency suffered by the clients. Similarly, consistency enforcement algorithms might require information on object staleness in order to react with switching to stricter consistency mechanisms. Finally, request routing policies may need to know the current load of replica servers in order to distribute the requests currently targeting heavily loaded servers to less loaded ones.

In this section, we discuss three issues that have to be addressed to enable scalable metric determination. The first issue is *metric selection*. Depending on the performance optimization criteria, a number of metrics must be carefully selected to accurately reflect the behavior of the system. Section 3.1 discusses metrics related to client latency, network distance, network usage, object hosting cost, and consistency enforcement.

The second issue is *client clustering*. Some client-related metrics should ideally be measured separately for each client. However, this can lead to scalability problems as the number of clients for a typical wide-area replica hosting system can be in the order of millions. A common solution to address this problem is to group clients into clusters and measure client-related metrics on a per-cluster basis. Section 3.2 discusses various client clustering schemes.

The third issue is *metric estimation* itself. We must choose mechanisms to collect metric data. These mechanisms typically use client-clustering schemes to estimate client-related metrics. Section 3.3 discusses some popular mechanisms that collect metric data.

3.1 Choice of metrics

The choice of metrics must reflect all aspects of the desired performance. First of all, the system must evaluate all metrics that take part in the computation of the objective function. Additionally, the system also needs to measure some extra metrics needed by the control components. For example, a map of host-to-host distances may help the replica placement algorithms, although it does not have to be used by the objective function.

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

Table 2: Five different classes of metrics used to evaluate performance in replica hosting systems.

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, the metrics related to latency, distance, and consistency can help evaluate the client-perceived performance. Similarly, the 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. We distinguish five classes of metrics, as shown in Figure 2, and which are discussed in the following sections.

3.1.1 Temporal metrics

An important class of metrics is related to the *time* it takes for peers to communicate, generally referred to as latency metrics. Latency can be defined in different ways. To explain, we consider a client-server system and follow the approach described in [Dykes et al. 2000] by modeling the total time to process a request, as seen from the client's perspective, as

$$T = T_{DNS} + T_{conn} + T_{res} + T_{rest}$$

T_{DNS} is the DNS lookup time needed to find the server's network address. As reported by Cohen and Kaplan [2001], DNS lookup time can vary tremendously due to cache misses (i.e., the client's local DNS server does not have the address of the requested host in its cache), although in many cases it stays below 500 milliseconds.

T_{conn} is the time needed to establish a TCP connection, which, depending on the type of protocols used in a replica hosting system, may be relevant to take into account. Zari et al. [2001] report that T_{conn} will often be below 200 milliseconds, but that, like in the DNS case, very high values up to even 10 seconds may also occur.

T_{res} is the time needed to transfer a request from the client to the server and receiving the first byte of the response. This metric is comparable to measuring the round-trip time (RTT) between two nodes, but includes the time the server needs to handle the incoming request. Finally, T_{rest} is the time needed to complete the transfer of the entire response.

When considering latency, two different versions are often considered. The *end-to-end latency* is taken as the time needed to send a request to the server, and is often taken as $0.5T_{res}$, possibly including the time T_{conn} to setup a connection. The *client-perceived latency* is defined as $T - T_{rest}$. This latter latency metric reflects the real delay observed by a user.

Obtaining accurate values for latency metrics is not a trivial task as it may require specialized mechanisms, or even a complete infrastructure. One particular problem is predicting client-perceived latency, which not only involves measuring the round-trip delay between two nodes (which is independent of the size of the response), but may also require measuring bandwidth to determine T_{rest}. The latter has shown to be particularly cumbersome requiring sophisticated techniques [Lai and Baker 1999]. We discuss the problem of latency measurement further in Section 3.3.

3.1.2 Spatial metrics

As an alternative to temporal metrics, many systems consider a spatial metric such as number of network-level hops or hops between autonomous systems, or even the geographical distance between two nodes. In these cases, the underlying assumption is generally that there exists a map of the network in which the spatial metric can be expressed.

Maps can have different levels of accuracy. Some maps depict the Internet as a graph of Autonomous Systems (ASes), thus unifying all machines belonging to the same AS. They are used for example by Pierre et al. [2002]. The graph of ASes is relatively simple and easy to operate on. However, because ASes significantly vary in size, this approach can suffer from inaccuracy. Other maps depict the Internet as a graph of routers, thus unifying all machines connected to the same router [Pansiot and Grad 1998]. These maps are more detailed than the AS-based ones, but are not satisfying predictors for latency. For example, Huffaker et al. [2002] found that the number of router hops is accurate in selecting the closest server in only 60% of the cases. The accuracy of router-level maps can be expected to decrease in the future with the adoption of new routing technologies such as Multi-Protocol Label Switching (MPLS) [Pepelnjak and Guichard 2001], which can hide the routing paths within a network. Finally, some systems use proprietary distance calculation schemes, for example by combining the two above approaches [Rabinovich and Aggarwal 1999].

Huffaker et al. [2002] examined to what extent geographical distance could be used instead of latency metrics. They showed that there is generally a close correlation between geographical distance and RTT. An earlier study using simple network measurement tools by Ballintijn et al. [2000], however, reported only a weak correlation between geographical distance and RTT. This difference may be caused by the fact that many more monitoring points *outside* the U.S. were used, but that many physical connections actually cross through networks located *in* the U.S. This phenomenon also caused large deviations in the results by Huffaker et al.

An interesting approach based on geographical distance is followed in the Global Network Positioning (GNP) project [Ng and Zhang 2002]. In this case, the Internet is modeled as an N-dimensional geometric space. GNP is used to estimate the latency between two arbitrary nodes. We describe GNP and its several variants in more detail in Section 3.3 when discussing metric estimation services.

Constructing and exploiting a map of the Internet is easier than running an infrastructure for latency measurements. The maps can be derived, for example, from routing tables. Interestingly, Crovella and Carter [1995] reported that the correlation between the distance in terms of hops and the latency is quite poor. However, other studies show that the situation has changed. McManus [1999] shows that the number of hops between ASes can be used as an indicator for client-perceived latency. Research reported in [Obraczka and Silva 2000] revealed that the correlation between the number of network-level or AS-level hops and round-trip times (RTTs) has further increased, but that RTT still remains the best choice when a single latency metric is needed for measuring client-perceived performance.

3.1.3 Network usage metrics

Another important metric is the total amount of consumed network resources. Such resources could include routers and other network elements, but often entails only the consumed bandwidth. The total network usage can be classified into two types. Internal usage is caused by the communication between replica servers to keep replicas consistent. External usage is caused by communication between clients and replica servers. Preferably, the ratio between external and internal usage is high, as internal usage can be viewed as a form of overhead introduced merely to keep replicas consistent. On the other hand, overall network usage may decrease in comparison to the non-replicated case, but may require measuring more than, for example, consumed bandwidth only.

To see the problem at hand, consider a non-replicated document of size s bytes that is requested r times per seconds. The total consumed bandwidth in this case is $r \cdot s$, plus the cost of r separate connections to the server. The cost of a connection can typically be expressed as a combination of setup costs and the average distance that each packet associated with that connection needs to travel. Assume that the distance is measured in the number of hops (which is reasonable when considering network usage). In that case, if l_r is the average length of a connection, we can also express the total consumed bandwidth for reading the document as $r \cdot s \cdot l_r$.

On the other hand, suppose the document is updated w times per second and that updates are always immediately pushed to, say, n replicas. If the average path length for a connection from the server to a replica is l_w, update costs are $w \cdot s \cdot l_w$. However, the average path length of a connection for reading a document will now be lower in comparison to the non-replicated case. If we assume that $l_r = l_w$, the total network usage may change by a factor w/r in comparison to the non-replicated case.

Of course, more precise models should be applied in this case, but the example illustrates that merely measuring consumed bandwidth may not be enough to properly determine network usage. This aspect becomes even more important given that pricing may be an issue for providing hosting services, an aspect that we discuss next.

3.1.4 Financial metrics

Of a completely different nature are metrics that deal with the economics of content delivery networks. To date, such metrics form a relatively unexplored area, although there is clearly interest to increase our insight (see, for example, [Janiga et al. 2001]). We need to distinguish at least two different roles. First, the owner of the hosting service is confronted with costs for developing and maintaining the hosting service. In particular, costs will be concerned with server placement, server capacity, and network resources (see, e.g., [Chandra et al. 2001]). This calls for metrics aimed at the hosting service provider.

The second role is that of customers of the hosting service. Considering that we are dealing with shared resources that are managed by service provider, accounting management by which a precise record of resource consumption is developed, is important for billing customers [Aboba et al. 2000]. However, developing pricing models is not trivial and it may turn out that simple pricing schemes will dominate the sophisticated ones, even if application of the latter are cheaper for customers [Odlyzko 2001]. For example, Akamai uses peak consumed bandwidth as its pricing metric.

The pricing model for hosting an object can directly affect the control components. For example, a model can mandate that the number of replicas of an object is constrained by the money paid by the object owner. Likewise, there exist various models that help in determining object hosting costs. Examples include a model with a flat base fee and a price linearly increasing along with the number of object replicas, and a model charging for the total number of clients serviced by all the object replicas.

We observe that neither financial metrics for the hosting service provider nor those for consumers have actually been established other than in some ad hoc and specific fashion. We believe that developing such metrics, and notably the models to support them, is one of the more challenging and interesting areas for content delivery networks.

3.1.5 Consistency metrics

Consistency metrics inform to what extent the replicas retrieved by the clients are consistent with the replica version that was up-to-date at the moment of retrieval. Many consistency metrics have been proposed and are currently in use, but they are usually quantified along three different axes.

In *time-based* consistency models, the difference between two replicas A and B is measured as the time between the latest update on A and the one on B. In effect, time-based models measure the staleness of a replica in comparison to another, more recently updated replica. Taking time as a consistency metric is popular in Web-hosting systems as it is easy to implement and independent of the semantics of the replicated object. Because updates are generally performed at only a single primary copy from where they are propagated to secondaries, it is easy to associate a single timestamp with each update and to subsequently measure the staleness of a replica.

In *value-based* models, it is assumed that each replica has an associated numerical value that represents its current content. Consistency is then measured as the numerical difference between two replicas. This metric requires that the semantics of the replicated object are known or otherwise it would be impossible to associate and compare object values. An example of where value-based metrics can be applied is a stock-market Web document containing the current values of shares. In such a case, we could define a Web document to be inconsistent if at least one of the displayed shares differs by more than 2% with the most recent value.

Finally, in *order-based* models, reads and writes are perceived as transactions and replicas can differ only in the order of execution of write transactions according to certain constraints. These constraints can be defined as the allowed number of out-of-order transactions, but can also be based on the dependencies between transactions as is commonly the case for distributed shared-memory systems [Mosberger 1993], or client-centric consistency models as introduced in Bayou [Terry et al. 1994].

3.1.6 Metric classification

Metrics can be classified into two types: static and dynamic. Static metrics are those whose estimates do not vary with time, as opposed to dynamic metrics. Metrics such as the geographical distance are static in nature, whereas metrics such as end-to-end latency, number of router hops or network usage are dynamic. The estimation of dynamic metrics can be a difficult problem as it must be performed regularly to be accurate. Note that determining how often a metric should be estimated is a problem by itself. For example, Paxson [1997a] found that the time periods over which end-to-end routes persist vary from seconds to days.

Dynamic metrics can be more useful when selecting a replica for a given client as they estimate the current situation. For example, Crovella and Carter [1995] conclude that the use of a dynamic metric instead of a static one is more useful for replica selection as the former can also account for dynamic factors such as network congestion. Static metrics, in turn, are likely to be exploited by replica server placement algorithms as they tend to be more directed toward a global, long-lasting situation than an instantaneous one.

In general, however, any combination of metrics can be used by any control component. For example, the placement algorithms proposed by Radoslavov et al. [2001] and Qiu et al. [2001] use dynamic metrics (end-to-end latency and network usage). Also Dilley et al. [2002] and Rabinovich and Aggarwal [1999] use end-to-end latency as a primary metric for determining the replica location. Finally, the request-routing algorithm described in [Szymaniak et al. 2003] exploits network distance measurements. We observe that the existing systems tend to support a small set of metrics, and use all of them in each control component.

3.2 Client clustering

As we noticed before, some metrics should be ideally measured on a per-client basis. In a wide-area replica hosting system, for which we can expect millions of clients, this poses a scalability problem to the estimation services as well as the components that need to use them. Hence, there is a need for scalable mechanisms for metric estimation.

A popular approach by which scalability is achieved is *client clustering* in which clients are grouped into clusters. Metrics are then estimated on a per-cluster basis instead of on a per-client basis. Although this solution allows to estimate metrics in a scalable manner, the efficiency of the estimation depends on the accuracy of clustering mechanisms. The underlying assumption here is that the metric value computed for a cluster is representative of values that would be computed for each individual client in that cluster. Accurate clustering schemes are those which keep this assumption valid.

The choice of a clustering scheme depends on the metric it aims to estimate. Below, we present different kinds of clustering schemes that have been proposed in the literature.

3.2.1 Local name servers

Each Internet client contacts its local DNS server to resolve a service host name to its IP address(es). The clustering scheme based on local name servers unifies clients contacting the same name server, as they are assumed to be located in the same network-topological region. This is a useful abstraction as DNS-based request-routing schemes are already used in the Internet. However, the success of these schemes relies on the assumption that clients and local name servers are close to each other. Shaikh et al. [2001] performed a study on the proximity of clients and their name servers based on the HTTP logs from several commercial Web sites. Their study concludes that clients are typically eight or more hops from their representative name servers. The authors also measured the round trip times both from the name servers to the servers (name-server latency) and from the clients to the servers (client latency). It turns out that the correlation between the name-server latency and the actual client latency is quite poor. They conclude that the latency measurements to the name servers are only a weak approximation of the latency to actual clients. These findings have been confirmed by [Mao et al. 2002].

3.2.2 Autonomous Systems

The Internet has been built as a graph of individual network domains, called Autonomous Systems (ASes). The AS clustering scheme groups together clients located in the same AS, as is done for example, in [Pierre et al. 2002]. This scheme naturally matches the AS-based distance metric. Further clustering can be achieved by grouping ASes into a hierarchy, as proposed by Barford et al. [2001], which in turn can be used to place caches.

Although an AS is usually formed out of a set of networks belonging to a single administrative domain, it does not necessarily mean that these networks are proximal to each other. Therefore, estimating latencies with an AS-based clustering scheme can lead to poor results. Furthermore, since ASes are global in scope, multiple ASes may cover the same geographical area. It is often the case that some IP hosts are very close to each other (either in terms of latency or hops) but belong to different ASes, while other IP hosts are very far apart but belong to the same AS. This makes the AS-based clustering schemes not very effective for proximity-based metric estimations.

3.2.3 Client proxies

In some cases, clients connect to the Internet through *proxies*, which provide them with services such as Web caching and prefetching. Client proxy-based clustering schemes group together all clients using the same proxy into a single cluster. Proxy-based schemes can be useful to measure latency if the clients are close to their proxy servers. An important problem with this scheme is that many clients in the Internet do not use proxies at all. Thus, this clustering scheme will create many clusters consisting of only a single client, which is inefficient with respect to achieving scalability for metric estimation.

3.2.4 Network-aware clustering

Researchers have proposed another scheme for clustering Web clients, which is based on client-network characteristics. Krishnamurthy and Wang [2000] evaluate the effectiveness of a simple mechanism that groups clients having the same first three bytes of their IP addresses into a single cluster. However, this simple mechanism fails in more than 50% of the cases when checking whether grouped clients actually belong to the same network. The authors identify two reasons for failure. First, their scheme wrongly merges small clusters that share the same first three bytes of IP addresses as a single class-C network. Second, it splits several class-A, class-B, and CIDR networks into multiple class-C networks. Therefore, the authors propose a novel method to identify clusters by using the prefixes and network masks information extracted from the Border Gateway Protocol routing tables [Rekhter and Li 1995]. The proposed mechanism consists of the following steps:

1. Creating a merged prefix table from routing table snapshots
2. Performing the longest prefix matching on each client IP address (as routers do) using the constructed prefix table
3. Classifying all the clients which have the same longest prefix into a single cluster.

The authors demonstrate the effectiveness of their approach by showing a success rate of 99.99% in their validation tests.

3.2.5 Hierarchical clustering

Most clustering schemes aim at achieving a scalable manner of metric estimation. However, if the clusters are too coarse grained, it decreases the accuracy of measurement simply because the underlying assumption that the difference between the metric estimated to a cluster and to a client is negligible is no longer valid. Hierarchical clustering schemes help in estimating metrics at different levels (such as intra-cluster and inter-cluster), thereby aiming at improving the accuracy of measurement, as in IDMaps [Francis et al. 2001] and Radar [Rabinovich and Aggarwal 1999]. Performing metric estimations at different levels results not only in better accuracy, but also in better scalability.

Note that there can be other possible schemes of client clustering, based not only on the clients' network addresses or their geographical proximities, but also on their content interests (see, e.g., Xiao and Zhang [2001]). However, such clustering is not primarily related to improving scalability through replication, for which reason we further exclude it from our study.

3.3 Metric estimation services

Once the clients are grouped into their respective clusters, the next step is to obtain the values for metrics (such as latency or network overhead). Estimation of metrics on a wide area scale such as the Internet is not a trivial task and has been addressed by several research initiatives before [Francis et al. 2001; Moore et al. 1996]. In this section, we discuss the challenges involved in obtaining the value for these metrics.

Metric estimation services are responsible for providing values for the various metrics required by the system. These services aid the control components in taking their decisions. For example, these services can provide replica placement algorithms with a map of the Internet. Also, metric estimation services can use client-clustering schemes to achieve scalability.

Metric estimations schemes can be divided into two groups: *active* and *passive* schemes. Active schemes obtain respective metric data by simulating clients and measuring the performance observed by these simulated clients. Active schemes are usually highly accurate, but these simulations introduce additional load to the replica hosting system. Examples of active mechanisms are Cprobes [Carter and Crovella 1997] and Packet Bunch Mode [Paxson 1997b]. Passive mechanisms obtain the metric data from observations of existing system behavior. Passive schemes do not introduce additional load to the network, but deriving the metric data from past events can suffer from poor accuracy. Examples of passive mechanisms include SPAND [Stemm et al. 2000] and EtE [Fu et al. 2002].

Different metrics are by nature estimated in different manners. For 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. In this section, our discussion of existing research efforts mainly covers services that estimate network-related metrics.

3.3.1 IDMaps

IDMaps is an active service that aims at providing an architecture for measuring and disseminating distance information across the Internet [Francis et al. 1999; Francis et al. 2001]. IDMaps uses programs called tracers that collect and advertise distance information as so-called distance maps. IDMaps builds its own client-clustering scheme. It groups different geographical regions as boxes and constructs distance maps between these boxes. The number of boxes in the Internet is relatively small (in the order of thousands). Therefore, building a distance table between these boxes is inexpensive. To measure client-server distance, an IDMaps client must calculate the distance to its own box and the distance from the target server to this server's box. Given these two calculations, and the distance between the boxes calculated based on distance maps, the client can discover its real distance to the server. It must be noted that the efficiency of IDMaps heavily depends on the size and placement of boxes.

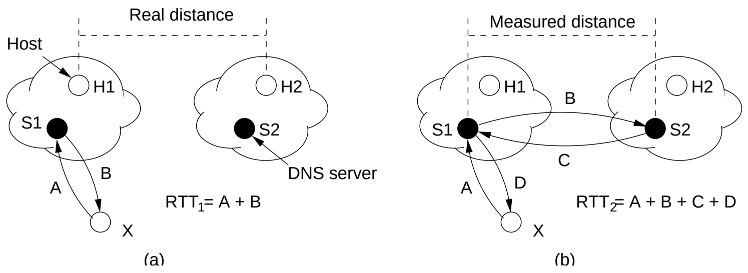

Figure 4: The two DNS queries of King

3.3.2 King

King is an active metric estimation method [Gummadi et al. 2002]. It exploits the global infrastructure of DNS servers to measure the latency between two arbitrary hosts. King approximates the latency between two hosts, H_1 and H_2, with the latency between their local DNS servers, S_1 and S_2.

Assume that a host X needs to calculate the latency between hosts H_1 and H_2. The latency between their local DNS servers, $L_{S_1S_2}$, is calculated based on round-trip times (RTTs) of two DNS queries. With the first query, host X queries the DNS server S_1 about some non-existing DNS name that belongs to a domain hosted by S_1 [see Figure 4(a)]. In this way, X discovers its latency to S_1:

$$L_{XS_1} = \frac{1}{2}RTT_1$$

By querying about a non-existing name, X ensures that the response is retrieved from S_1, as no cached copy of that response can be found anywhere in the DNS.

With the second query, host X queries the DNS server S_1 about another non-existing DNS name that this time belongs to a domain hosted by S_2 (see Figure 4b). In this way, X measures the latency of its route to S_2 that goes through S_1:

$$L_{XS_2} = \frac{1}{2}RTT_2$$

A crucial observation is that this latency is a sum of two partial latencies, one between X and S_1, and the other between S_1 and S_2: $L_{XS_2} = L_{XS_1} + L_{S_1S_2}$ Since L_{XS_1} has been measured by the first DNS query, X may subtract it from the total latency L_{XS_2} to determine the latency between the DNS servers:

$$L_{S_1S_2} = L_{XS_2} - L_{XS_1} = \frac{1}{2}RTT_2 - \frac{1}{2}RTT_1$$

Note that S_1 will forward the second query to S_2 only if S_1 is configured to accept so-called “recursive” queries from X [Mockapetris 1987b].

In essence, King is actively probing with DNS queries. A potential problem with this approach is that an extensive use of King may result in overloading the global infrastructure of DNS servers. In that case, the efficiency of DNS is likely to decrease, which can degrade the performance of the entire Internet. Also, according to the DNS specification, it is recommended to reject recursive DNS queries that come from non-local clients, which renders many DNS servers unusable for King [Mockapetris 1987a].

3.3.3 Network positioning

The idea of network positioning has been proposed in [Ng and Zhang 2002], where it is called Global Network Positioning (GNP). GNP is a novel approach to the problem of network distance estimation,

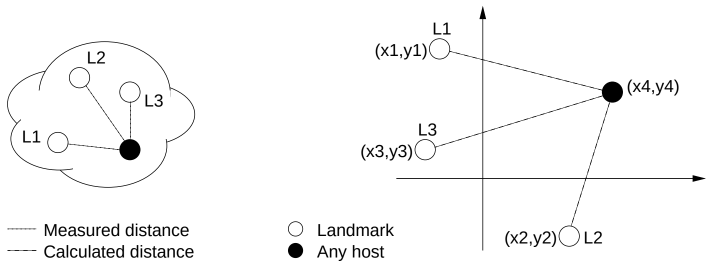

Figure 5: Positioning in GNP

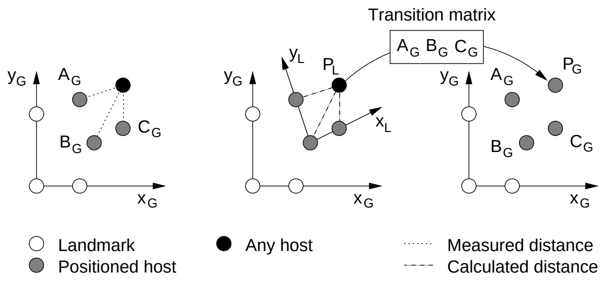

Figure 6: Positioning in Lighthouses

where the Internet is modeled as an N-dimensional geometric space [Ng and Zhang 2002]. GNP approximates the latency between any two hosts as the Euclidean distance between their corresponding positions in the geometric space.

GNP relies on the assumption that latencies can be triangulated in the Internet. The position of any host X is computed based on its measured latencies between X and k *landmark* hosts, whose positions have been computed earlier ($k \geq N+1$, to ensure that the calculated position is unique). By treating these latencies as distances, GNP triangulates the position of X (see Figure 5). The triangulation is implemented by means of Simplex-downhill, which is a classical optimization method for multi-dimensional functions [Nelder and Mead 1965].

The most important limitation of GNP is that the set of landmarks can never change. If any of them becomes unavailable, the latency to that landmark cannot be measured and GNP is no longer able to position any more hosts. It makes GNP sensitive to landmark failures.

This limitation is removed in the Lighthouses system [Pias et al. 2003]. The authors have shown that hosts can be accurately positioned relative to any previously positioned hosts, acting as “local” landmarks. This eliminates the need for contacting the original landmarks each time a host is positioned (see Figure 6). It also allows to improve the positioning accuracy, by selecting some of the local landmarks close to the positioned host [Castro et al. 2003].

SCoLE further improves the scalability of the system by allowing each host to select its own positioning parameters, construct its own “private” space, and position other hosts in that space [Szymaniak et al. 2004]. This effectively removes the necessity of a global negotiation to determine positioning parameters, such as the space dimension, the selection of global landmarks, and the positioning algorithm. Such an agreement is difficult to reach in large-scale systems, where different hosts can have different requirements with respect to the latency estimation process. Latency estimates performed in different private spaces have been shown to be highly correlated, even though these spaces have completely different parameters.

Another approach is to position all hosts simultaneously as a result of a global optimization process [Cox et al. 2004; Shavitt and Tankel 2003; Waldvogel and Rinaldi 2003]. In this case, there is no need to choose landmarks, since every host is in fact considered to be a landmark. The global optimization approach is generally faster than its iterative counterpart, which positions hosts one by one. The authors also claim that it leads to better accuracy, and that it is easy to implement in a completely distributed fashion. However, because it operates on all the latencies simultaneously, it can potentially have to be re-run every time new latency measurements are available. Such a re-run is likely to be computationally expensive in large-scale systems, where the number of performed latency measurements is high.

3.3.4 SPAND

SPAND is a shared passive network performance measurement service [Stemm et al. 2000]. This service aims at providing network-related measures such as client end-to-end latency, available bandwidth, or even application-specific performance details such as access time for a Web object. The components of SPAND are client applications that can log their performance details, a packet-capturing host that logs performance details for SPAND-unaware clients, and performance servers that process the logs sent by the above two components. The performance servers can reply to queries concerning various network-related and application-specific metrics. SPAND has an advantage of being able to produce accurate application-specific metrics if there are several clients using that application in the same shared network. Further, since it employs passive measurement, it does not introduce any additional traffic.

3.3.5 Network Weather Service

The Network Weather Service (NWS) is an active measurement service [Wolski et al. 1999]. It is primarily used in for Grid computing, where decisions regarding scheduling of distributed computations are made based on the knowledge of server loads and several network performance metrics, such as available bandwidth and end-to-end latency. Apart from measuring these metrics, it also employs prediction mechanisms to forecast their value based on past events. In NWS, the metrics are measured using special sensor processes, deployed on every potential server node. Further, to measure end-to-end latency active probes are sent between these sensors. NWS uses an adaptive forecasting approach, in which the service dynamically identifies the model that gives the least prediction error. NWS has also been used for replica selection [McCune and Andresen 1998]. However, exploiting NWS directly by a wide-area replica hosting system can be difficult, as this service does not scale to the level of the Internet. This is due to the fact that it runs sensors in every node and does not use any explicit client clustering schemes. On the other hand, when combined with a good client clustering scheme and careful sensor placement, NWS may become a useful metric estimation service.

3.3.6 Akamai metric estimation

Commercial replica hosting systems often use their own monitoring or metric estimation services. Akamai has built its own distributed monitoring service to monitor server resources, network-related metrics and overall system performance. The monitoring system places monitors in every replica server to measure server resources. The monitoring system simulates clients to determine if the overall system performance is in an acceptable state as perceived by clients. It measures network-related information by employing agents that communicate with border routers in the Internet as peers and derive the distance-related metrics to be used for its placement decisions [Dilley et al. 2002].

3.3.7 Other systems

In addition to the above wide-area metric estimation systems, there are different classes of systems that measure service-related metrics such as content popularity, client-aborted transfers, and amount of consumed bandwidth. These kinds of systems perform estimation in a smaller scale, and mostly measure metrics as seen by a single server.

Web page instrumentation and associated code (e.g., in Javascript) is being used in various commercial tools for measuring service-related metrics. In these schemes, instrumentation code is downloaded by the client browser after which it tracks the download time for individual objects and reports performance characteristics to the Web site.

EtE is a passive system used for measuring metrics such as access latency, and content popularity for the contents hosted by a server [Fu et al. 2002]. This is done by running a special model near the analyzed server that monitors all the service-related traffic. It is capable of determining sources of delay (distinguishing between network and server delays), content popularity, client-aborted transfers and the impact of remote caching on the server performance.

3.4 Discussion

In this section, we discussed three issues related to metric estimation: metric selection, client clustering, and metric estimation.

Metric selection deals with deciding which metrics are important to evaluate system performance. In most cases, optimizing latency is considered to be most important, but the problem is that actually measuring latencies can be quite difficult. For this reason, simpler spatial metrics are used under the assumption that, for example, a low number of network-level hops between two nodes also implies a relatively low latency between those two nodes. Spatial metrics are easier to measure, but also show to be fairly inaccurate as estimators for latency.

An alternative metric is consumed bandwidth, which is also used to measure the efficiency of a system. However, in order to measure the efficiency of a consistency protocol as expressed by the ratio between the consumed bandwidth for replica updates and the bandwidth delivered to clients, some distance metric needs to be taken into account as well. When it comes to consistency metrics, three different types need to be considered: those related to time, value, and the ordering of operations. It appears that this differentiation is fairly complete, leaving the actual implementation of consistency models and the enforcement of consistency the main problem to solve.

An interesting observation is that hardly no systems today use financial metrics to evaluate system performance. Designing and applying such metrics is not obvious, but extremely important in the context of system evaluation.

A scalability problem that these metrics introduce is that they, in theory, require measurements on a per-client basis. With millions of potential clients, such measurements are impossible. Therefore, it is necessary to come to client-clustering schemes. The goal is to design a cluster such that measurement at a single client is representative for any client in that cluster (in a mathematical sense, the clients should form an equivalence class). Finding the right metric for clustering, and also one that can be easily established has shown to be difficult. However, network-aware clustering by which a prefix of the network address is taken as criterion for clustering has lead to very accurate results.

Once a metric has been chosen, its value needs to be determined. This is where metric estimation services come into place. Various services exist, including some very recent ones that can handle difficult problems such as estimating the latency between two arbitrary nodes in the Internet. Again, metric estimation services can turn out to be rather complicated, partly due to the lack of sufficient acquisition points in the Internet. In this sense, the approach to model the Internet as a Euclidean N-dimensional space is very powerful as it allows *local computations* concerning remote nodes. However, this approach can be applied only where the modeled metric can be triangulated, making it more difficult when measuring, for example, bandwidth.

4 Adaptation triggering

The performance of a replica hosting system changes with the variations of uncontrollable system parameters such as client access patterns and network conditions. These changes make the current value of the system's λ drift away from the optimal value λ^*, and fall out of the acceptable interval. The system needs to maintain a desired level of performance by keeping λ in an acceptable range amidst these changes. The adaptation triggering component of the system is responsible for identifying changes in the system and for adapting the system configuration to bound λ within the acceptable range. This adaptation consists of a combination of changes in replica placement, consistency policy, and request routing policy.

We classify adaptation triggering components along two criteria. First, they can be classified based on their timing nature. Second, they can be classified based on which element of the system actually performs the triggering.

4.1 Time-based classification

Taking timing into account, we distinguish three different triggering mechanisms: periodic triggers, aperiodic triggers, and triggers that combine these two.

4.1.1 Periodic triggers

A periodic triggering component analyzes a number of input variables, or λ itself, at fixed time intervals. If the analysis reveals that λ is too far from λ^*, the system triggers the adaptation. Otherwise, it allows the system to continue with the same configuration. Such a periodic evaluation scheme can be effective for systems that have relatively stable uncontrollable parameters. However, if the uncontrollable parameters fluctuate a lot, then it may become very hard to determine a good evaluation periodicity. A too short period will lead to considerable adaptation overhead, whereas a too long period will result in slow reactions to important changes.

4.1.2 Aperiodic triggers

Aperiodic triggers can trigger adaptation at any time. A trigger is usually due to an event indicating a possible drift of λ from the acceptable interval. Such events are often defined as changes of the uncontrollable parameters, such as the client request rates or end-to-end latency, which may reflect issues that the system has to deal with.

The primary advantage of aperiodic triggers is their responsiveness to emergency situations such as flash crowds where the system must be adapted quickly. However, it requires continuous monitoring of metrics that can indicate events in the system, such as server load or client request rate.

4.1.3 Hybrid triggers

Periodic and aperiodic triggers have opposite qualities and drawbacks. Periodic triggers are well suited for detecting slow changes in the system that aperiodic triggers may not detect, whereas aperiodic triggers are well suited to detect emergency situations where immediate action is required. Consequently, a good approach may be a combination of periodic and aperiodic triggering schemes. For example, Radar and Globule use both periodic and aperiodic triggers, which give them the ability to perform global optimizations and to react to emergency situations.

In Radar [Rabinovich and Aggarwal 1999], every replica server periodically runs a replication algorithm that checks for the number of client accesses to a particular replica and server load. An object is deleted for low client accesses and a migration/replication component is invoked if the server load is above a threshold. In addition, a replica server can detect that it is overloaded and ask its replication-managing component to offload it. Adaptation in this case consists either of distributing the load over other replica servers, or to propagate the request to another replication-managing component in case there are not enough replica servers available.

In Globule [Pierre and van Steen 2001], each primary server periodically evaluates recent client access logs. The need for adapting the replication and consistency policies is determined by this evaluation. Similarly to Radar, each replica server also monitors its request rate and response times. When a server is overloaded, it can request its primary server to reevaluate the replication strategy.

4.2 Source-based classification

Adaptation triggering mechanisms also vary upon which part of the system actually performs the triggering. We describe three different kinds of mechanisms.

4.2.1 Server-triggered adaptation

Server-triggered adaptation schemes consider that replica servers are in a good position to evaluate metrics from the system. Therefore, the decision that adaptation is required is taken by one or more replica servers.

Radar and Globule use server-triggered adaptation, as they make replica servers monitor and possibly react to system conditions.

Server-triggered adaptation is also well suited for reacting to internal server conditions, such as overloads resulting from flash crowds or denial-of-service (DoS) attacks. For example, Jung et al. [2002] studied the characteristics of flash crowds and DoS attacks. They propose an adaptation scheme where servers can differentiate these two kinds of events and react accordingly: increase the number of replicas in case of a flash crowd, or invoke security mechanisms in case of a DoS attack.

Server-triggered adaptation is effective as the servers are in a good position to determine the need for changes in their strategies in view of other constraints, such as total system cost. Also, these mechanisms do not require running triggering components on elements (hosts, routers) that may not be under the control of the replica hosting system.

4.2.2 Client-triggered adaptation

Adaptation can be triggered by the clients. In client-triggered schemes, clients or client representatives can notice that they experience poor quality of service and request the system to take the appropriate measures. Sayal et al. [2003] describe such a system where smart clients provide the servers with feedback information to help take replication decisions.

Client-triggered adaptation can be efficient in terms of preserving a client's QoS. However, it has three important drawbacks. First, the clients or client representatives must cooperate with the replica hosting system. Second, client transparency is lost, as clients or their representatives need to monitor events and take explicit action. Third, by relying on individual clients to trigger adaptation, this scheme may suffer from poor scalability in a wide-area network, unless efficient client clustering methods are used.

4.2.3 Router-triggered adaptation

In router-triggered schemes, adaptation is initiated by the routers that can inform the system of network congestion, link and network failures, or degraded end-to-end request latencies. These schemes observe network-related metrics and operate on them.

Such an adaptation scheme is used in SPREAD [Rodriguez and Sibal 2000]. In SPREAD, every network has one special router with a distinguished proxy attached to it. If the router notices a TCP communication from a client to retrieve data from a primary server, it intercepts this communication and redirects the request to the proxy attached to it. The proxy gets a copy of the referenced object from the primary server and services this client and all future requests passing through its network. By using the network layer to implement replication, this scheme builds an architecture that is transparent to the client.

Router-triggered schemes have the advantage that routers are in a good position to observe network-related metrics, such as end-to-end latency and consumed bandwidth while preserving client transparency. Such schemes are useful to detect network congestion or dead links, and thus may trigger changes in replica location. However, they suffer from two disadvantages. First, they require the support of routers, which may not be available to every enterprise building a replica hosting system in the Internet. Second, they introduce an overhead to the network infrastructure, as they need to isolate the traffic targeting Web hosting systems, which involves processing all packets received by the routers.

4.3 Discussion

Deciding when to trigger system adaptation is difficult because explicitly computing λ and λ^* may be expensive, if not impossible. This calls for schemes that are both responsive enough to detect the drift of λ from the acceptable interval and are computationally inexpensive. This is usually realized by monitoring simple metrics which are believed to significantly influence λ.

Another difficulty is posed by the fact that it is not obvious which adaptive components should be triggered. Depending on the origin of the performance drift, the optimal adaptation may be any combination of changes in replica placement, request routing or consistency policies.

5 Replica placement

The task of replica placement algorithms is to find good locations to host replicas. As noted earlier, replica placement forms a controllable input parameter of the objective function. Changes in uncontrollable parameters, such as client request rate or client latencies, may warrant changing the replica locations. In such case, the adaptation triggering component triggers the replica placement algorithms, which subsequently adapt the current placement to new conditions.

The problem of replica placement consists of two subproblems: *replica server placement* and *replica content placement*. Replica server placement is the problem of finding suitable locations for replica servers. Replica content placement is the problem of selecting replica servers that should host replicas of an object. Both these placements can be adjusted by the system to optimize the objective function value λ.

There are some fundamental differences between the server and content placement problems. Server placement concerns the selection of locations that are good for hosting replicas of *many* objects, whereas content placement deals with the selection of locations that are good for replicas of a *single* object. Furthermore, these two problems differ in how often and by whom their respective solutions need to be applied. The server placement algorithms are used in a larger time scale than the content placement algorithms. They are usually used by the system operator during installation of server infrastructure or while upgrading the hosting infrastructure, and runs once every few months. The content placement algorithms are run more often, as they need to react to possibly rapidly changing situations such as flash crowds.

We note that Karlsson et al. [2002] present a framework for evaluating replica placement algorithms for content delivery networks and also in other fields such as distributed file systems and databases. Their framework can be used to classify and qualitatively compare the performance of various algorithms using a generic set of primitives covering problem definition and heuristics. They also provide an analytical model to predict the decision times of each algorithm. Their framework is useful for evaluating the *relative performance* of different replica placement algorithms, and as such, complements the material discussed in this section.

5.1 Replica server placement

The problem of replica server placement is to select K servers out of N potential sites such that the objective function is optimized for a given network topology, client population, and access patterns. The objective function used by the server placement algorithms operates on some of the metrics defined in Section 3. These metrics may include, for example, client latencies for the objects hosted by the system, and the financial cost of server infrastructure.

The problem of determining the number and locations of replica servers, given a network topology, can be modeled as the center placement problem. Two variants used for modeling it are the facility location problem and the minimum K-median problem. Both these problems are NP-hard. They are defined in [Shmoys et al. 1997; Qiu et al. 2001], and we describe them here again for the sake of completeness.

Facility Location Problem Given a set of candidate server locations i in which the replica servers ("facilities") may be installed, running a server in location i incurs a cost of f_i. Each client j must be assigned to one replica server, incurring a cost of $d_j c_{ij}$ where d_j denotes the demand of the node j, and c_{ij} denotes the distance between i and j. The objective is to find the number and location of replica servers which minimizes the overall cost.

Minimum K-Median Problem Given N candidate server locations, we must select K of them (called "centers"), and then assign each client j to its closest center. A client j assigned to a center i incurs a cost of $d_j c_{ij}$. The goal is to select k centers, so that the overall cost is minimal.

The difference between the minimum K-median problem and the facility location problem is that the former associates no cost with opening a center (as with a facility, which has an operating cost of f_i). Further, in the minimum K-median problem, the number of servers is bounded by K.

Some initial work on the problem of replica server placement has been addressed in [da Cunha 1997]. However, it has otherwise been seldom addressed by the research community and only few solutions have been proposed.

Li et al. [1999] propose a placement algorithm based on the assumption that the underlying network topologies are trees and solve it using dynamic programming techniques. The algorithm is designed for Web proxy placement but is also relevant to server placement. The algorithm works by dividing a tree T into smaller subtrees T_i; the authors show that the best way to place t proxies is by placing t_i proxies for each T_i such that $\sum t_i = t$. The algorithm is shown to be optimal if the underlying network topology is a tree. However, this algorithm has the following limitations: (i) it cannot be applied to a wide-area network such as the Internet whose topology is not a tree, and (ii) it has a high computational complexity of $O(N^3K^2)$ where K is the number of proxies and N is the number of candidate locations. We note that the first limitation of this algorithm is due to its assumption about the presence of a single origin server and henceforth to find servers that can host a target Web service. This allows to construct only a tree topology with this origin server as root. However, a typical Web replica hosting system will host documents from multiple origins, falsifying this assumption. This nature of problem formulation is more relevant for content placement, where every document has a single origin Web server.

Qiu et al. [2001] model the replica placement problem as a minimum K-median problem and propose a greedy algorithm. In each iteration, the algorithm selects one server, which offers the least cost, where cost is defined as the average distance between the server and its clients. In the i^{th} iteration, the algorithm evaluates the cost of hosting a replica at the remaining $N - i + 1$ potential sites in the presence of already selected $i - 1$ servers. The computational cost of the algorithm is $O(N^2K)$. The authors also present a hot-spot algorithm, in which the replicas are placed close to the clients generating most requests. The computational complexity of the hot-spot algorithm is $N^2 + min(NlogN, NK)$. The authors evaluate the performance of these two algorithms and compare each one with the algorithm proposed in [Li et al. 1999]. Their analysis shows that the greedy algorithm performs better than the other two algorithms and its performance is only 1.1 to 1.5 times worse than the optimal solution. The authors note that the placement algorithms need to incorporate the client topology information and access pattern information, such as client end-to-end distances and request rates.

Radoslavov et al. [2001] propose two replica server placement algorithms that do not require the knowledge of client location but decide on replica location based on the network topology alone. The proposed algorithms are *max router fanout* and *max AS/max router fanout*. The first algorithm selects servers closest to the router having maximum fanout in the network. The second algorithm first selects the Autonomous System (AS) with the highest fanout, and then selects a server within that AS that is closest to the router having maximum fanout. The performance studies show that the second algorithm performs only 1.1 to 1.2 times worse than that of the greedy algorithm proposed in [Qiu et al. 2001]. Based on this, the authors argue that the need for knowledge of client locations is not essential. However, it must be noted that these topology-aware algorithms assume that the clients are uniformly spread throughout the network, which may not be true. If clients are not spread uniformly throughout the network, then the algorithm can select replica servers that are close to routers with highest fanout but distant from most of the clients, resulting in poor client-perceived performance.

5.2 Replica content placement

The problem of replica content placement consists of two sub-problems: *content placement* and *replica creation*. The first problem concerns the selection of a set of replica servers that must hold the replica of a given object. The second problem concerns the selection of a mechanism to inform a replica server about the creation of new replicas.

5.2.1 Content placement

The content placement problem consists of selecting K out of N replica servers to host replicas of an object, such that the objective function is optimized under a given client access pattern and replica update pattern. The content placement algorithms select replica servers in an effort to improve the quality of service provided to the clients and minimize the object hosting cost.

Similarly to the server placement, the content placement problem can be modeled as the facility location placement. However, such solutions can be computationally expensive making it difficult to be applied to this problem, as the content placement algorithms are run far more often their server-related counterparts. Therefore, existing replica hosting systems exploit simpler solutions.

In Radar [Rabinovich and Aggarwal 1999], every host runs the replica placement algorithm, which defines two client request rate thresholds: R_{rep} for replica replication, and R_{del} for object deletion, where $R_{del} < R_{rep}$. A document is deleted if its client request rate drops below R_{del}. The document is replicated if its client request rate exceeds R_{rep}. For request rates falling between R_{del} and R_{rep}, documents are migrated to a replica server located closer to clients that issue more than a half of requests. The distance is calculated using a Radar-specific metric called preference paths. These preference paths are computed by the servers based on information periodically extracted from routers.

In SPREAD, the replica servers periodically calculate the expected number of requests for an object. Servers decide to create a local replica if the number of requests exceeds a certain threshold [Rodriguez and Sibal 2000]. These servers remove a replica if the popularity of the object decreases. If required, the total number of replicas of an object can be restricted by the object owner.

Chen et al. [2002] propose a dynamic replica placement algorithm for scalable content delivery. This algorithm uses a *dissemination-tree*-based infrastructure for content delivery and a peer-to-peer location service provided by Tapestry for locating objects [Zhao et al. 2004]. The algorithm works as follows. It first organizes the replica servers holding replicas of the same object into a load-balanced tree. Then, it starts receiving client requests which target the origin server containing some latency constraints. The origin server services the client, if the server's capacity constraints and client's latency constraints are met. If any of these conditions fail, it searches for another server in the dissemination tree that satisfies these two constraints and creates a replica at that server. The algorithm aims to achieve better scalability by quickly locating the objects using the peer-to-peer location service. The algorithm is good in terms of preserving client latency and server capacity constraints. On the other hand, it has a considerable overhead caused by checking QoS requirements for every client request. In the worst case a single client request may result in creating a new replica. This can significantly increase the request servicing time.

Kangasharju et al. [2001] model the content placement problem as an optimization problem. The problem is to place K objects in some of N servers, in an effort to minimize the average number of inter-AS hops a request must traverse to be serviced, meeting the storage constraints of each server. The problem is shown to be NP-complete and three heuristics are proposed to address this problem. The first heuristic uses popularity of an object as the only criterion and every server decides upon the objects it needs to host based on the objects' popularity. The second heuristic uses a cost function defined as a product of object popularity and distance of server from origin server. In this heuristic, each server selects the objects to host as the ones with high cost. The intuition behind this heuristic is that each server hosts objects that are highly popular and also that are far away from their origin server, in an effort to minimize the client latency. This heuristic always tries to minimize the distance of a replica from its origin server oblivious of the presence of other replicas. The third heuristic overcomes this limitation and uses a coordinated replication strategy where replica locations are decided in a global/coordinated fashion for all objects. This heuristic uses a cost function that is a product of total request rate for a server, popularity and shortest distance of a server to a copy of object. The central server selects the object and replica pairs that yield the best cost at every iteration and recomputes the shortest distance between servers for each object. Using simulations, the authors show that the global heuristic outperforms the other two heuristics. The drawback is its high computational complexity.

5.2.2 Replica creation mechanisms

Various mechanisms can be used to inform a replica server about the creation of a new replica that it needs to host. The most widely used mechanisms for this purpose are *pull-based caching* and *push replication*.

In pull-based caching, replica servers are not explicitly informed of the creation of a new replica. When a replica server receives a request for a document it does not own, it treats it as a miss and fetches the replica from the master. As a consequence, the creation of a new replica is delayed until the first request for this replica. This scheme is adopted in Akamai [Dilley et al. 2002]. Note that in this case, pull-based caching is used only as a mechanism for replica creation. The decision to place a replica in that server is taken by the system, when redirecting client requests to replica servers.

In push replication, a replica server is informed of a replica creation by explicitly pushing the replica contents to the server. Similar scheme is used in Globule [Pierre and van Steen 2001] and Radar [Rabinovich and Aggarwal 1999].

5.3 Discussion

The problem of replica server and content placement is not regularly addressed by the research community. A few recent works have proposed solution for these problems [Qiu et al. 2001; Radoslavov et al. 2001; Chen et al. 2002; Kangasharju et al. 2001]. We note that an explicit distinction between server and content placement is generally not made. Rather, work has concentrated on finding server locations to host contents of a single content provider. However, separate solutions for server placement and content placement would be more useful in a replica hosting system, as these systems are intended to host different contents with varying client access patterns.

Choosing the best performing content placement algorithm is not trivial as it depends on the access characteristics of the Web content.Pierre and van Steen [2001] showed there is no single best performing replica placement strategy and it must be selected on a per-document basis based on their individual access patterns. Karlsson and Karamanolis [2004] propose a scheme where different placement heuristics are evaluated off-line and the best performing heuristic is selected on a per-document basis. In [Pierre and van Steen 2001; Sivasubramanian et al. 2003], the authors propose to perform this heuristic selection dynamically where the system adapts to change in access patterns by switching the documents' replication strategies on-the-fly.

Furthermore, the existing server placement algorithms improve client QoS by minimizing client latency or distance [Qiu et al. 2001; Radoslavov et al. 2001]. Even though client QoS is important to make placement decisions, in practice the selection of replica servers is constrained by administrative reasons, such as business relationship with an ISP, and financial cost for installing a replica server. Such a situation introduces a necessary tradeoff between financial cost and performance gain, which are not directly comparable entities. This drives the need for server placement solutions that not only takes into account the financial cost of a particular server facility but that can also translate the performance gains into potential monetary benefits. To the best of our knowledge little work has been done in this area, which requires building economic models that translate the performance of replica hosting system into the monetary profit gained. These kinds of economic models are imperative to enable system designers to make better judgments in server placement and provide server placement solutions that can be applied in practice.

6 Consistency enforcement

The consistency enforcement problem concerns selecting *consistency models* and implementing them using various *consistency policies*, which in turn can use several *content distribution mechanisms*. A consistency model is a contract between a replica hosting system and its clients that dictates the consistency-related properties of the content delivered by the system. A consistency policy defines how, when, and to which object replicas the various content distribution mechanisms are applied. For each object, a policy adheres to a certain consistency model defined for that object. A single model can be implemented using different policies. A content distribution mechanism is a method by which replica servers exchange replica updates. It defines in what form replica updates are transferred, who initiates the transfer, and when updates take place.

Although consistency models and mechanisms are usually well defined, choosing a valid one for a given object is a non-trivial task. The selection of a consistency model, policies, and mechanisms must ensure that the required level of consistency (defined by various consistency metrics discussed in section 3) is met, while keeping the communication overhead to be as low as possible.

6.1 Consistency models

Consistency models differ in their strictness of enforcing consistency. By strong consistency, we mean that the system guarantees that all replicas are identical from the perspective of the system's clients. If a given replica is not consistent with others, it cannot be accessed by clients until it is brought up to date. Due to high replica synchronization costs, strong consistency is seldom used in wide-area systems. Weak consistency, in turn, allows replicas to differ, but ensures that all updates reach all replicas after some (bound) time. Since this model is resistant to delays in update propagation and incurs less synchronization overhead, it fits better in wide-area systems.

6.1.1 Single vs. multiple master

Depending on whether updates originate from a single site or from several ones, consistency models can be classified as single-master or multi-master, respectively. The single-master models define one machine to be responsible for holding an up-to-date version of a given object. These models are simple and fit well in applications where the objects by nature have a single source of changes. They are also commonly used in existing replica hosting systems, as these systems usually deliver some centrally managed data. For example, Radar assumes that most of its objects are static Web objects that are modified rarely and uses primary-copy mechanisms for enforcing consistency. The multi-master models allow more than one server to modify the state of an object. These models are applicable to replicated Web objects whose state can be modified as a result of a client access. However, these models introduce new problems such as the necessity of solving update conflicts. Little work has been done on multi-master models in the context of Web replica hosting systems.

6.1.2 Types of consistency

As we explained, consistency models usually define consistency along three different axes: time, value, and order.

Time-based consistency models were formalized in [Torres-Rojas et al. 1999] and define consistency based on real time. These models require a content distribution mechanism to ensure that an update to a replica at time t is visible to the other replicas and clients before time $t+\Delta$. Cate [1992] adopts a time-based consistency model for maintaining consistency of FTP caches. The consistency policy in this system guarantees that the only updates that might not yet be reflected in a site are the ones that have happened in the last 10% of the reported age of the file. Time-based consistency is applicable to all kinds of objects. It can be enforced using different content distribution mechanisms such as *polling* (where a client or replica polls often to see if there is an update), or server invalidation (where a server invalidates a copy held by other replicas and clients if it gets updated). These mechanisms are explained in detail in the next section.

Value-based consistency schemes ensure that the difference between the value of a replica and that of other replicas (and its clients) is no greater than a certain Δ. Value-based schemes can be applied only to objects that have a precise definition of value. For example, an object encompassing the details about the number of seats booked in an aircraft can use such a model. This scheme can be implemented by using polling or server invalidation mechanisms. Examples of value-based consistency schemes and content distribution mechanisms can be found in [Bhide et al. 2002].

Order-based consistency schemes are generally exploited in replicated databases. These models perceive every read/write operation as a transaction and allow the replicas to operate in different state if the out-of-order transactions adhere to the rules defined by these policies. For example, Krishnakumar and Bernstein [1994] introduce the concept of N-ignorant transactions, where a transaction can be carried out in a replica while it is ignorant of N prior transactions in other replicas. The rules constraining the order of execution of transactions can also be defined based on dependencies among transactions. Implementing order-based consistency policies requires content distribution mechanisms to exchange the transactions among all replicas, and transactions need to be timestamped using mechanisms such as logical clocks [Raynal and Singhal 1996]. This consistency scheme is applicable to a group of objects that jointly constitute a regularly updated database.

A continuous consistency model, integrating the above three schemes, is presented by Yu and Vahdat [2002]. The underlying premise of this model is that there is a continuum between strong and weak consistency models that is semantically meaningful for a wide range of replicated services, as opposed to traditional consistency models, which explore either strong or weak consistency [Bernstein and Goodman 1983]. The authors explore the space between these two extremes by making applications specify their desired level of consistency using *conits*. A conit is defined as a physical or logical unit of consistency. The model uses a three-dimensional vector to quantify consistency: *(numerical error, order error, staleness)*. *Numerical error* is used to define and implement value-based consistency, *order error* is used to define and implement order-based consistency schemes, and *staleness* is used for time-based consistency. If each of these metrics is bound to zero, then the model implements strong consistency. Similarly, if there are no bounds then the model does not provide any consistency at all. The conit-based model allows a broad range of applications to express their consistency requirements. Also, it can precisely describe guarantees or bounds with respect to differences between replicas on a per-replica basis. This enables replicas having poor network connectivity to implement relaxed consistency, whereas replicas with better connectivity can still benefit from stronger consistency. The mechanisms implementing this conit-based model are described in [Yu and Vahdat 2000] and [Yu and Vahdat 2002].

6.2 Content distribution mechanisms

Content distribution mechanisms define how replica servers exchange updates. These mechanisms differ on two aspects: the forms of the update and the direction in which updates are triggered (from source of update to other replicas or vice versa). The decision about these two aspects influences the system's attainable level of consistency as well as the communication overhead introduced to maintain consistency.

6.2.1 Update forms

Replica updates can be transferred in three different forms. In the first form, called *state shipping*, a whole replica is sent. The advantage of this approach is its simplicity. On the other hand, it may incur significant communication overhead, especially noticeable when a small update is performed on a large object.

In the second update form, called *delta shipping*, only differences with the previous state are transmitted. It generally incurs less communication overhead compared to state shipping, but it requires each replica server to have the previous replica version available. Furthermore, delta shipping assumes that the differences between two object versions can be quickly computed.

In the third update form, called *function shipping*, replica servers exchange the actions that cause the changes. It generally incurs the least communication overhead as the size of description of the action is usually independent from the object state and size. However, it forces each replica server to convey a certain, possibly computationally demanding, operation.

The update form is usually dictated by the exploited replication scheme and the object characteristics. For example, in *active replication* requests targeting an object are processed by all the replicas of this object. In such a case, function shipping is the only choice. In *passive replication*, in turn, requests are first processed by a single replica, and then the remaining ones are brought up-to-date. In such a case, the update form selection depends on the object characteristics and the change itself: whether the object structure allows for changes to be easily expressed as an operation (which suggests function shipping), whether the object size is large compared to the size of the changed part (which suggests delta shipping), and finally, whether the object was simply replaced with a completely new version (which suggests state shipping).

In general, it is the job of a system designer to select the update form that minimizes the overall communication overhead. In most replica hosting systems, updating means simply replacing the whole replica with its new version. However, it has been shown that updating Web objects using delta shipping could reduce the communication overhead by up to 22% compared to commonly used state shipping [Mogul et al. 1997].

6.2.2 Update direction

The update transfer can be initiated either by a replica server that is in need for a new version and wants to *pull* it from one of its peers, or by the replica server that holds a new replica version and wants to *push* it to its peers. It is also possible to combine both mechanisms to achieve a better result.

Pull In one version of the pull-based approach, every piece of data is associated with a *Time To Refresh* (TTR) attribute, which denotes the next time the data should be validated. The value of TTR can be a constant, or can be calculated from the update rate of the data. It may also depend on the consistency requirements of the system. Data with high update rates and strong consistency requirements require a small TTR, whereas data with less updates can have a large TTR. Such a mechanism is used in [Cate 1992]. The advantage of the pull-based scheme is that it does not require replica servers to store state information, offering the benefit of higher fault tolerance. On the other hand, enforcing stricter consistency depends on careful estimation of TTR: small TTR values will provide good consistency, but at the cost of unnecessary transfers when the document was not updated.

In another pull-based approach, HTTP requests targeting an object are extended with the HTTP *if-modified-since* field. This field contains the modification date of a cached copy of the object. Upon receiving such a request, a Web server compares this date with the modification date of the original object. If the Web server holds a newer version, the entire object is sent as the response. Otherwise, only a header is sent, notifying that the cached copy is still valid. This approach allows for implementing strong consistency. On the other hand, it can impose large communication overhead, as the object home server has to be contacted for each request, even if the cached copy is valid.

In practice, a combination of TTR and checking the validity of a document at the server is used. Only after the TTR value expires, will the server contact the document's origin server to see whether the cached copy is still validate. If it is still valid, a fresh TTR value is assigned to it and a next validation check is postponed until the TTR value expires again.

Push The push-based scheme ensures that communication occurs only when there is an update. The key advantage of this approach is that it can meet strong consistency requirements without introducing the communication overhead known from the "if-modified-since" approach: since the replica server that initiates the update transfer is aware of changes, it can precisely determine which changes to push and when. An important constraint of the push-based scheme is that the object home server needs to keep track of all replica servers to be informed. Although storing this list may seem costly, it has been shown that it can be done in an efficient way [Cao and Liu 1998]. A more important problem is that the replica holding the state becomes a potential single point of failure, as the failure of this replica affects the consistency of the system until it is fully recovered.

Push-based content distribution schemes can be associated with leases [Gray and Cheriton 1989]. In such approaches, a replica server registers its interest in a particular object for an associated lease time. The replica server remains registered at the object home server until the lease time expires. During the lease time, the object home server pushes all updates of the object to the replica server. When the lease expires, the replica server can either consider it as potentially stale or register at the object home server again.

Leases can be divided into three groups: age-based, renewal-frequency-based, and load-based ones [Duvvuri et al. 2000]. In the age-based leases, the lease time depends on the last time the object was modified. The underlying assumption is that objects that have not been modified for a long time will remain unmodified for some time to come. In the renewal-frequency-based leases, the object home server gives longer leases to replica servers that ask for replica validation more often. In this way, the object server prefers replica servers used by clients expressing more interest in the object. Finally, with load-based leases the object home server tends to give away shorter lease times when it becomes overloaded. By doing that, the object home server reduces the number of replica servers to which the object updates have to be pushed, which is expected to reduce the size of the state held at the object home server.

Other schemes The pull and push approaches can be combined in different ways. Bhide et al. [2002] propose three different combination schemes of Push and Pull. The first scheme, called *Push-and-Pull (PaP)*, simultaneously employs push and pull to exchange updates and has tunable parameters to control the extent of push and pulls. The second scheme, *Push-or-Pull (PoP)*, allows a server to adaptively choose between a push- or pull-based approach for each connection. This scheme allows a server to characterize which clients (other replica servers or proxies to which updates need to be propagated) should use either of these two approaches. The characterization can be based on system dynamics. By default, clients are forced to use the pull-based approach. PoP is a more effective solution than PaP, as the server can determine the moment of switching between push and pull, depending on its resource availability. The third scheme, called *PoPoPaP*, is an extended version of PoP, that chooses from Push, Pull and PaP. PoPoPaP improves the resilience of the server (compared to PoP), offers graceful degradation, and can maintain strong consistency.

Another way of combining push and pull is to allow the former to trigger the latter. It can be done either explicitly, by means of *invalidations*, or implicitly, with *versioning*. Invalidations are pushed by an object's origin server to a replica server. They inform the replica server or the clients that the replica it holds is outdated. In case the replica server needs the current version, it pulls it from the origin server. Invalidations may reduce the network overhead, compared to pushing regular updates, as the replica servers do not have to hold the current version for all the time and can delay its retrieval until it is really needed. It is particularly useful for often-updated, rarely-accessed objects.

Versioning techniques are exploited in Akamai [Dilley et al. 2002; Leighton and Lewin 2000]. In this approach, every object is assigned a version number, increased after each update. The parent document that contains a reference to the object is rewritten after each update as well, so that it points to the latest version. The consistency problem is thus reduced to maintaining the consistency of the parent document. Each time a client retrieves the document, the object reference is followed and a replica server is queried for that object. If the replica server notices that it does not have a copy of the referenced version, the new version is pulled in from the origin server.

Scalable mechanisms All the aforesaid content distribution mechanisms do not scale for large number of replicas (say, in the order of thousands). In this case, push-based mechanisms suffer from the overhead of storing the state of each replica and updating them (through separate unicast connections). Pull-based mechanisms suffer from the disadvantage of creating a hot spot around the origin server with thousands of replicas requesting the origin server (again through separate connections) for an update periodically. Both mechanisms suffer from excessive network traffic for updating large number of replicas, as the same updates are sent to different replicas using separate connections. This also introduces considerable overhead on the server, in addition to increasing the network overhead. These scalability limitations require the need for building scalable mechanisms.

Scalable content distribution mechanisms proposed in the literature aim to solve scalability problems of conventional push and pull mechanisms by building a content distribution hierarchy of replicas or clustering objects.

The first approach is adopted in [Ninan et al. 2002; Tewari et al. 2002; Fei 2001]. In this approach, a content distribution tree of replicas is built for each object. The origin server sends its update only to the root of the tree (instead of the entire set of replicas), which in turn forwards the update to the next level of nodes in the tree and so on. The content distribution tree can be built either using network multicasting or application-level multicasting solutions. This approach drastically reduces the overall amount of data shipped by the origin server. In [Ninan et al. 2002], the authors proposed a scalable lease-based consistency mechanism where leases are made with a replica group (with the same consistency requirement), instead of individual replicas. Each lease group has its own content distribution hierarchy to send their replica updates. Similarly Tewari et al. [2002] propose a mechanism that builds a content distribution hierarchy and also uses object clustering to improve the scalability.

Fei [2001] propose a mechanism that chooses between update propagation (through a multicast tree) or invalidation schemes on a per-object basis, periodically, based on each object's update and access rate. The basic intuition of the mechanism is to choose propagation if an object is accessed more than it is updated (thereby reducing the pull traffic) and invalidation otherwise (as the overhead for shipping updates is higher than pulling updates of an object only when it is accessed). The mechanism computes the traffic overhead of the two methods for maintaining consistency of an object, given its past update and access rate. It chooses the one that introduces the least overhead as the mechanism to be adopted for that object.

Object clustering is the process of clustering various objects with similar properties (update and/or request patterns) and treating them as a single clustered object. It reduces the connection initiation overhead during the transmission of replica updates, from a per-object level to per-cluster level, as updates for a cluster are sent in a single connection instead of individuals connection for each object (note the amount of updates transferred using both mechanisms are the same). Clustering also reduces the number of objects to be maintained by a server, which can help in reducing the adaptation overhead as a single decision will affect more objects. To our knowledge, object clustering is not used in any well-known replica hosting system.

6.3 Mechanisms for dynamically generated objects

In recent years, there has been a sharp increase in providing Web content using technologies like ASPs or CGIs that dynamically generate Web objects, possibly from an underlying database. These dynamically generated objects affect Web server performance as their generation may require many CPU cycles [Iyengar et al. 1997].

Several systems cache the generated pages [Challenger et al. 1999; Labrinidis and Roussopoulos 2000]. Assuming that the temporal locality of requests is high and updates to the underlying data are infrequent, these systems avoid generating the same document multiple times. Challenger et al. [1999] additionally proposes to maintain a dependency graph between cached pages and the underlying data that are used to generate them. Upon data updates, all pages depending on the updated data are invalidated.

Unfortunately, many applications receive large numbers of unique requests or must handle frequent data updates. Such applications can be distributed only through replication, where the application code is executed at the replica servers. This avoids the wide-area network latency for each request and ensures quicker response time to clients. Replicating a dynamic Web object requires replicating both the code (e.g., EJBs, CGI scripts, PHPs) and the data that the code acts upon (databases or files). This can reduce the latency of requests, as the requests can be answered by the application hosted by the server located close to the clients.

Replicating applications is relatively easy provided that the code does not modify the data [Cao et al. 1998; Rabinovich et al. 2003]. However, most Web applications do modify their underlying data. In this case, it becomes necessary to manage data consistency across all replicas. To our knowledge, there are very few existing systems that handle consistency among the replicated data for dynamic Web objects. Gao et al. [2003] propose an application-specific edge service architecture, where the object is assumed to take care of its own replication and consistency. In such a system, access to the shared data is abstracted by object interfaces. This system aims to achieve scalability by using weaker consistency models tailored to the application. However, this requires the application developer to be

Table 3: A comparison of approaches for enforcing consistency.

Systems and Protocols	Push		Pull	Vari-ants	Comments
	Inv.	Prop.			
Akamai	X		X		Uses push-based invalidation for consistency and pull for distribution
Radar		X			Uses primary-copy
SPREAD				X	Chooses strategy on a per-object basis based on its access and update rate
[Pierre et al. 2002]				X	Chooses strategy on a per-object basis based on its access and update rate
[Duvvuri et al. 2000]	X				Invalidates content until lease is valid
Adaptive Push-Pull		X	X		Chooses between push and pull strategy on a per-object basis
[Fei 2001]	X	X			Chooses between propagation and invalidation on a per-object basis

aware of the application's consistency and distribution semantics.

Further research on these topics would be very beneficial to replica hosting systems, as the popularity of dynamic documents technique is increasing.

6.4 Discussion

In this section, we discussed two important components of consistency enforcement namely, consistency models and content distribution mechanisms. In consistency models, we listed different types of consistency models – based on time, value or transaction orders. In addition to these models, we also discussed the continuous consistency model, in which different network applications can express the consistency constraints in any point in the consistency spectrum. This model is useful to capture the consistency requirements for a broad range of applications being hosted by a replica hosting system. However, the mechanisms proposed to enforce its policies do not scale with increasing number of replicas. Similar models need to be developed for Web replica hosting systems that can provide bounds on inconsistent access of its replicas with no loss of scalability.

In content distribution mechanisms, we discussed the advantages and disadvantages of push, pull, and other adaptive mechanisms. These mechanisms can be broadly classified as server-driven and client-driven consistency mechanisms, depending on who is responsible for enforcing consistency. At the outset, client-driven mechanisms seems to be a more scalable option for large-scale hosting systems, as in this case the server is not overloaded with the responsibility of enforcing consistency. However, in [Yin et al. 2002] the authors have shown that server-driven consistency protocols can meet the scalability requirements of large-scale dynamic Web services delivering both static and dynamic Web content.

We note that existing systems and protocols concentrate only on time-based consistency models and very little has been done on other consistency models. Hence, in our summary table of consistency approaches adopted by various systems and protocols (Table 3), we discuss only the content distribution mechanisms adopted by them.

Maintaining consistency among dynamic documents requires special mechanisms. Existing replication solutions usually incur high overhead because they need to global synchronization upon updates of their underlying replicated data. We consider that optimistic replication mechanisms such as those used in [Gao et al. 2003] may allow massively scalable content distribution mechanisms for dynamically generated documents. An in-depth survey on optimistic replication techniques can be found in [Saito and Shapiro 2003].

7 Request routing

In request routing, we address the problem of deciding which replica server shall best service a given client request, in terms of the metrics selected in Section 3. These metrics can be, for example, replica server load (where we choose the replica server with the lowest load), end-to-end latency (where we choose the replica server that offers the shortest response time to the client), or distance (where we choose the replica server that is closest to the client).

Selecting a replica is difficult, because the conditions on the replica servers (e.g., load) and in the network (e.g., link congestion, thus its latency) change continuously. These changing conditions may lead to different replica selections, depending on when and for which client these selections are made. In other words, a replica optimal for a given client may not necessarily remain optimal for the same client forever. Similarly, even if two clients request the same document simultaneously, they may be directed to different replicas. In this section, we refer to these two kinds of conditions as "system conditions."

The entire request routing problem can be split into two: devising a redirection policy and selecting a redirection mechanism. A redirection policy defines how to select a replica in response to a given client request. It is basically an algorithm invoked when the client request is invoked. A redirection mechanism, in turn, is a mean of informing the client about this selection. It first invokes a redirection policy, and then provides the client with the redirecting response that the policy generates.

A redirection system can be deployed either on the client side, or on the server side, or somewhere in the network between these two. It is also possible to combine client-side and server-side techniques to achieve better performance [Karaul et al. 1998]. Interestingly, a study by Rodriguez et al. [2000] suggests that clients may easily circumvent the problem of replica selection by simultaneously retrieving their data from several replica servers. This claim is disputed by Kangasharju et al. [2001], who notice that the delay caused by opening connections to multiple servers can outweigh the actual gain in content download time. In this paper, we assume that we leave the client-side unmodified, as the only software that usually works there is a Web browser. We therefore do not discuss the details of client-side server-selection techniques, which can be found in [Conti et al. 2002]. Finally, we do not discuss various Web caching schemes, which have been thoroughly described in [Rodriguez et al. 2001], as caches are by nature deployed on the client-side.

In this section, we examine redirection policies and redirection mechanisms separately. For each of them, we discuss several related research efforts, and summarize with a comparison of these efforts.

7.1 Redirection policies

A redirection policy can be either adaptive or non-adaptive. The former considers current system conditions while selecting a replica, whereas the latter does not. Adaptive redirection policies are usually more complex than non-adaptive ones, but this effort is likely to pay off with higher system performance. The systems we discuss below usually implement both types of policies, and can be configured to use any combination of them.

7.1.1 Non-adaptive policies

Non-adaptive redirection policies select a replica that a client should access without monitoring the current system conditions. Instead, they exploit heuristics that assume certain properties of these conditions of which we discuss examples below. Although non-adaptive policies are usually easier to implement, the system works efficiently only when the assumptions made by the heuristics are met.

An example of a non-adaptive policy is round-robin. It aims at balancing the load of replica servers by evenly distributing all the requests among these servers [Delgadillo 1999; Radware 2002; Szymaniak et al. 2003]. The assumption here is that all the replica servers have similar processing capabilities, and that any of them can service any client request. This simple policy has proved to work well in clusters, where all the replica servers are located in the same place [Pai et al. 1998]. In wide-area systems, however, replica servers are usually distant from each other. Since round-robin ignores this aspect, it cannot prevent directing client requests to more distant replica servers. If it happens, the client-perceived performance may turn out to be poor. Another problem is that the aim of load balancing itself is not necessarily achieved, as processing different requests can involve significantly different computational costs.

A non-adaptive policy exploited in Radar is the following. All replica servers are ranked according to their predicted load, which is derived from the number of requests each of them has serviced so far. Then, the policy redirects clients so that the load is balanced across the replica servers, and that (additionally) the client-server distance is as low as possible. The assumption here is that the replica server load and the client-server distance are the main factors influencing the efficiency of request processing. Aggarwal and Rabinovich [1998] observe that this simple policy often performs nearly as good as its adaptive counterpart, which we describe below. However, as both of them ignore network congestion, the resulting client-perceived performance may still turn out to be poor.

Several interesting non-adaptive policies were implemented in Cisco DistributedDirector [Delgadillo 1999]. The first one defines the percentage of all requests that each replica server receives. In this way, it can send more requests to more powerful replica servers and achieve better resource utilization. Another policy allows for defining preferences of one replica server over the other. It may be used to temporarily relieve a replica server from service (for maintenance purposes, for example), and delegate the requests it would normally service to another server. Finally, DistributedDirector enables random request redirection, which can be used for comparisons during some system efficiency tests. Although all these policies are easy to implement, they completely ignore current system conditions, making them inadequate to react to emergency situations.

One can imagine a non-adaptive redirection policy that statically assigns clients to replicas based on their geographical location. The underlying assumptions are that the clients are evenly distributed over the world, and that the geographical distance to a server reflects the network latency to that server. Although the former assumption is not likely to be valid in a general case, the latter has been verified positively as we discussed earlier. According to Huffaker et al. [2002], the correlation between the geographical distance and the network latency reaches up to 75%. Still, since this policy ignores the load of replica servers, it can redirect clients to overloaded replica servers, which may lead to substantially degraded client experience.

Finally, an interesting non-adaptive policy that has later been used in developing the Chord peer-to-peer system, is consistent hashing [Karger et al. 1999]. The idea is straightforward: a URL is hashed to a value h from a large space of identifiers. That value is then used to efficiently route a request along a logical ring consisting of cache servers with IDs from that same space. The cache server with the smallest ID larger than h is responsible for holding copies of the referenced data. Variations of this scheme have been extensively researched in the context of peer-to-peer file sharing systems [Balikrishnan et al. 2003], including those that take network proximity into account (see, e.g., [Castro et al. 2003]).

7.1.2 Adaptive policies

Adaptive redirection policies discover the current system conditions by means of metric estimation mechanisms discussed in Section 3. In this way, they are able to adjust their behavior to situations that normally do not occur, like flash crowds, and ensure high system robustness [Wang et al. 2002].

The information that adaptive policies obtain from metric estimation mechanisms may include, for example, the load of replica servers or the congestion of selected network links. Apart from these data, a policy may also need to know some request-related information. The bare minimum is what object is requested and where the client is located. More advanced replica selection can also take client QoS requirements into account.

Knowing the system conditions and the client-related information, adaptive policies first determine a set of replica servers that are capable of handling the request (i.e., they store a replica of the document and can offer required quality of service). Then, these policies select one (or more) of these servers, according to the metrics they exploit. Adaptive policies may exploit more than one metric. More importantly, a selection based on one metric is not necessarily optimal in terms of others. For example, Johnson et al. [2001] observed that most CDNs do not always select the replica server closest to the client.

The adaptive policy used by Globule selects the replica servers that are closest to the client in terms of network distance [Szymaniak et al. 2003]. Globule employs the AS-path length metric, originally proposed by [McManus 1999], and determines the distance based on a periodically-refreshed, AS-based map of the Internet. Since this approach uses passive metric estimation services, it does not introduce any additional traffic to the network. We consider it to be adaptive, because the map of the Internet is periodically rebuilt, which results in (slow) adaptation to network topology changes. Unfortunately, the AS-based distance calculations, although simple to perform, are not very accurate [Huffaker et al. 2002].

A distance-based adaptive policy is also implicitly exploited by SPREAD [Rodriguez and Sibal 2000]. In this system, routers simply intercept requests on their path toward the object home server, and redirected to a near-by replica server. Consequently, requests reach their closest replica servers, and the resulting client-server paths are shortened. This policy in a natural way adapts to changes in routing. Its biggest disadvantage is the high cost of deployment, as it requires modifying many routers.

A combined policy, considering both replica server load and client-server distance, is implemented in Radar. The policy first isolates the replica servers whose load is below a certain threshold. Then, from these servers, the client-closest one is selected. The Radar redirection policy adapts to changing replica server loads and tries to direct clients to their closest replica servers. However, by ignoring network congestion and end-to-end latencies, Radar focuses more on load balancing than on improving the client-perceived performance.

Adaptive policies based on the client-server latency have been proposed by Ardaiz et al. [2001] and Andrews et al. [2002]. Based either on the client access logs, or on passive server-side latency measurements, respectively, these policies redirect a client to the replica server that has recently reported the minimal latency to the client. The most important advantage of these schemes is that they exploit latency measurements, which are the best indicator of actual client experience [Huffaker et al. 2002]. On the other hand, both of them require maintaining a central database of measurements, which limits the scalability of systems that exploit these schemes.

A set of adaptive policies is supported by Web Server Director [Radware 2002]. It monitors the number of clients and the amount of network traffic serviced by each replica server. It also takes advantage of performance metrics specific for Windows NT, which are included in the Management Information Base (MIB). Since this information is only provided in a commercial white paper, it is difficult to evaluate the efficiency of these solutions.

Another set of adaptive policies is implemented in Cisco DistributedDirector [Delgadillo 1999]. This system supports many different metrics, including inter-AS distance, intra-AS distance, and end-to-end latency. The redirection policy can determine the replica server based on weighted combination of these three metrics. Although this policy is clearly more flexible than a policy that uses only one metric, measuring all the metrics requires deploying an "agent" on every replica server. Also, the exploited active latency measurements introduce additional traffic to the Internet. Finally, because DistributedDirector is kept separate from the replica servers, it cannot probe their load – it can be approximated only with the non-adaptive policies discussed above.

A complex adaptive policy is used in Akamai [Dilley et al. 2002]. It considers a few additional metrics, like replica server load, the reliability of routes between the client and each of the replica servers, and the bandwidth that is currently available to a replica server. Unfortunately, the actual policy is subject to trade secret and cannot be found in the published literature.

7.2 Redirection mechanisms

Redirection mechanisms provide clients with the information generated by the redirection policies. Redirection mechanisms can be classified according to several criteria. For example, Barbir et al. [2002], classify redirection mechanisms into DNS-based, transport-level, and application-level ones. The authors use the term "request routing" to refer to what we call "redirection" in this paper. Such classification is dictated by the diversity of request processing stages, where redirection can be incorporated: name resolution, packet routing, and application-specific redirection implementation.

In this section, we distinguish between transparent, non-transparent, and combined mechanisms. Transparent redirection mechanisms hide the redirection from the clients. In other words, a client cannot determine which replica server is servicing it. In non-transparent redirection mechanisms, the redirection is visible to the client, which can then explicitly refer to the replica server it is using. Combined redirection mechanisms combine two previous types. They take the best from these two types and eliminate their disadvantages.

As we only focus on wide-area systems, we do not discuss solutions that are applicable only to local environments. An example of such a solution is packet hand-off, which is thoroughly discussed in a survey of load-balancing techniques by Cardellini et al. [1999]. A more recent survey by the same authors covers other techniques for local-area Web clusters [Cardellini et al. 2002].

7.2.1 Transparent mechanisms

Transparent redirection mechanisms perform client request redirection in a transparent manner. Therefore, they do not introduce explicit bounds between clients and replica servers, even if the clients store references to replicas. It is particularly important for mobile clients and for dynamically changing network environments, as in both these cases, a replica server now optimal for a given client can become suboptimal shortly later.

Several transparent redirection mechanisms are based on DNS [Delgadillo 1999; Cardellini et al. 2003; Rabinovich and Aggarwal 1999; Radware 2002; Szymaniak et al. 2003]. They exploit specially modified DNS servers. When a modified DNS server receives a resolution query for a replicated service, a redirection policy is invoked to generate one or more service IP addresses, which are returned to the client. The policy chooses the replica servers based on the IP address of the query sender. In DNS-based redirection, transparency is achieved assuming that services are referred to by means of their DNS names, and not their IP addresses. The entire redirection mechanism is extremely popular, because of its simplicity and independence from the actual replicated service – as it is incorporated in the name resolution service, it can be used by any Internet application.

On the other hand, DNS-based redirection has some limitations [Shaikh et al. 2001]. The most important ones are poor client identification and coarse redirection granularity. The poor client identification is caused by the fact that a DNS query does not carry the addresses of the querying client. The query can pass through several DNS servers before it reaches the one that knows the answer. However, any of these DNS servers knows only the DNS server with which it directly communicates, and not the querying client. Consequently, using the DNS-based redirection mechanisms forces the system to use the clustering scheme based on local DNS servers, which was discussed in Section 3. The coarse redirection granularity is caused by the granularity of DNS itself: as it deals only with machine names, it can redirect based only on that part of an object URL that is related to the machine name. Therefore, as long as two URLs refer to the same machine name, they are identical for the DNS-based redirection mechanism, which makes it difficult to use different distribution schemes for different objects.

A scalable version of the DNS-based redirection is implemented in Akamai. This system improves the scalability of the redirection mechanism by maintaining two groups of DNS servers: top- and low-level ones. Whereas the former share one location, the latter are scattered over several Internet data centers, and are usually accompanied by replica servers. A top-level DNS servers redirects a client query to a low-level DNS server proximal to the query sender. Then, the low-level DNS server redirects the sender to an actual replica server, usually placed in the same Internet data center. What is important, however, is that the top-to-low level redirection occurs only periodically (about once per hour) and remains valid during all that time. For this reason, the queries are usually handled by proximal low-level DNS servers, which results in short name-resolution latency. Also, because the low-level DNS servers and the replica servers share the same Internet data center, the former may have accurate system condition information about the latter. Therefore, the low-level DNS servers may quickly react to sudden changes, such as flash crowds or replica server failures.

An original transparent redirection scheme is exploited in SPREAD, which makes proxies responsible for client redirection [Rodriguez and Sibal 2000]. SPREAD assumes the existence of a distributed infrastructure of proxies, each handling all HTTP traffic in its neighborhood. Each proxy works as follows. It inspects the HTTP-carrying IP packets and isolates those that are targeting replicated services. All other packets are routed traditionally. If the requested replica is not available locally, the service-related packets are forwarded to another proxy along the path toward the original service site. Otherwise, the proxy services them and generates IP packets carrying the response. The proxy rewrites source addresses in these packets, so that the client thought that the response originates from the original service site. The SPREAD scheme can be perceived as a distributed packet hand-off. It is transparent to the clients, but it requires a whole infrastructure of proxies.

7.2.2 Non-transparent mechanisms

Non-transparent redirection mechanisms reveal the redirection to the clients. In this way, these mechanisms introduce an explicit binding between a client and a given replica server. On the other hand, non-transparent redirection mechanisms are easier to implement than their transparent counterparts.

They also offer fine redirection granularity (per object), thus allowing for more flexible content management.

The simplest method that gives the effect of non-transparent redirection is to allow a client to choose from a list of available replica servers. This approach is called “manual redirection” and can often be found on Web services of widely-known corporations. However, since this method is entirely manual, it is of little use for replica hosting systems, which require an automated client redirection scheme.

Non-transparent redirection can be implemented with HTTP. It is another redirection mechanism supported by Web Server Director [Radware 2002]. An HTTP-based mechanism can redirect clients by rewriting object URLs inside HTML documents, so that these URLs point at object replicas stored on some replica servers. It is possible to treat each object URL separately, which allows for using virtually any replica placement. The two biggest advantages of the HTTP-based redirection are flexibility and simplicity. Its biggest drawback is the lack of transparency.

Cisco DistributedDirector also supports the HTTP-based redirection, although in a different manner [Delgadillo 1999]. Instead of rewriting URLs, this system exploits the HTTP 302 (temporary moved) response code. In this way, the redirecting machine does not need to store any service-related content – all it does is activate the redirection policy and redirect client to a replica server. On the other hand, this solution can efficiently redirect only per entire Web service, and not per object.

7.2.3 Combined mechanisms

It is possible to combine transparent and non-transparent redirection mechanisms to achieve a better result. Such approaches are followed by Akamai [Dilley et al. 2002], Cisco DistributedDirector [Radware 2002] and Web Server Director [Delgadillo 1999]. These systems allow to redirect clients using a “cascade” of different redirection mechanisms.

The first mechanism in the cascade is HTTP. A replica server may rewrite URLs inside an HTML document so that the URLs of different embedded objects contain different DNS names. Each DNS name identifies a group of replica servers that store a given object.

Although it is in general not recommended to scatter objects embedded in a single Web page over too many servers [Kangasharju et al. 2001], it may be sometimes beneficial to host objects of different *types* on separate groups of replica servers. For example, as video hosting may require specialized replica server resources, it may be reasonable to serve video streams with dedicated video servers, while providing images with other, regular ones. In such cases, video-related URLs contain a different DNS name (like “video.cdn.com”) than the image-related URLs (like “images.cdn.com”).

URL rewriting weakens the transparency, as the clients are able to discover that the content is retrieved from different replica servers. However, because the rewritten URLs contain DNS names that point to *groups* of replica servers, the clients are not bound to any *single* replica server. In this way, the system preserves the most important property of transparent redirection systems.

The second mechanism in the cascade is DNS. The DNS redirection system chooses the best replica server within each group by resolving the group-corresponding DNS name. In this way, the same DNS-based mechanism can be used to redirect a client to its several best replica servers, each belonging to a separate group.

By using DNS, the redirection system remains scalable, as it happens in the case of pure DNS-based mechanisms. By combining DNS with URL rewriting, however, the system may offer finer redirection granularity and thus allow for more flexible replica placement strategies.

The third mechanism in the cascade is packet hand-off. The processing capabilities of a replica server may be improved by deploying the replica server as a cluster of machines that share the same

IP address. In this case, the packet-handoff is implemented locally to scatter client requests across several machines.

Similarly to pure packet-handoff techniques, this part of the redirection cascade remains transparent for the clients. However, since packet hand-off is implemented only locally, the scalability of the redirection system is maintained.

As can be observed, combining different redirection mechanisms leads to constructing a redirection system that is simultaneously fine-grained, transparent, and scalable. The only potential problem is that deploying and maintaining such a mechanism is a complex task. In practice, however, this problem turns out to be just one more task of a replica hosting system operator. The duties like maintaining a set of reliable replica servers, managing multiple replicas of many objects, and making these replicas consistent, are likely to be at similar (if not higher) level of complexity.

7.3 Discussion

The problem of request routing can be divided into two sub-problems: devising a redirection policy and selecting a redirection mechanism. The policy decides to which replica a given client should be redirected, whereas the mechanism takes care of delivering this decision to the client.

We classify redirection policies into two groups: adaptive and non-adaptive ones. Non-adaptive policies perform well only when the system conditions do not change. If they do change, the system performance may turn out to be poor. Adaptive policies solve this problem by monitoring the system conditions and adjusting their behavior accordingly. However, they make the system more complex, as they need specialized metric estimation services. We note that all the investigated systems implement both adaptive and non-adaptive policies (see Table 4).

We classify redirection mechanisms into three groups: transparent, non-transparent, and combined ones. Transparent mechanisms can be based on DNS or packet hand-off. As can be observed in Table 4, DNS-based mechanisms are very popular. Among them, a particularly interesting is the scalable DNS-based mechanism built by Akamai. As for packet hand-off, its traditional limitation to clusters can be alleviated by means of a global infrastructure, as it is done in SPREAD. Non-transparent mechanisms are based on HTTP. They achieve finer redirection granularity, on the cost of introducing an explicit binding between a client and a replica server. Transparent and non-transparent mechanisms can be combined. Resulting hybrids offer fine-grained, transparent, and scalable redirection at the cost of higher complexity.

We observe that the request routing component has to cooperate with the metric estimation services to work efficiently. Consequently, the quality of the request routing component depends on the accuracy of the data provided by the metric estimation service.

Further, we note that simple, unadaptive policies sometimes work nearly as efficient as their adaptive counterparts. Although this phenomenon may justify using only non-adaptive policies in simple systems, we do believe that monitoring system conditions is of a key value for efficient request routing in large infrastructures. Moreover, combining several different metrics in the process of replica selection may additionally improve the system performance.

Finally, we are convinced that using combined redirection mechanisms is inevitable for large-scale wide-area systems. These mechanisms offer fine-grained, transparent, and scalable redirection on the cost of higher complexity. The resulting complexity, however, is not significantly larger compared to that of other parts of a replica hosting system. Since the ability to support millions of clients can be of fundamental importance, using a combined redirection mechanism is definitely worth the effort.

Table 4: The comparison of representative implementations of a redirection system

System	Redirection																
	Policies											Mechanisms					
	Adaptive						Non-Adaptive					TCP		DNS		HTTP	Comb.
	DST	LAT	NLD	CPU	USR	OTH	RR	%RQ	PRF	RND	PLD	CNT	DST	1LV	2LV		
Akamai	X	X	X	X		X									X	X	X
Globule	X						X		X					X			
Radar	X			X							X			X			
SPREAD	X												X				
Cisco DD	X	X					X	X	X	X		X		X		X	X
Web Direct			X		X	X	X					X		X		X	X

DST	: Network distance	OTH	: Other metrics	PLD	: Predicted load
LAT	: End-to-end latency	RR	: Round robin	CNT	: Centralized
NLD	: Network load	%RQ	: Percentage of requests	DST	: Distributed
CPU	: Replica server CPU load	PRF	: Server preference	1LV	: One-level
USR	: Number of users	RND	: Random selection	2LV	: Two-level

8 Conclusion

In this paper we have discussed the most important aspects of replica hosting system development. We have provided a generalized framework for such systems, which consists of five components: metric estimation, adaptation triggering, replica placement, consistency enforcement, and request routing. The framework has been built around an objective function, which allows to formally express the goals of replica hosting system development. For each of the framework components, we have discussed its corresponding problems, described several solutions for them, and reviewed some representative research efforts.

We observe that the work done on various problems related to a single component tends to be imbalanced. In metric estimation, for example, many papers focus on network-related metrics, such as distance and latency, whereas very little has been written about financial and consistency metrics. Similarly, in consistency enforcement, usually only time-based schemes are investigated, although their value- and order-based counterparts can also be useful. The reason for this situation may be that these ignored problems are rather non-technical, and addressing them requires knowledge exceeding the field of computer systems.

Another observation is that researchers seldom distinguish between the problems of server and content placement. Consequently, the proposed solutions are harder to apply to each of these problems separately – as these problems are usually tackled in very different circumstances, it seems to be unlikely that a single solution solves both of them. We find that a clear distinction between server and content placement should be more common among future research efforts.

Furthermore, there is a need for building economic models that enables comparison of orthogonal metrics like performance gain and financial returns. Such a model would enable the designer of a replica-hosting system to provide mechanisms that can capture the tradeoff between financial cost (of running and maintaining a new replica server) and possible performance gain (obtained by the new infrastructure).

We also notice that the amount of research effort devoted to different framework components is imbalanced as well. Particularly neglected ones are object selection and adaptation triggering, for which hardly any paper can be found. These problems can only be investigated in the context of an entire replica hosting system (and the applications that exploit it). This drives the need for more research efforts that do not assume the components to be *independent* from each other. Instead, they should investigate the components in the context of an entire system, with special attention paid to interactions and dependencies among them.

References

ABOBA, B., ARKKO, J., AND HARRINGTON, D. 2000. Introduction to Accounting Management. RFC 2975.

AGGARWAL, A. AND RABINOVICH, M. 1998. Performance of Replication Schemes for an Internet Hosting Service. Tech. Rep. HA6177000-981030- 01-TM, AT&T Research Labs, Florham Park, NJ. Oct.

ALVISI, L. AND MARZULLO, K. 1998. Message Logging: Pessimistic, Optimistic, Causal, and Optimal. *IEEE Transactions on Software Engineering 24,* 2 (Feb.), 149–159.

ANDREWS, M., SHEPHERD, B., SRINIVASAN, A., WINKLER, P., AND ZANE, F. 2002. Clustering and Server Selection Using Passive Monitoring. In *Proc. 21st INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA.

ARDAIZ, O., FREITAG, F., AND NAVARRO, L. 2001. Improving the Service Time of Web Clients using Server Redirection. In *Proc. 2nd Workshop on Performance and Architecture of Web Servers*. ACM Press, New York, NY.

BALIKRISHNAN, H., KAASHOEK, M. F., KARGER, D., MORRIS, R., AND STOICA, I. 2003. Looking up Data in P2P Systems. *Communications of the ACM 46,* 2 (Feb.), 43–48.

BALLINTIJN, G., VAN STEEN, M., AND TANENBAUM, A. 2000. Characterizing Internet Performance to Support Wide-area Application Development. *Operating Systems Review 34,* 4 (Oct.), 41–47.

BARBIR, A., CAIN, B., DOUGLIS, F., GREEN, M., HOFFMAN, M., NAIR, R., POTTER, D., AND SPATSCHECK, O. 2002. Known CDN Request-Routing Mechanisms. Work in progress.

BARFORD, P., CAI, J.-Y., AND GAST, J. 2001. Cache Placement Methods Based on Client Demand Clustering. Tech. Rep. TR1437, University of Wisconsin at Madison. July.

BERNSTEIN, P. A. AND GOODMAN, N. 1983. The Failure and Recovery Problem for Replicated Databases. In *Proc. 2nd Symposium on Principles of Distributed Computing*. ACM Press, New York, NY, 114–122.

BHIDE, M., DEOLASEE, P., KATKAR, A., PANCHBUDHE, A., RAMAMRITHAM, K., AND SHENOY, P. 2002. Adaptive Push-Pull: Disseminating Dynamic Web Data. *IEEE Transactions on Computers 51,* 6 (June), 652–668.

CAO, P. AND LIU, C. 1998. Maintaining Strong Cache Consistency in the World Wide Web. *IEEE Transactions on Computers 47,* 4 (Apr.), 445–457.

CAO, P., ZHANG, J., AND BEACH, K. 1998. Active Cache: Caching Dynamic Contents on the Web. In *Proc. Middleware '98*. Springer-Verlag, Berlin, 373–388.

CARDELLINI, V., CASALICCHIO, E., COLAJANNI, M., AND YU, P. 2002. The State of the Art in Locally Distributed Web-Server Systems. *ACM Computing Surveys 34,* 2 (June), 263–311.

CARDELLINI, V., COLAJANNI, M., AND YU, P. 1999. Dynamic Load Balancing on Web-Server Systems. *IEEE Internet Computing 3,* 3 (May), 28–39.

CARDELLINI, V., COLAJANNI, M., AND YU, P. S. 2003. Request Redirection Algorithms for Distributed Web Systems. *IEEE Transactions on Parallel and Distributed Systems 14,* 4 (Apr.), 355–368.

CARTER, R. L. AND CROVELLA, M. E. 1997. Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks. In *Proc. 16th INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA., 1014–1021.

CASTRO, M., COSTA, M., KEY, P., AND ROWSTRON, A. 2003. PIC: Practical Internet Coordinates for Distance Estimation. Tech. Rep. MSR-TR-2003-53, Microsoft Research Laboratories. Sept.

CASTRO, M., DRUSCHEL, P., HU, Y. C., AND ROWSTRON, A. 2003. Proximity Neighbor Selection in Tree-based Structured Peer-to-Peer Overlays. Tech. Rep. MSR-TR-2003-52, Microsoft Research, Cambridge, UK.

CATE, V. 1992. Alex – A Global File System. In *Proc. File Systems Workshop*. USENIX, Berkeley, CA, 1–11.

CHALLENGER, J., DANTZIG, P., AND IYENGAR, A. 1999. A Scalable System for Consistently Caching Dynamic Web Data. In *Proc. 18th INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA., 294–303.

CHANDRA, P., CHU, Y.-H., FISHER, A., GAO, J., KOSAK, C., NG, T. E., STEENKISTE, P., TAKAHASHI, E., AND ZHANG, H. 2001. Darwin: Customizable Resource Management for Value-Added Network Services. *IEEE Network 1,* 15 (Jan.), 22–35.

CHEN, Y., KATZ, R., AND KUBIATOWICZ, J. 2002. Dynamic Replica Placement for Scalable Content Delivery. In *Proc. 1st International Workshop on Peer-to-Peer Systems*. Lecture Notes on Computer Science, vol. 2429. Springer-Verlag, Berlin, 306–318.

CHEN, Y., QIU, L., CHEN, W., NGUYEN, L., AND KATZ, R. H. 2002. Clustering Web Content for Effi cient Replication. In *Proc. 10th International Conference on Network Protocols*. IEEE Computer Society Press, Los Alamitos, CA.

CHEN, Y., QIU, L., CHEN, W., NGUYEN, L., AND KATZ, R. H. 2003. Effi cient and Adaptive Web Replication Using Content Clustering. *IEEE Journal on Selected Areas in Communication 21,* 6 (Aug.), 979–994.

COHEN, E. AND KAPLAN, H. 2001. Proactive Caching of DNS Records: Addressing a Performance Bottleneck. In *Proc. 1st Symposium on Applications and the Internet*. IEEE Computer Society Press, Los Alamitos, CA.

CONTI, M., GREGORI, E., AND LAPENNA, W. 2002. Replicated Web Services: A Comparative Analysis of Client-Based Content Delivery Policies. In *Proc. Networking 2002 Workshops*. Lecture Notes on Computer Science, vol. 2376. Springer-Verlag, Berlin, 53–68.

COX, R., DABEK, F., KAASHOEK, F., LI, J., AND MORRIS, R. 2004. Practical, Distributed Network Coordinates. *ACM Computer Communications Review 34,* 1 (Jan.), 113–118.

CROVELLA, M. AND CARTER, R. 1995. Dynamic Server Selection in the Internet. In *Proc. 3rd Workshop on High Performance Subsystems*. IEEE Computer Society Press, Los Alamitos, CA.

DA CUNHA, C. R. 1997. Trace Analysis and its Applications to Performance Enhancements of Distributed Information Systems. Ph.D. Thesis, Boston University.

DELGADILLO, K. 1999. Cisco DistributedDirector. Tech. rep., Cisco Systems, Inc. June.

DILLEY, J., MAGGS, B., PARIKH, J., PROKOP, H., SITARAMAN, R., AND WEIHL, B. 2002. Globally Distributed Content Delivery. *IEEE Internet Computing 6,* 5 (Sept.), 50–58.

DUVVURI, V., SHENOY, P., AND TEWARI, R. 2000. Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web. In *Proc. 19th INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA., 834–843.

DYKES, S. G., ROBBINS, K. A., AND JEFFREY, C. L. 2000. An Empirical Evaluation of Client-side Server Selection. In *Proc. 19th INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA., 1361–1370.

ELNOZAHY, E., ALVISI, L., WANG, Y.-M., AND JOHNSON, D. 2002. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. *ACM Computing Surveys 34,* 3 (Sept.), 375–408.

FEI, Z. 2001. A Novel Approach to Managing Consistency in Content Distribution Networks. In *Proc. 6th Web Caching Workshop*. North-Holland, Amsterdam.

FRANCIS, P., JAMIN, S., JIN, C., JIN, Y., RAZ, D., SHAVITT, Y., AND ZHANG, L. 2001. IDMaps: Global Internet Host Distance Estimation Service. *IEEE/ACM Transactions on Networking 9,* 5 (Oct.), 525–540.

FRANCIS, P., JAMIN, S., PAXSON, V., ZHANG, L., GRYNIEWICZ, D., AND JIN, Y. 1999. An Architecture for a Global Internet Host Distance Estimation Service. In *Proc. 18th INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA., 210–217.

FU, Y., CHERKASOVA, L., TANG, W., AND VAHDAT, A. 2002. EtE: Passive End-to-End Internet Service Performance Monitoring. In *Proc. USENIX Annual Technical Conference*. USENIX, Berkeley, CA, 115–130.

GAO, L., DAHLIN, M., NAYATE, A., ZHENG, J., AND IYENGAR, A. 2003. Application Specifi c Data Replication for Edge Services . In *Proc. 12th International World Wide Web Conference*. ACM Press, New York, NY.

GRAY, C. AND CHERITON, D. 1989. Leases: An Effi cient Fault-Tolerant Mechanism for Distributed File Cache Consistency. In *Proc. 12th Symposium on Operating System Principles*. ACM Press, New York, NY, 202–210.

GUMMADI, K. P., SAROIU, S., AND GRIBBLE, S. D. 2002. King: Estimating Latency between Arbitrary Internet End Hosts. In *Proc. 2nd Internet Measurement Workshop*. ACM Press, New York, NY, 5–18.

HUFFAKER, B., FOMENKOV, M., PLUMMER, D. J., MOORE, D., AND CLAFFY, K. 2002. Distance Metrics in the Internet. In *Proc. International Telecommunications Symposium*. IEEE Computer Society Press, Los Alamitos, CA.

HULL, S. 2002. *Content Delivery Networks*. McGraw-Hill, New York, NY.

IYENGAR, A., MACNAIR, E., AND NGUYEN, T. 1997. An Analysis of Web Server Performance. In *Proc. Globecom*. IEEE Computer Society Press, Los Alamitos, CA.

JALOTE, P. 1994. *Fault Tolerance in Distributed Systems*. Prentice Hall, Englewood Cliffs, N.J.

JANIGA, M. J., DIBNER, G., AND GOVERNALI, F. J. 2001. Internet Infrastructure: Content Delivery. Goldman Sachs Global Equity Research.

JOHNSON, K. L., CARR, J. F., DAY, M. S., AND KAASHOEK, M. F. 2001. The Measured Performance of Content Distribution Networks. *Computer Communications 24,* 2 (Feb.), 202–206.

JUNG, J., KRISHNAMURTHY, B., AND RABINOVICH, M. 2002. Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In *Proc. 11th International World Wide Web Conference*. ACM Press, New York, NY, 293–304.

KANGASHARJU, J., ROBERTS, J., AND ROSS, K. 2001. Object Replication Strategies in Content Distribution Networks. In *Proc. 6th Web Caching Workshop*. North-Holland, Amsterdam.

KANGASHARJU, J., ROSS, K., AND ROBERTS, J. 2001. Performance Evaluation of Redirection Schemes in Content Distribution Networks. *Computer Communications 24,* 2 (Feb.), 207–214.

KARAUL, M., KORILIS, Y., AND ORDA, A. 1998. A Market-Based Architecture for Management of Geographically Dispersed, Replicated Web Servers. In *Proc. 1st International Conference on Information and Computation Economics*. ACM Press, New York, NY, 158–165.

KARGER, D., SHERMAN, A., BERKHEIMER, A., BOGSTAD, B., DHANIDINA, R., IWAMOTO, K., KIM, B., MATKINS, L., AND YERUSHALMI, Y. 1999. Web Caching with Consistent Hashing. In *Proc. 8th International World Wide Web Conference*. Toronto, Canada.

KARLSSON, M. AND KARAMANOLIS, C. 2004. Choosing Replica Placement Heuristics for Wide-Area Systems. In *Proc. 24th International Conference on Distributed Computing Systems*. IEEE Computer Society Press, Los Alamitos, CA.

KARLSSON, M., KARAMANOLIS, C., AND MAHALINGAM, M. 2002. A Framework for Evaluating Replica Placement Algorithms. Tech. rep., HP Laboratories, Palo Alto, CA.

KRISHNAKUMAR, N. AND BERNSTEIN, A. J. 1994. Bounded Ignorance: A Technique for Increasing Concurrency in a Replicated System. *ACM Transactions on Database Systems 4,* 19, 586–625.

KRISHNAMURTHY, B. AND WANG, J. 2000. On Network-Aware Clustering of Web Clients. In *Proc. SIGCOMM*. ACM Press, New York, NY, 97–110.

LABRINIDIS, A. AND ROUSSOPOULOS, N. 2000. WebView Materialization. In *Proc. SIGMOD International Conference on Management Of Data*. ACM Press, New York, NY, 367–378.

LAI, K. AND BAKER, M. 1999. Measuring Bandwidth. In *Proc. 18th INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA., 235–245.

LEIGHTON, F. AND LEWIN, D. 2000. Global Hosting System. United States Patent, Number 6,108,703.

LI, B., GOLIN, M. J., ITALIANO, G. F., AND DENG, X. 1999. On the Optimal Placement of Web Proxies in the Internet. In *Proc. 18th INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA., 1282–1290.

MAO, Z. M., CRANOR, C. D., DOUGLIS, F., RABINOVICH, M., SPATSCHECK, O., AND WANG, J. 2002. A Precise and Effi cient Evaluation of the Proximity between Web Clients and their Local DNS Servers. In *Proc. USENIX Annual Technical Conference*. USENIX, Berkeley, CA, 229–242.

MCCUNE, T. AND ANDRESEN, D. 1998. Towards a Hierarchical Scheduling System for Distributed WWW Server Clusters. In *Proc. 7th International Symposium on High Performance Distributed Computing*. IEEE Computer Society Press, Los Alamitos, CA., 301–309.

MCMANUS, P. R. 1999. A Passive System for Server Selection within Mirrored Resource Environments Using AS Path Length Heuristics.

MOCKAPETRIS, P. 1987a. Domain Names - Concepts and Facilities. RFC 1034.

MOCKAPETRIS, P. 1987b. Domain Names - Implementation and Specifi cation. RFC 1035.

MOGUL, J. C., DOUGLIS, F., FELDMANN, A., AND KRISHNAMURTHY, B. 1997. Potential Benefi ts of Delta Encoding and Data Compression for HTTP. In *Proc. SIGCOMM*. ACM Press, New York, NY, 181–194.

MOORE, K., COX, J., AND GREEN, S. 1996. Sonar - A network proximity service. Internet-draft. [Online] `http://www.netlib.org/utk/projects/sonar/`.

MOSBERGER, D. 1993. Memory Consistency Models. *Operating Systems Review 27,* 1 (Jan.), 18–26.

NELDER, J. A. AND MEAD, R. 1965. A Simplex Method for Function Minimization. *The Computer Journal 4,* 7, 303–318.

NG, E. AND ZHANG, H. 2002. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proc. 21st INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA.

NINAN, A., KULKARNI, P., SHENOY, P., RAMAMRITHAM, K., AND TEWARI, R. 2002. Cooperative Leases: Scalable Consistency Maintenance in Content Distribution Networks. In *Proc. 11th International World Wide Web Conference*. ACM Press, New York, NY, 1–12.

OBRACZKA, K. AND SILVA, F. 2000. Network Latency Metrics for Server Proximity. In *Proc. Globecom*. IEEE Computer Society Press, Los Alamitos, CA.

ODLYZKO, A. 2001. Internet Pricing and the History of Communications. *Computer Networks 36*, 493–517.

PAI, V., ARON, M., BANGA, G., SVENDSEN, M., DRUSCHEL, P., ZWAENEPOEL, W., AND NAHUM, E. 1998. Locality-Aware Request Distribution in Cluster-Based Network Servers. In *Proc. 8th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM Press, New York, NY, 205–216.

PANSIOT, J. AND GRAD, D. 1998. On Routes and Multicast Trees in the Internet. *ACM Computer Communications Review 28,* 1, 41–50.

PAXSON, V. 1997a. End-to-end Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking 5,* 5 (Oct.), 601–615.

PAXSON, V. 1997b. Measurements and Analysis of End-to-End Internet Dynamics. Tech. Rep. UCB/CSD-97-945, University of California at Berkeley. Apr.

PEPELNJAK, I. AND GUICHARD, J. 2001. *MPLS and VPN Architectures*. Cisco Press, Indianapolis, IN.

PIAS, M., CROWCROFT, J., WILBUR, S., HARRIS, T., AND BHATTI, S. 2003. Lighthouses for Scalable Distributed Location. In *Proc. 2nd International Workshop on Peer-to-Peer Systems*. Vol. 2735. Springer-Verlag, Berlin, 278–291.

PIERRE, G. AND VAN STEEN, M. 2001. Globule: A Platform for Self-Replicating Web Documents. In *Proc. Protocols for Multimedia Systems*. Lecture Notes on Computer Science, vol. 2213. Springer-Verlag, Berlin, 1–11.

PIERRE, G. AND VAN STEEN, M. 2003. Design and Implementation of a User-Centered Content Delivery Network. In *Proc. 3rd Workshop on Internet Applications*. IEEE Computer Society Press, Los Alamitos, CA.

PIERRE, G., VAN STEEN, M., AND TANENBAUM, A. 2002. Dynamically Selecting Optimal Distribution Strategies for Web Documents. *IEEE Transactions on Computers 51,* 6 (June), 637–651.

PRADHAN, D. 1996. *Fault-Tolerant Computer System Design*. Prentice Hall, Englewood Cliffs, N.J.

QIU, L., PADMANABHAN, V., AND VOELKER, G. 2001. On the Placement of Web Server Replicas. In *Proc. 20th INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA., 1587–1596.

RABINOVICH, M. AND AGGARWAL, A. 1999. Radar: A Scalable Architecture for a Global Web Hosting Service. *Computer Networks 31,* 11–16, 1545–1561.

RABINOVICH, M. AND SPASTSCHECK, O. 2002. *Web Caching and Replication*. Addison-Wesley, Reading, MA.

RABINOVICH, M., XIAO, Z., AND AGGARWAL, A. 2003. Computing on the Edge: A Platform for Replicating Internet Applications. In *Proc. 8th Web Caching Workshop*.

RADOSLAVOV, P., GOVINDAN, R., AND ESTRIN, D. 2001. Topology-Informed Internet Replica Placement. In *Proc. 6th Web Caching Workshop*. North-Holland, Amsterdam.

RADWARE. 2002. Web Server Director. Tech. rep., Radware, Inc. Aug.

RAYNAL, M. AND SINGHAL, M. 1996. Logical Time: Capturing Causality in Distributed Systems. *Computer 29,* 2 (Feb.), 49–56.

REKHTER, Y. AND LI, T. 1995. A Border Gateway Protocol 4 (BGP-4). RFC 1771.

RODRIGUEZ, P., KIRPAL, A., AND BIERSACK, E. 2000. Parallel-Access for Mirror Sites in the Internet. In *Proc. 19th INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA., 864–873.

RODRIGUEZ, P. AND SIBAL, S. 2000. SPREAD: Scalable Platform for Reliable and Effi cient Automated Distribution. *Computer Networks 33,* 1–6, 33–46.

RODRIGUEZ, P., SPANNER, C., AND BIERSACK, E. 2001. Analysis of Web Caching Architecture: Hierarchical and Distributed Caching. *IEEE/ACM Transactions on Networking 21,* 4 (Aug.), 404–418.

SAITO, Y. AND SHAPIRO, M. 2003. Optimistic Replication. Tech. Rep. MSR-TR-2003-60, Microsoft Research Laboratories. Sept.

SAYAL, M., SHEUERMANN, P., AND VINGRALEK, R. 2003. Content Replication in Web++. In *Proc. 2nd International Symposium on Network Computing and Applications*. IEEE Computer Society Press, Los Alamitos, CA.

SCHNEIDER, F. 1990. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys 22,* 4 (Dec.), 299–320.

SHAIKH, A., TEWARI, R., AND AGRAWAL, M. 2001. On the Effectiveness of DNS-based Server Selection. In *Proc. 20th INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA., 1801–1810.

SHAVITT, Y. AND TANKEL, T. 2003. Big-Bang Simulation for Embedding Network Distances in Euclidean Space. In *Proc. 22nd INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA.

SHMOYS, D., TARDOS, E., AND AARDAL, K. 1997. Approximation Algorithms for Facility Location Problems. In *Proc. 29th Symposium on Theory of Computing*. ACM Press, New York, NY, 265–274.

SIVASUBRAMANIAN, S., PIERRE, G., AND VAN STEEN, M. 2003. A Case for Dynamic Selection of Replication and Caching Strategies. In *Proc. 8th Web Caching Workshop*.

STEMM, M., KATZ, R., AND SESHAN, S. 2000. A Network Measurement Architecture for Adaptive Applications. In *Proc. 19th INFOCOM Conference*. IEEE Computer Society Press, Los Alamitos, CA., 285–294.

SZYMANIAK, M., PIERRE, G., AND VAN STEEN, M. 2003. Netairt: A DNS-based Redirection System for Apache. In *Proc. International Conference WWW/Internet*.

SZYMANIAK, M., PIERRE, G., AND VAN STEEN, M. 2004. Scalable Cooperative Latency Estimation. In *Proc. 10th International Conference on Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA.

TERRY, D., DEMERS, A., PETERSEN, K., SPREITZER, M., THEIMER, M., AND WELSH, B. 1994. Session Guarantees for Weakly Consistent Replicated Data. In *Proc. 3rd International Conference on Parallel and Distributed Information Systems*. IEEE Computer Society Press, Los Alamitos, CA., 140–149.

TEWARI, R., NIRANJAN, T., AND RAMAMURTHY, S. 2002. WCDP: A Protocol for Web Cache Consistency. In *Proc. 7th Web Caching Workshop*.

TORRES-ROJAS, F. J., AHAMAD, M., AND RAYNAL, M. 1999. Timed consistency for shared distributed objects. In *Proc. 18th Symposium on Principles of Distributed Computing*. ACM Press, New York, NY, 163–172.

VERMA, D. C. 2002. *Content Distribution Networks: An Engineering Approach*. John Wiley, New York.

WALDVOGEL, M. AND RINALDI, R. 2003. Effi cient topology-aware overlay network. *ACM Computer Communications Review 33,* 1 (Jan.), 101–106.

WANG, J. 1999. A Survey of Web Caching Schemes for the Internet. *ACM Computer Communications Review 29,* 5 (Oct.), 36–46.

WANG, L., PAI, V., AND PETERSON, L. 2002. The Effectiveness of Request Redirection on CDN Robustness. In *Proc. 5th Symposium on Operating System Design and Implementation*. USENIX, Berkeley, CA.

WOLSKI, R., SPRING, N., AND HAYES, J. 1999. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems 15,* 5-6 (Oct.), 757–768.

XIAO, J. AND ZHANG, Y. 2001. Clustering of Web Users Using Session-Based Similarity Measures. In *Proc. International Conference on Computer Networks and Mobile Computing*. IEEE Computer Society Press, Los Alamitos, CA., 223–228.

YIN, J., ALVISI, L., DAHLIN, M., AND IYENGAR, A. 2002. Engineering Web Cache Consistency. *ACM Transactions on Internet Technology 2,* 3 (Aug.), 224–259.

YU, H. AND VAHDAT, A. 2000. Effi cient Numerical Error Bounding for Replicated Network Services. In *Proc. 26th International Conference on Very Large Data Bases*, A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, Eds. Morgan Kaufman, San Mateo, CA., 123–133.

YU, H. AND VAHDAT, A. 2002. Design and Evaluation of a Conit-Based Continuous Consistency Model for Replicated Services. *ACM Transactions on Computer Systems 20,* 3, 239–282.

ZARI, M., SAIEDIAN, H., AND NAEEM, M. 2001. Understanding and Reducing Web Delays. *Computer 34,* 12 (Dec.), 30–37.

ZHAO, B., HUANG, L., STRIBLING, J., RHEA, S., JOSEPH, A., AND KUBIATOWICZ, J. 2004. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communication 22,* 1 (Jan.), 41–53.