Dynamically Selecting Optimal Distribution Strategies for Web Documents

Guillaume Pierre, Maarten van Steen, *Member*, *IEEE*, and
Andrew S. Tanenbaum, *Fellow*, *IEEE*

Abstract—To improve the scalability of the Web, it is common practice to apply caching and replication techniques. Numerous strategies for placing and maintaining multiple copies of Web documents at several sites have been proposed. These approaches essentially apply a global strategy by which a single family of protocols is used to choose replication sites and keep copies mutually consistent. We propose a more flexible approach by allowing each distributed document to have its own associated strategy. We propose a method for assigning an optimal strategy to each document separately and prove that it generates a family of optimal results. Using trace-based simulations, we show that optimal assignments clearly outperform any global strategy. We have designed an architecture for supporting documents that can dynamically select their optimal strategy and evaluate its feasibility.

Index Terms—Web replication and caching, adaptive replication, Web documents, wide-area distributed systems.

---◆---

1 INTRODUCTION

W EB users often experience slow document transfers caused by poorly performing servers and overloaded parts of the network. These scalability problems are commonly tackled through caching and replication, by which multiple copies of a document are distributed across servers in the network [24]. A user's request for a document is then directed to a nearby copy, thus reducing access times, average server loads, and overall network traffic.

There are several ways in which copies of a document can be distributed. In general, a distinction is made between caching and replication. With *caching*, whenever a user requests a document for the first time, the client process or local server handling the request will fetch a copy from the document's server. Before passing it to the user, the document is stored locally in a cache. Whenever that document is requested again, it can simply be fetched from the cache. In principle, there is no need to contact the document's server again; the request can be entirely handled locally. In the case of *replication*, a document's server proactively places copies of the document at various servers in the network, anticipating that enough clients will make use of those copies that warrant their replication. Apart from this difference in the creation time of copies, we consider caching and replication to be fundamentally the same mechanism.

Although caching and replication can significantly alleviate scalability problems, having multiple copies of a document also introduces a consistency problem. Whenever a document is updated, it is necessary to ensure that all copies are brought up-to-date as well; otherwise, clients may access stale data. Unfortunately, maintaining strong consistency, that is, keeping all copies of a document identical, is often costly. For example, in the case of a cache, it may be necessary to first contact the document's server to see if the cached copy is still valid. Contacting the server introduces global communication that may negate the performance initially gained by caching. In the case of replication, strong consistency may require that updates are immediately propagated to all copies, even if there are currently no clients requesting the document. In that case, update propagation wastes network bandwidth.

A solution to this problem is to weaken the consistency for carefully selected documents. For example, many Web caches follow a policy in which a cached document is always returned to the requesting user without checking for consistency with the document's server. To avoid having cached documents becoming too old to be useful, each one has an associated expiration time beyond which it is purged from the cache. This caching policy is derived from the Alex file system [7] and is followed in the widely used Squid caches [8].

Weak consistency is not universally applicable. In fact, in most cases, users simply want to have an up-to-date copy of a document returned when requested and protocols implementing weak consistency generally fail to meet this requirement. Unfortunately, since there are several evaluation metrics involved, it is not clear what the best solution for maintaining strong consistency is. We argue that there is no single best solution and that, for each document, it should be decided separately what the best strategy is to distribute its copies and to keep them consistent. Moreover, servers must be prepared to dynamically adapt a strategy for a document, for example, because its usage and update pattern have changed.

We have conducted a series of trace-driven simulation experiments that substantiate these claims. In this paper, we first propose a method for associating a replication policy to each document and prove that it generates a family of

---

● *The authors are with the Department of Mathematics and Computer Science, Vrije Universiteit, de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands. E-mail: {gpierre, steen, ast}@cs.vu.nl.*
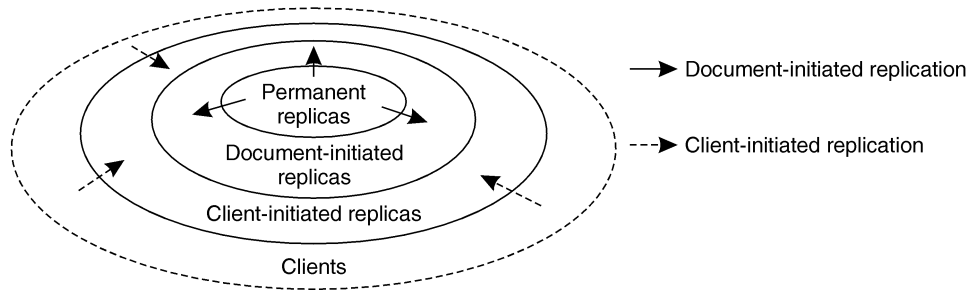
Fig. 1. Conceptual layering of document hosts.

optimal assignments. We then show that the resulting assignments perform significantly better than assignments where every document uses the same replication strategy. Finally, we propose an architecture for supporting replicated Web documents that can dynamically analyze their recent access pattern and select the policy which suits them best.

Preliminary results have been reported in [28]. However, it presented only empirical results based on a single trace file, which confirmed our intuition that differentiating replication strategies provides better performance than one-size-fits-all assignments. The contributions made in the present paper are threefold: 1) We provide a mathematical proof of optimality of our assignment method; 2) simulations are based on two different trace files, and 3) we also now present results on the dynamic adaptation of replication policies. None of these contributions have been reported in [28].

The paper is organized as follows: In Section 2, we present trade-offs that need to be made when implementing replication strategies and discuss related work on selecting a replication policy. We proceed with describing how we evaluate strategies in Section 3 and describe the results of our simulations in Section 4. In Section 5, we propose and evaluate an architecture for supporting our approach to distributed Web documents. We conclude in Section 6.

2 RELATED WORK

There are many ways in which copies of a Web document can be distributed across multiple servers. One has to decide how many copies are needed, where and when to create them, and how to keep them consistent. We define a replication policy as an algorithm that makes these decisions.

We briefly describe commonly used placement protocols (which decide on where and when to create copies) and consistency protocols (which decide on how to keep the copies consistent). We then discuss the existing results on selecting an appropriate replication policy.

2.1 Placement Protocols

A *placement protocol* is used to determine when and where a copy of a document is to be placed or removed. Placement can be initiated either by servers or clients, as shown in Fig. 1. We distinguish three different layers of hosts that can hold a copy of a document.

The core layer consists of servers that host *permanent replicas* of a document. In many cases, each Web document is hosted by only a single primary server. Clusters of Web servers [1], [12] and servers that mirror entire Web sites are examples of multiple permanent replicas.

The middle layer consists of servers for hosting *document-initiated replicas*. These replicas are normally created by one of the permanent replicas, but possibly also by one of the other document-initiated replicas. In the context of the Internet, document-initiated replicas appear in *Content Delivery Networks* (CDNs), such as RaDaR [32] and Akamai [20]. In these systems, content is transferred to servers in the proximity of requesting clients. How the transfer is initiated is part of the consistency protocol, as we describe below.

The outer layer consists of servers for hosting *client-initiated replicas*, also known as cache servers. Creating a cached version of a document is entirely a local decision that is, in principle, taken independently from the replication strategy of the document. However, the decision to cache may be subject to many constraints. For example, a client may decide to cache only those documents that it expects will not change soon. Also, it may have limited disk space available for caching. Web proxy caches form a typical example of client-initiated replicas in the Internet.

2.2 Consistency Protocols

A *consistency protocol* implements a specific consistency model. There are various trade-offs to be made when implementing such models and the most efficient implementation is often dependent on the current state of the network or usage of the document. Table 1 shows various parameters by which consistency protocols can be characterized (an overview of various algorithms can be found in [33]).

Change distribution is about *what* is distributed. Possibilities include a notification that something has changed (such as an invalidation message) [6], the full content of a document [13], only differences [3], or the operation that caused a change.

Another issue is *when* a replica actually responds after it notices it contains stale data. For example, assume a replica has just processed an update sent to it by a client. Its responsiveness describes whether it immediately propagates the update or waits some time, as in lazy replication [19]. It may also decide to do nothing and wait until a validation request is issued by another replica.

The reaction of a replica is another design issue and describes *how* an update is propagated. There are essentially

TABLE 1
Various Trade-Offs for Implementing Consistency Protocols

Parameter	Values	Meaning
Change distribution	- notification - full state - state differences - operation	Describes how changes between replicas are distributed: is only a notification (or invalidation) sent telling that an update is needed, is the full state sent, or only differences, or is the operation sent that is to be carried out to update the receiver's state?
Replica responsiveness	- immediate - lazy (e.g., periodic) - passive	Describes how quickly a replica reacts when it notices it is no longer consistent with the other replicas. A passive replica will do nothing.
Replica reaction	- pull - push	Describes what a (non passive) replica does when it notices it is inconsistent with other replicas. It either sends or requests updates.
Write set	- single - multiple	This parameter gives the number of writers that may simultaneously access the document.
Coherence group	- permanent only - permanent and document-initiated	Describes who implements the consistency model: permanent and/or document-initiated replicas. Caches are generally not part of the coherence group.

only two alternatives: An update is either pushed to a replica or a replica pulls in an update from another replica. Most Web caching schemes follow a pull-based strategy.

When dealing with replication, it is important to consider how many processes are allowed to update a document. If the number of updating processes is one, then inconsistencies resulting from concurrent updates cannot occur. In many existing solutions, there is a single primary server that serializes all updates. For Web-based applications, this approach often works because there is only a single owner for a document. However, problems may arise in the case of collaborative Web-based applications, such as proposed in [38].

Finally, it is important to consider which group of replicas is responsible for maintaining consistency. If there is only a single permanent replica, maintaining consistency is easy, although other replicas may be updated in a lazy fashion. This approach comes close to the single-writer case. With several replicas being responsible for maintaining consistency, various synchronization techniques are needed to ensure that updates are propagated in the order dictated by the consistency model.

In conclusion, we see that there is a large variety of strategies and that we cannot decide in advance which strategy is best. Our research concentrates on finding the best strategy for each document separately, aiming at a global optimization of various performance metrics, as we explain next.

2.3 Selecting an Appropriate Policy

Our work is based on the premise that it makes sense to differentiate caching and replication policies for Web documents. Obvious as this may seem, it is only recently that researchers have started to look for solutions that allow very different strategies to coexist in a single system.

Most research has concentrated on supporting a single *family* of strategies. For example, the TACT toolkit [40]

provides support for replication based on antientropy schemes [9] for a range of consistency models. In a similar fashion, caching algorithms exist that base their decisions on temporal correlations between requests [15], [35], but otherwise essentially follow the same protocol. Closer to our approach are systems that have protocols that adapt the way updates are propagated. For example, the adaptive leases described in [11] provide a way for switching from a protocol in which updates are pushed to replicas to one in which updates are pulled in on demand. Combinations of push and pull strategies are also possible, as described in [10].

Related to our work are systems that dynamically decide on the placement of replicas. Examples of such systems in the Web are content delivery networks like RaDaR [32] and Akamai [20]. These systems adapt the number and location of document copies to provide copies close to the users' locations and to balance the load of servers. Dynamic placement of replicas has also been exploited in areas such as parallelism [2] and distributed databases [39].

Studying past access patterns to optimize future behavior of a system is not a new idea in the Web community. Services such as prefetching, for example, rely on past access analysis to determine which documents are worth downloading [14], [21], [26]. Other systems dynamically organize the search path for a URL among a cache mesh based on a shared knowledge of caches' contents [23].

All these systems have in common that they do not provide support for very different consistency protocols. At best, they offer facilities for optimizing protocols that belong to a single (relatively small) family of solutions, although optimizations can often be done on a per-object basis.

Also related to our work are systems that can dynamically change their internal composition. Such flexibility has been deployed in many domains. Flexible group communication systems, such as the x-kernel [25] and Horus [36]
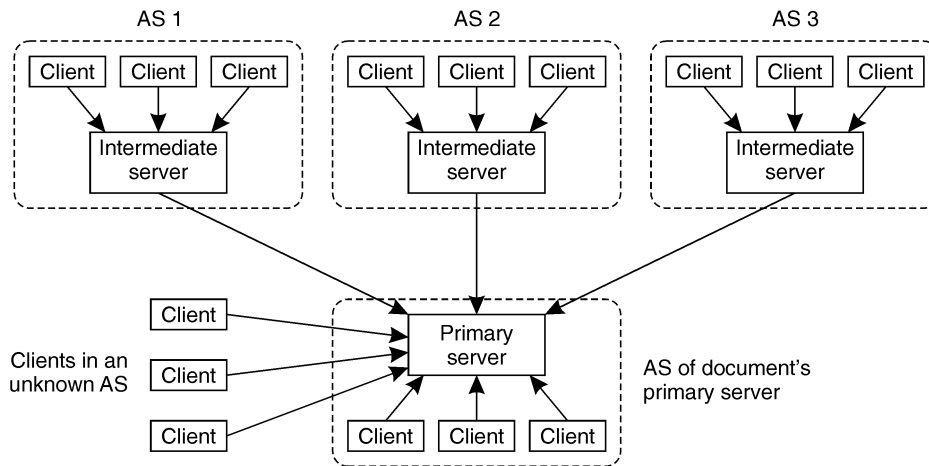
Fig. 2. The general organization of clients, intermediate servers, and primary server for a single Web document.

split protocols into elementary modules that can be composed together to obtain required features. The same principle has been applied for building routers [17], network traffic analyzers [27], and so on. However, there are relatively few replication systems that allow one to statically or dynamically choose between different replication strategies. Examples of such systems are described in [5], [16], but neither of these have been deployed in the context of the Web.

3 EVALUATING REPLICATION STRATEGIES

To see whether per-document replication strategies are useful, we set up an experiment in which we compared various strategies. Our experiment consisted of collecting access and update traces from various Web servers and replaying those traces for different caching and replication strategies. In this section, we explain our approach to evaluating strategies using trace-driven simulations.

3.1 Simulation Model

To simplify matters, we assume that each document has a single owner who is allowed to update its content. Each document has an associated *primary server* holding a main copy. In terms of our model, presented in the previous section, we adopt the situation that there is only a single permanent replica. Any other server hosting a replica of a document is referred to as a *secondary server* for that document.

We consider only static documents, that is, documents of which the content changes only when they are updated by their owner. Conceptually, when the primary server of such a document receives a request, it merely needs to fetch the document from its local file system and initiate a transfer to the requesting client. A widely applied alternative is to generate a document on each request using a myriad of techniques, such as server-side scripts and CGI programs. However, in many cases, this approach can be emulated by replicating the generators to other servers along with the necessary data needed to construct the document that is to be returned to the client. For simplicity, we did not consider these dynamic documents in our experiments.

To evaluate caching and replication strategies, we need a model in which documents can be placed on multiple hosts throughout the Internet. The validity of this model is important for justifying the final results. We decided to group clients based on the autonomous system (AS) of which they are part. An AS plays a crucial role in deciding how to route packets across the Internet [4]. At the highest level, the Internet can be viewed as a routing network in which the ASes jointly form the nodes. ASes are pairwise connected based on various routing-policy decisions. Within an AS, routing takes place following an AS-specific internal routing protocol.

An interesting feature of many ASes that is relevant for our experiments is that an AS groups hosts that are relatively close to each other in a network-topological sense. In other words, communication performance within an AS is often much better than between different ASes. Based on this assumption, we decided to allocate at most one intermediate server for each AS in our simulation models. All clients within an AS forward their requests through the intermediate server for that AS, as shown in Fig. 2. Clients for which we could not determine their AS were assumed to directly contact a document's primary server. This approach for clustering clients is similar to that of [18], although at a coarser grain.

In our simulations, an intermediate server was configured either as a cache server or as a server for document-initiated replicas. Documents were never allowed to be cached longer than seven days. We also considered situations in which clients in an AS sent their requests directly to the primary.

We simulated the benefit that our approach would bring when replicating a given Web server's content on a worldwide basis. In order to keep simulation results as realistic as possible, we chose to run trace-based simulations [29]. Input data came from access traces that we collected for a Web server. In addition, we gathered traces at the primary on when documents were created or updated.

We also measured network performance to simulate transfer delays. For this purpose, we measured actual latency and bandwidth between the primary server for a
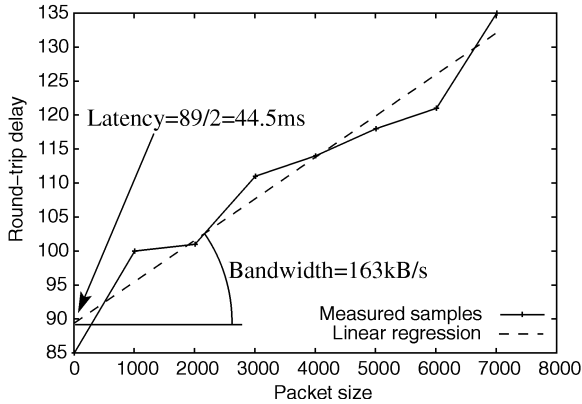
Fig. 3. Determining the network performance to a host, based on ping samples.

document and each of the intermediate servers. In other words, we did not adopt a hypothetical model of the network. To measure the network performance from our server to each AS in our experiment, we randomly chose five hosts inside each AS. For each of these hosts, we sent a number of "ping" packets of different sizes and measured the round-trip time. By running a linear regression, we approximated the latency and bandwidth of the network connection to these hosts. The latency corresponds to half of the round-trip delay for a packet of size 0; the bandwidth corresponds to additional delays caused by varying packet sizes (see Fig. 3). We assume symmetrical network performance: The communication from the server to any host is considered to perform equally well as the communication from that host to the server. Since more detailed information was not available, we assumed that communication performance within each AS was roughly the same.

We evaluated overall system performance for various strategies, using three performance metrics. For each client request, we measured its turnaround time, that is, the interval between submission of a request and completion of the response. Turnaround time values were added for a set of requests to obtain the total turnaround time. The second metric was the number of stale documents that were delivered to clients. Finally, we measured the consumed bandwidth for requests issued by intermediate servers or clients to the primary server, thus obtaining the total consumed bandwidth over inter-AS links.

The properties of these metrics allow us to simulate the behavior for each document separately and, subsequently, add the results per metric to obtain the overall performance for a set of documents.

3.2 Caching and Replication Strategies

Our experiment consisted of selecting a specific caching or replication strategy and distributing the copies of a document across the intermediate servers. Table 2 lists the various strategies that we evaluated. Besides applying no replication or caching at all (NR), a distinction was made between caching, replication, and hybrid strategies that combined both.

The CV caching strategy corresponds to having a cache server check the validity of a cached document by sending an If-Modified-Since request to the primary server each time a client asks for such a document. The CLV strategy has been implemented in the Alex file system [7]. When a document is cached, it is timestamped with an expiration time T_{expire} that depends on the last time the document was modified. A cached document is considered valid until its expiration time. If T_{cached} is the time when the document is stored in the cache and $T_{last\_modified}$ the time it was last modified, then

$$T_{expire} = T_{cached} + \alpha \cdot (T_{cached} - T_{last\_modified}),$$

TABLE 2
Evaluated Caching and Replication Strategies

Abbr.	Name	Description
NR	No replication	No replication or caching takes place. All clients forward their requests directly to the primary.
CV	Verification	Intermediate servers cache documents. At each subsequent request, the primary is contacted for revalidation.
CLV	Limited validity	Intermediate servers cache documents. A cached document has an associated expiration time before it becomes invalid and is removed from the cache.
CDV	Delayed verification	Intermediate servers cache documents. A cached document has an associated expiration time after which the primary is contacted for revalidation.
SI	Server invalidation	Intermediate servers cache documents, but the primary invalidates cached copies when the document is updated.
SUx	Server updates	The primary maintains copies at the x most relevant intermediate servers; $x = 10$, 25 or 50
SU50+CLV	Hybrid SU50 & CLV	The primary maintains copies at the 50 most relevant intermediate servers; the other intermediate servers follow the CLV strategy.
SU50+CDV	Hybrid SU50 & CDV	The primary maintains copies at the 50 most relevant intermediate servers; the other intermediate servers follow the CDV strategy.

where α is generally set equal to 0.2, such as in the default Squid configuration files. The CLV strategy thus simply assigns a longer expiration time to documents that have not been modified for a long time. After the expiration time, the document is removed from the cache. A variation to CLV is CDV. In this case, instead of removing the document at its expiration time, the cache server keeps the document in the cache, but issues an **If-Modified-Since** request to the primary server the next time it is requested, thus pulling in a fresh copy only if necessary. CDV is followed in the Squid cache server [8].

In the SI strategy, intermediate servers still follow a caching policy, but the server promises it will send an invalidation message whenever the document is updated. This strategy has been followed in the AFS distributed file system [34], but has also been used in combination with leases for Web caching [6], [11]. This strategy obviously requires the server to keep track of all copies of its documents.

In the SU family of replication strategies, the primary server chooses the "best" x ASes for which it pushes and maintains copies of a document. Whenever a document is updated, the primary will propagate the new version of the document to the x selected intermediate servers. We define "best" as the ASes where most requests came from in the past. Our traces showed that a relatively small number of ASes were responsible for the bulk of the client requests. For example, 53 percent of all requests for one of our servers came from clients distributed in only 10 ASes out of a total of 1,480 ASes in our traces and 71 percent of the requests could be traced back to no more than 50 ASes. In our experiments, the values chosen for x were 10, 25, and 50.

Finally, we experimented with two *hybrid* strategies. For the top 50 best ASes, we used strategy SU50, while all remaining intermediate servers were configured as cache servers, following strategy CLV and CDV, respectively.

4 SIMULATIONS

We collected traces from two different Web servers: the Vrije Universiteit Amsterdam in The Netherlands (VU Amsterdam) and the Friedrich-Alexander University Erlangen-Nürnberg in Germany (FAU Erlangen). Table 3 shows the general statistics for these two sites. Although we collected traces at other sites as well, they turned out to be too small in terms of number of accesses that we decided to exclude them from further experiments.

We filtered the traces by removing documents that had been requested fewer than 10 times during the months-long tracing period. Simulating replication policies for such documents would not provide any meaningful result nor would these results be useful for predicting which replication policy will be optimal in the near future. The filtering process removed about 67 percent of the documents from the traces, yet only 5 percent of the requests. One can safely associate any replication policy to these documents, but they will hardly have any effect on the total evaluation metrics over the whole document set.

TABLE 3
General Access and Update Statistics for the Chosen Web Sites

Issue	FAU Erlangen	VU Amsterdam
Start date	20/3/2000	13/9/1999
End date	11/9/2000	18/12/1999
Duration (days)	175	96
Number of documents	22,637	33,266
Number of requests	1,599,777	4,858,369
Number of updates	3338	11,612
Number of ASes	1480	2567

4.1 Applying a Global Strategy

In our first experiment, each document was assigned the same strategy. The results are shown in Table 4. Not surprisingly, strategy NR (i.e., no caching or replication) leads to much consumed bandwidth and gives relatively bad performance with respect to the total turnaround time. Likewise, strategy CV, in which the validity of a cached document is always checked with the primary upon each request, leads to high total turnaround times. Improvement is achieved with CLV and CDV at the cost of returning stale documents. When comparing SI to CV, CLV, and CDV, it shows to be best with respect to total turnaround time. Of course, SI cannot return stale documents except in the rare case when a request is sent during the invalidation propagation period. Its performance with respect to consumed bandwidth is approximately the same as the others.

The replication strategies can bring down the total turnaround times, but generally lead to an increase of consumed bandwidth. This increase in bandwidth is caused by the fact that an update may be outdated by a next update before a client issued a request. Combining SU50 with a caching strategy for the remaining intermediate servers improves the total turnaround time, but also leads to returning stale documents.

Table 4 also shows that most strategies are relatively good with respect to one or more metrics, but no strategy is optimal in all cases. In the next section, we discuss the effects if a global strategy is replaced by assigning a strategy to each document separately and show that per-document replication policies lead to better performance with respect to *all* metrics at the same time.

4.2 Applying Per-Document Strategies

Instead of applying the same strategy to all documents, we propose to assign each document its own strategy. By doing so, it becomes possible to obtain good performance with respect to each of the three metrics. Crucial to this approach is the method by which a strategy is assigned to each document separately, referred to as an *assignment method*. In this section, we derive assignment methods that will lead to sets of (*document, strategy*)-pairs that are optimal with respect to overall system performance. We first explain what optimality actually means before discussing our assignment methods.

TABLE 4
Performance Results Using the Same Strategy for All Documents, Measuring the Total Turnaround Time (TaT), the Number of Stale Documents that Were Returned, and the Total Consumed Bandwidth

Strategy	FAU Erlangen			VU Amsterdam		
	TaT (hrs)	# Stale docs	Bandw. (GB)	TaT (hrs)	# Stale docs	Bandw. (GB)
NR	158.4	**0**	16.50	312.6	**0**	114.87
CV	176.6	**0**	15.82	324.3	**0**	**99.76**
CLV	141.8	203	15.82	241.8	136	**99.88**
CDV	141.8	196	15.80	241.8	130	**99.72**
SI	141.7	**0**	15.81	241.1	**0**	**99.72**
SU10	99.4	**0**	**14.00**	273.2	**0**	114.12
SU25	88.9	**0**	17.25	224.1	**0**	118.50
SU50	**79.4**	**0**	23.16	194.9	**0**	131.90
SU50+CLV	**77.9**	35	23.11	**170.4**	43	124.88
SU50+CDV	**77.9**	35	23.11	**170.4**	38	124.86

Optimal and near-optimal values are highlighted for each metric.

4.2.1 Optimal Arrangements

Let D be a set of documents and S a set of strategies for replicating documents. If we assign a specific strategy to each document $d \in D$, we obtain what we refer to as an *arrangement*: a collection of (*document, strategy*)-pairs. Each arrangement will consist of $|D|$ elements. With $|S|$ strategies, there are a total of $|S|^{|D|}$ different arrangements. We denote the set of all possible arrangements as $\mathcal{A}$.

To compare arrangements, we take a look at how well an arrangement performs with respect to various performance metrics. We assume that there are N performance metrics and that each metric is designed such that a lower value indicates better performance. The three performance metrics introduced above (i.e., turnaround time, stale copies delivered, and bandwidth) meet this criterion. Let $s_A(d)$ denote the strategy that is assigned to document d in arrangement A and $res(m_k, d, s_A(d))$ the value in metric m_k that is attained for document d using strategy $s_A(d)$. For each arrangement A, we can then construct the following *result vector* $\mathbf{total}(A)$:

$$\mathbf{total}(A) = \langle total(A)[1], \dots, total(A)[N] \rangle$$

with

$$total(A)[k] = \sum_{d \in D} res(m_k, d, s_A(d)).$$

Note that $total(A)[k]$ is simply the aggregated performance in metric m_k. For example, in our experiments, each $total(A)[k]$ represents the total turnaround time, the total number of returned stale documents, or the total consumed bandwidth on inter-AS links, respectively.

Because we assume that a lower value in a metric always indicates a better performance, using result vectors introduces a partial ordering on the complete set $\mathcal{A}$ of arrangements, such that

$$\mathbf{total}(A_1) < \mathbf{total}(A_2) \quad \text{iff}$$
$$\forall i \in \{1, \dots, N\} : total(A_1)[i] \leq total(A_2)[i] \quad \text{and}$$
$$\exists j \in \{1, \dots, N\} : total(A_1)[j] < total(A_2)[j].$$

Obviously, if $\mathbf{total}(A_1) < \mathbf{total}(A_2)$, then A_1 should be considered to be better than A_2 as it leads to better performance values for each metric.

As an example, consider the results from Table 4 for FAU Erlangen. Let A_{CV} be the arrangement in which each document is assigned strategy CV, A_{CLV} be the arrangement with CLV, and A_{CDV} be the one with CDV for each document. In this case, we have

$$\begin{aligned}\mathbf{total}(A_{CV}) &= \langle 176.6, \quad 0, \quad 15.82 \rangle \\ \mathbf{total}(A_{CLV}) &= \langle 141.8, \quad 203, \quad 15.82 \rangle \\ \mathbf{total}(A_{CDV}) &= \langle 141.8, \quad 196, \quad 15.80 \rangle\end{aligned}$$

and that $\mathbf{total}(A_{CDV}) < \mathbf{total}(A_{CLV})$. In our experiments with traces from FAU Erlangen, it is seen that CDV is indeed better than CLV. However, A_{CV} cannot be ordered with respect to either A_{CLV} or A_{CDV}; A_{CV} is neither better nor worse than either of the other two.

It does not make sense to further consider an arrangement that is outperformed by another arrangement on all metrics. So, for example, choosing A_{CLV} as an arrangement is pointless because A_{CDV} is better than or equal to with respect to all metrics for the set of documents in our experiment.

In general, the collection of all possible arrangements $\mathcal{A}$ has a subset of *optimal arrangements* $\mathcal{A}^*$. Formally, $\mathcal{A}^*$ is defined as

$$\mathcal{A}^* = \{A \in \mathcal{A} | \nexists A' \in \mathcal{A} : \mathbf{total}(A') < \mathbf{total}(A)\}.$$

Our goal is to find assignment methods that will lead to this set of optimal arrangements. One approach to finding $\mathcal{A}^*$ is to use the *brute-force assignment method*. With this method, we simply compute the result vectors for all $|S|^{|D|}$ arrangements in $\mathcal{A}$ and choose the arrangements with the best ones. Of course, this approach is computationally
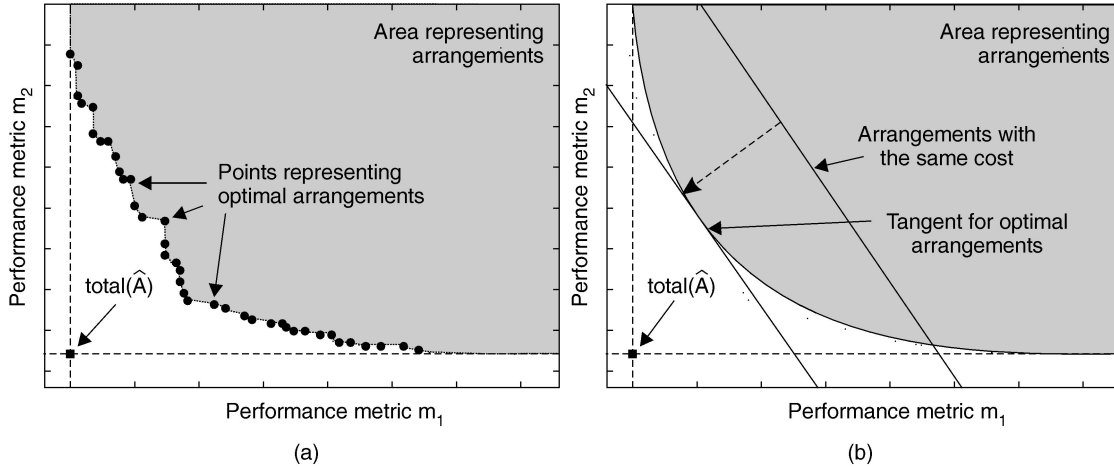
Fig. 4. (a) The area of attainable result vectors for arrangements. (b) Approximating the set of optimal arrangements using tangents.

infeasible. In the following, we derive a much more efficient method.

4.2.2 Optimal Assignment Methods

To simplify our explanation, consider only two performance metrics, m_1 and m_2, for example, turnaround time and bandwidth. In Fig. 4a, the shaded area represents all possible result vectors for arrangements irrespective of the assignment method used. As we explained, this area represents $|S|^{|D|}$ arrangements. Optimal arrangements will be represented by points on the border of this area, as also shown in Fig. 4a. Keep in mind that an optimal arrangement is achieved by the appropriate assignment of strategies to documents.

The area representing arrangements is bounded by the two lines $k_1 = min_i(A_i[1])$ and $k_2 = min_i(A_i[2])$. We define $\hat{A}$ as an *ideal* arrangement that produces the performance values k_1 and k_2, with $\mathbf{total}(\hat{A}) \leq \mathbf{total}(A) \, \forall A \in \mathcal{A}$. Note that, in general, $\hat{A}$ will not exist, that is, there is no assignment of strategies to documents that produces $\mathbf{total}(\hat{A})$. However, we can consider $\mathbf{total}(\hat{A})$ as a best point: It represents the best attainable performance for any possible arrangement.

The question is how can we efficiently find optimal arrangements. If we consider infinitely large sets of documents and, likewise, assume that there are an infinite number of replication strategies to select from, the set of optimal arrangements can be approximated by a continuous and convex curve, as shown in Fig. 4b. Each point on this curve represents an optimal arrangement. If we can devise a method for finding a point on this curve, we argue that such a method can also be used to find an optimal arrangement for a finite set of documents and a finite set of strategies. In the following, we first devise a method for finding points on the curve and then transform that method so that it can be applied to finite sets of arrangements.

Our first concern is to find points on the curve. To do so, we construct tangents. With two performance metrics, m_1 and m_2, a tangent can be represented by a straight line

$$a \cdot m_1 + b \cdot m_2 = C$$

which has a specific slope for any given constant C. The tangent has the property that it intersects the curve at exactly one point (which corresponds to an optimal arrangement). In addition, a tangent for a convex curve has the property that for any constant $C' < C$, the line

$$a \cdot m_1 + b \cdot m_2 = C'$$

will *not* intersect it.

Tangents can be constructed as follows: We take a look at linear combinations of the values in each performance metric. In particular, we can associate a *total cost* with each arrangement A by considering a weight vector $\mathbf{w} = \langle w[1], \ldots, w[N] \rangle$ with $\sum_{k=1}^{N} w[k] = 1$ and $\forall k : w[k] \geq 0$, leading to:

$$cost_{\mathbf{w}}(A) = \sum_{k=1}^{N} w[k] \cdot total(A)[k].$$

With $N = 2$, each weight vector $\mathbf{w}$ is uniquely associated with a specific slope. It is important to note that, although we associate a specific cost with each arrangement, it is senseless to compare two costs if they are computed with different weight vectors. In fact, it may be hard to interpret a cost meaningfully as we are adding noncommensurate values. Costs are used only for comparing arrangements, provided they are computed using the same weight vector.

Each weight vector leads to a collection of parallel lines, each line representing arrangements that have the same cost under $\mathbf{w}$ as shown in Fig. 4b. We then need to find the arrangement A for which $cost_{\mathbf{w}}(A)$ is minimal. This arrangement will be optimal for $\mathbf{w}$ and will lie on the curve forming the border of the area representing arrangements, as shown in Fig. 4b. As a consequence, the curve itself can be expressed as the set $\mathcal{A}^*$ (recall that we are still considering infinitely large sets of arrangements):

$$\mathcal{A}^* = \bigcup_{\mathbf{w}} \{A \in \mathcal{A} | \forall A' \in \mathcal{A} : cost_{\mathbf{w}}(A) \leq cost_{\mathbf{w}}(A')\}.$$

The following observation is important. Constructing a tangent will lead to finding an optimal arrangement. To construct a tangent, we are looking for a method that, for a
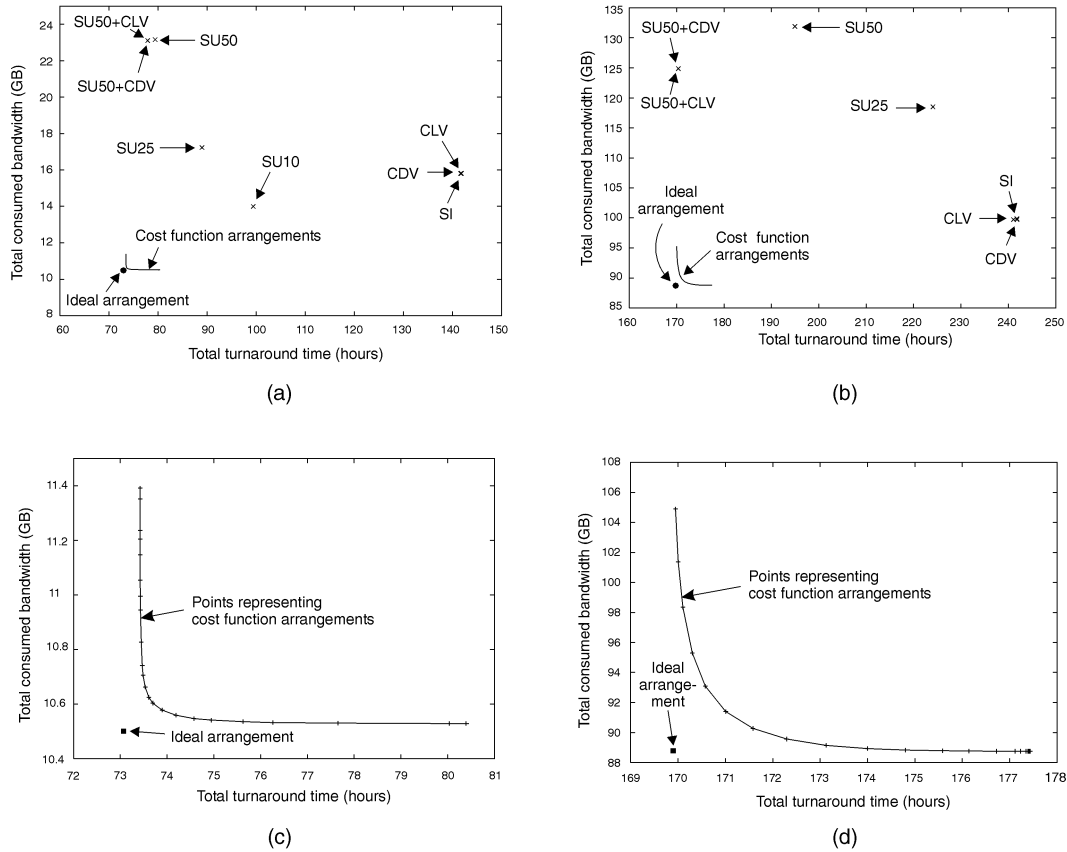
Fig. 5. Performance of arrangements vs. global strategies. (a) FAU Erlangen: complete view. (b) VU Amsterdam: complete view. (c) FAU Erlangen: detailed view. (d) Vrije Universiteit: detailed view.

given weight vector **w** and an infinitely large set of arrangements $\mathcal{A}$, will minimize $cost_{\mathbf{w}}(A)$. This method can also be used for finding an optimal arrangement in a finite set of arrangements.

A method that accomplishes this is the one that assigns to each document d a strategy s from the complete set of strategies S, for which $\sum_{k=1}^{N} w[k] \cdot res(m_k, d, s)$ is minimal. This assignment method will indeed lead to an optimal arrangement A^* for the given weight vector **w**, which can be seen as follows:

$$cost_{\mathbf{w}}(A^*) = \min_{A \in \mathcal{A}}[cost_{\mathbf{w}}(A)]$$

$$= \min_{A \in \mathcal{A}}\left[\sum_{k=1}^{N} w[k] \cdot total(A)[k]\right]$$

$$= \min_{A \in \mathcal{A}}\left[\sum_{k=1}^{N} w[k] \cdot \left(\sum_{d \in A} res(m_k, d, s_A(d))\right)\right]$$

$$= \min_{A \in \mathcal{A}}\left[\sum_{d \in A} \sum_{k=1}^{N} w[k] \cdot res(m_k, d, s_A(d))\right]$$

$$\geq \min_{A \in \mathcal{A}}\left[\sum_{d \in A}\left(\min_{s \in S}\left[\sum_{k=1}^{N} w[k] \cdot res(m_k, d, s)\right]\right)\right].$$

We refer to arrangements resulting from these assignment methods as *cost function arrangements* as they directly take the total cost of a document in terms of performance into account. Note that to compute an optimal arrangement for a given weight vector requires at most $|D| \cdot |S|$ computations.

This is a considerable improvement over the brute-force assignment method discussed earlier.

4.3 Results

To simplify matters, we decided to make optimization of consistency a major requirement by considering cost function arrangements with a large weight for the number of stale documents returned. In other words, we looked at models that would implement strong consistency. By subsequently modifying the relative weights of total turnaround time and total consumed bandwidth, we obtain optimal arrangements that implement various turnaround/bandwidth trade-offs.

Fig. 5a shows the performance of arrangements in terms of total turnaround time and consumed bandwidth for the data collected at FAU Erlangen. Each point on the curve corresponds to a cost function arrangement over the set of documents with one particular set of weights. Comparable results for the VU Amsterdam are shown in Fig. 5b.

We compare each arrangement with the ideal arrangement $\hat{A}$ discussed above. This point corresponds to the best achievable total turnaround time (obtained by selecting for each document the strategy with the smallest turnaround time) and the best achievable bandwidth usage (obtained by selecting for each document the strategy with the smallest consumed bandwidth).

Fig. 5c and Fig. 5d show a detailed view of the cost function arrangements. Each point on the curve represents an optimal arrangement. Note that none of the global

TABLE 5
Number of Documents to Which a Specific Strategy Is Assigned

Strategy	FAU Erlangen	VU Amsterdam
NR	1351	3239
CV	0	2
CLV	66	437
CDV	0	0
SI	137	295
SU10	13,779	4396
SU25	3243	5371
SU50	3374	15,369
SU50+CLV	682	4099
SU50+CDV	5	58

strategies comes close to the point representing the ideal arrangement and do not even fall in the graphs shown in Fig. 5c and Fig. 5d. However, all cost function arrangements are close to the target if we compare them to any global strategy. In other words, selecting replication strategies on a per-document basis provides a performance improvement over any global strategy that we considered in our experiments.

To further substantiate our claim that differentiating strategies makes sense, we considered a specific cost function arrangement to see which strategies were actually assigned to documents. The results are shown in Table 5. These results show that it indeed makes sense to apply several strategies, although neither CV or CDV are used often. The latter is in line with the research results reported in [6].

5 SUPPORTING ADAPTIVE DISTRIBUTED DOCUMENTS

Up to this point, we have provided arguments to use per-document replication strategies instead of applying a global, system-wide strategy. An important question is how assigning strategies can be put to practical use. In this section, we discuss and evaluate an architecture for documents that can dynamically select a strategy using real-time trace-driven simulations. We show that periodically running the simulation experiments described above using fresh traces allows a server to dynamically replace a document's strategy while incurring only a little overhead.

5.1 System Architecture

To support adaptive replication, we consider a collection of servers hosting Web documents, as described in Section 3.1. A document's primary server is responsible for evaluating and possibly replacing the replication strategy currently assigned to a document. Evaluating a strategy is done by taking traces collected over the most recent time period ΔT extracted from a local log. The secondary servers for a document also collect traces, which they regularly send to the primary, as shown in Fig. 6.

The primary reevaluates its choice of replication strategy by looking at the document's most recent trace data and simulating several alternative strategies, as described in the previous section. The primary informs the secondary servers when it chooses a new strategy. Note that not all secondaries may be registered at the primary; Cache servers are not registered, for example. Such servers continue to use the previous strategy, but are informed when contacting the primary the next time. This scheme allows for gradual dissemination of a newly selected strategy.

Servers holding a copy of a given document must collect traces of their activity for two reasons. First, we believe that the owner of a document (i.e., the primary) should be allowed to obtain logs of every request, independently of which server handled the request. This may encourage the use of caching and replication also for sites whose revenue depend on their popularity [31]. Second, access traces must be centralized at the primary to enable selection of a replication strategy.

Each secondary keeps a log of the requests it receives. Periodically, new log entries are sent to the primary. Sending a log entry to the primary is delayed at most, say, 10 minutes, which guarantees that the primary's view of the logs is at most 10 minutes late. We estimate that this limit is adequate for piggybacking log entries while allowing responsive adaptations of the current strategy as
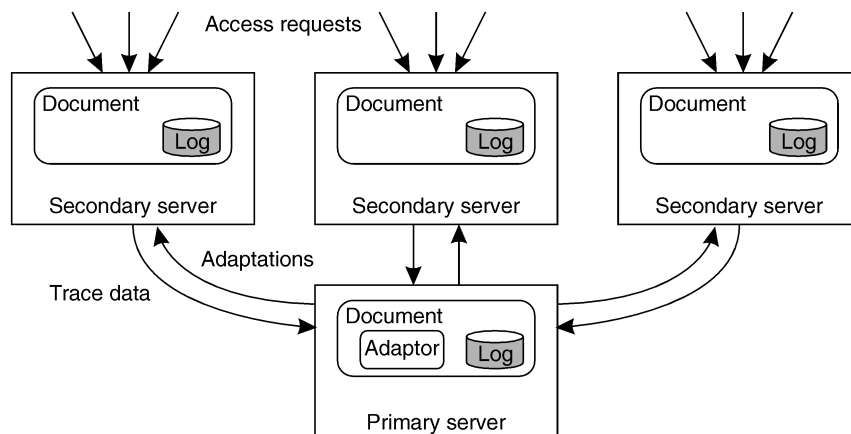

Fig. 6. An architecture for supporting distributed documents.

access patterns change. However, secondaries can send traces more often if they wish, for example, to reclaim storage space in their logs.

The primary writes the traces it receives to disk immediately. Since it can receive traces from multiple secondaries, the set of entries in the primary's log file is not sorted in chronological order. Therefore, when a reevaluation takes place, the primary log file is first sorted into a proper trace file that can be used for simulation.

5.2 Deciding When to Adapt

An important issue is when the primary should decide to reevaluate a document's strategy. Doing so too often would waste computing resources, while reevaluating too rarely would decrease overall system performance. The simplest scheme for adaptation is to reevaluate the replication strategies at fixed time intervals, such as once a week. However, this approach does not allow a document to react quickly to sudden changes in access or update patterns. It would be more efficient to adapt as soon as such patterns change.

To detect these changes, each server monitors a number of variables, such as frequency of requests and average response time. Significant variation in one of these variables indicates a change that may warrant replacing the current strategy. Variables are computed using a standard technique. When a copy is created, a server initializes each variable V to a value derived from a first sample. Each time a new sample S is taken, V is updated using an aging algorithm:

$$V := \omega \cdot S + (1 - \omega)V,$$

where the value of ω controls the relative weight given to a new sample with respect to the previous sequence of samples.

Each time an adaptation takes place, low and high watermarks are set up for each variable. If the value of V ever reaches one of these watermarks, we assume that the access or update pattern may have changed enough for the current strategy not to be optimal any more so that a reevaluation should take place.

A problem that must be solved is where to monitor the variables. One possibility is that the primary server of a document does all the necessary computations. However, this would not be very practical since variables can be computed only from the trace data sent by copies. Since the primary receives traces out of order, computing a sequential history of a variable would become quite complex. Therefore, each secondary computes variables locally and transmits its value to the primary, together with the trace data. The primary does not compute a single value, but keeps the variables separate.

The primary monitors all the variables received from the document's secondaries. Because many of the variables account only for a small fraction of the overall traffic, one variable reaching its watermark does not necessarily mean that a significant change is occurring. On the other hand, if several variables reach their watermarks within a small time interval, it is likely that a real change in the access patterns has occurred.

TABLE 6
Profiling Distributed Documents

Operation	Execution time	
	Primary	Secondary
Network I/O	49%	48%
Document Delivery	24%	28%
Logging	15%	9%
Replication Policy	12%	15%

Sometimes, a secondary cannot wait for the primary to reevaluate strategies. For example, during a *flash crowd*, there is a sudden and significant increase in the number of requests received for a specific document. In such cases, the load increase on a secondary may deteriorate not only the turnaround time of requests for the document, but also that of every other document hosted by the same server. This performance degradation is clearly not acceptable.

When a secondary server is receiving so many requests that its performance is being degraded, it can decide to adapt by itself without requiring the document's primary to reevaluate strategies. This approach allows fast responsiveness in case of a flash crowd. The reaction consists of creating more copies at other servers to handle the load. Although sharing the load among several servers may solve the overload problem, such a trivial adaptation is not likely to be optimal. Therefore, an alarm message is sent to the primary requesting it to immediately reevaluate the overall replication strategy.

5.3 Incurred Overhead Costs

Compared to traditional Web documents, distributed documents need to perform additional operations such as logging trace data, sending traces to the primary, and, of course, regularly running simulations to evaluate strategies. As we show in this section, the extra costs incurred by these operations is small compared to the performance improvements that per-document replication strategies provide.

5.3.1 Overhead Due to Collecting Traces

Collecting traces consists of logging data and sending traces to the primary. To evaluate the overhead incurred by merely collecting traces, we built a small prototype system for distributed documents. This prototype was used to replay the trace files collected for the various sites mentioned in Section 4. We profiled each process to get an idea of how much time is spent for various tasks.

Table 6 shows the time that the program spends in its different modules. Network I/O operations account for most of the computation. Logging adds up to 9 percent of the processing time at each secondary. As the primary must log the requests that are addressed to it as well as the trace data sent by secondaries, it requires more time, up to 15 percent. Although our approach introduces some additional overhead, we argue that the extra costs are acceptable compared to the benefits of adaptive replication policies. Each log entry contains an average 12 requests, adding to 25 bytes of data per logged request. However, one can expect that this figure is highly dependent on the number of requests that the document receives every 10 minutes.
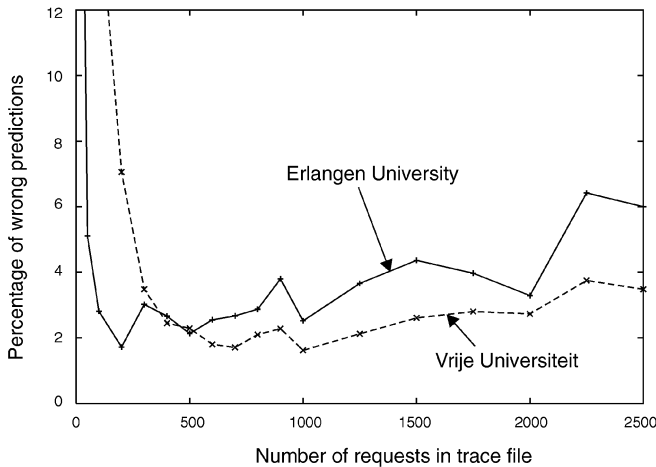
Fig. 7. Percentage of incorrect predictions as the chunk size varies.

These figures have been obtained with a somewhat naive implementation of the log collection: In the current prototype, each document copy collects traces and sends them to its primary in isolation from all other documents; grouping log data from several documents on the same site into a single message would allow for a much better use of network resources.

5.3.2 Balancing Simulation Overhead vs. Prediction Accuracy

Adapting the replication strategy of a document requires running as many simulations as there are candidate strategies. Simulations are trace driven, which means that they execute roughly in linear time compared to the number of requests in the trace. From this perspective, traces should be kept as small as possible to save computing resources. However, short traces may not reliably represent the current access pattern of a document. On the other hand, very long traces may be unsuitable for *predicting* the next best strategy. This can easily be seen if we assume that changes in access patterns occur in phases. In that case, a very long trace will span multiple phases, often making it much harder to use as a predictor for a next phase. Therefore, we need to find a trade-off between trace size and accuracy, while at the same time ensuring that traces have an appropriate maximum length.

To evaluate the accuracy of using traces to predict the next best strategy, we used the same traces as described in Section 4. We selected only those documents that received at least 5,000 requests, leading to a sample size of 98 documents for the VU Amsterdam and 30 documents for FAU Erlangen. For each of these documents, we split the trace into successive chunks of N requests each. We simulated each of the trace chunks with different replication strategies. If the "best" policy of chunk n is the same as the "best" policy of chunk $n + 1$, then the prediction made at time n is assumed to have been correct.

Fig. 7 shows the incorrect predictions when the chunk size varies. We can see that short trace files lead to many incorrect predictions. However, as the chunk size grows, the proportion of error decreases to approximately 2 percent at 500 requests, after which it gradually increases again.

In our case, we conclude that a reasonable chunk size is something like 500 requests. Note that the irregular shape of the FAU Erlangen traces is most likely caused by the relatively small sample size of 30 documents. We measured the computing time required by simulations on a 600 MHz Pentium-III workstation. Each simulated request took about 28 μs. So, for example, simulating a 500-request trace over 10 different configurations takes about 140 ms of CPU time.

5.4 Organization of a Distributed Document

To integrate adaptive distributed documents into the World Wide Web, we are developing the Globule platform [30]. In this system, Web servers cooperate with each other to replicate documents among them and to transparently direct client requests to the "best" replica. Globule is implemented as a module for Apache, so turning normal Web documents into adaptive replicated documents only requires compiling an extra module into an existing Web server. Its internal organization is derived from Globe distributed shared objects [37].

Fig. 8 shows the general organization of a distributed document. Each replica is made of two separate local subobjects: a document's content, which is available in the form of delivery subobject capable of producing documents, and a replication subobject, which is responsible for enforcing the document's replication policy.

Incoming requests are first intercepted by the redirection subobject before actually reaching the document. This subobject figures out which replica is preferable for treating the incoming request and directs the client to this replica. This redirection can be implemented via basic HTTP redirection or by more sophisticated mechanisms such as DNS redirection [22]. Selecting an optimal replica for each client is considered an orthogonal problem and is not addressed in this article.

Requests are then intercepted by the replication subobject, whose role is to achieve document replication according to the document's replication policy. Once the replication subobject has authorized the request, the Web server uses one of its standard document delivery modules to respond. These can be modules that deliver static documents or modules that generate a document on request.

Although this architecture may seem overly complex for static documents, it also supports replication of dynamic documents. By merely replacing the document delivery module, all mechanisms such as replication and adaptation can be applied without modification.

Each replication subobject is internally organized as shown in Fig. 9. The *policy subobject* implements one specific strategy, such as the ones described in Section 3. It maintains information about the consistency of the copy, such as the date of last modification and the date of the last consistency check. Each time a request is issued, the *protocol subobject* first transmits the characteristics of the request to the policy subobject. Based on its implementation, the policy subobject responds by indicating how to treat the request: answer immediately, check the validity before responding (i.e., send an If-Modified-Since request to the primary server), etc. The protocol subobject is in charge of actually performing the operation. The protocol subobject
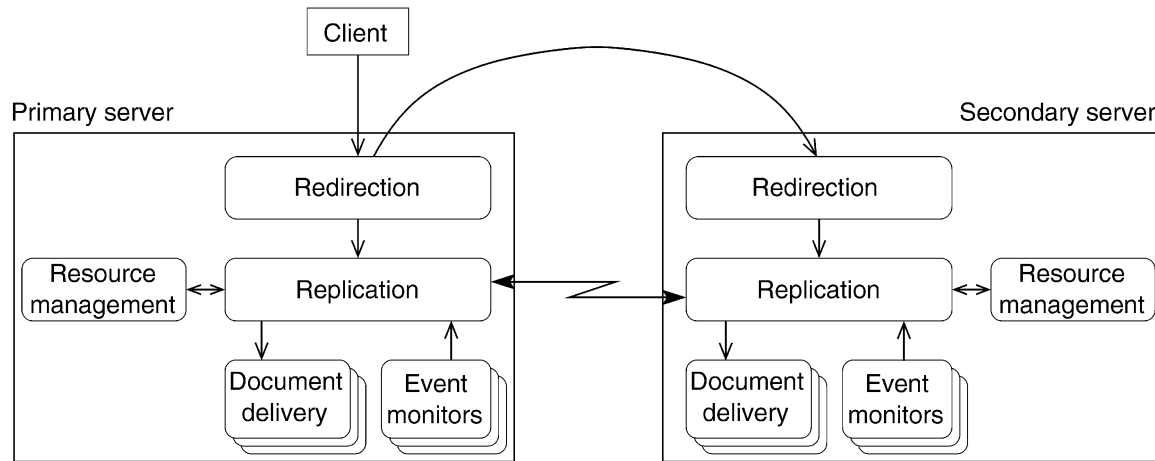
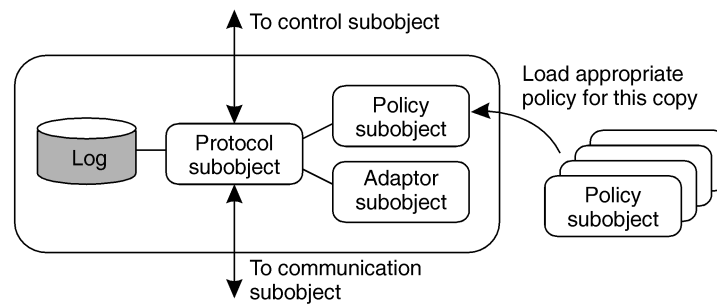Fig. 8. Organization of an adaptive replicated document.

Fig. 9. The internal organization of a replication subobject.

can also directly receive incoming network messages, such as a notification that the document has been updated.

The protocol subobject is in charge of collecting log data and transmitting them to the primary. It also transmits the monitoring variables to the *adaptor subobject*. The latter implements the adaptor component described above (see Fig. 6). The adaptor subobject decides whether an adaptation should take place. If so, it sorts the most recently received traces, runs simulations, and informs the protocol subobject of the new optimal strategy. The protocol subobject is then responsible for replacing the policy subobject.

Although adaptations take place at the primary, each secondary also has an adaptor subobject. This adaptor is used only to detect flash crowds and create new replicas to handle the sudden load. As we mentioned, whenever a secondary takes such a decision, it immediately requests that the primary reevaluate the overall strategy.

6 CONCLUSIONS

Based on our experiments, we argue that it makes sense to look for solutions that allow assigning a replication strategy to *individual* Web documents. In the approach described in this paper, it turns out that using trace-driven real-time simulations can be used for dynamically adapting a strategy.

Our approach does require us to consider documents as objects instead of data. This allows the encapsulation of replication strategies *inside* each document. Of course, this is a fundamental change that prevents current Web servers

and caching proxies from hosting distributed documents. A new platform is necessary. We are building it as a module that makes Apache servers cooperate to replicate documents [30]. Nonstandard protocols can be confined to interserver communication, while clients access documents using standard protocols without further modification to their software. Doing this will provide users with distributed documents in a transparent manner.

However, as more strategies are introduced and will thus need to be evaluated, our approach may possibly introduce performance problems. A solution can be sought in combining methods for adaptive protocols from the same family with replacement techniques for switching between different families of protocols using real-time simulations. Our future work concentrates on further developing our architecture and its implementation and seeking solutions for efficient adaptations within and between families of consistency protocols.

REFERENCES

[1] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel, "Scalable Content-Aware Request Distribution in Cluster-Based Network Servers," *Proc. Usenix Ann. Technical Conf.,* pp. 323-336, June 2000.

[2] H.E. Bal, R. Bhoedjang, R. Hofman, C. Jacobs, K. Langendoen, T. Ruhl, and F. Kaashoek, "Performance Evaluation of the Orca Shared Object System," *ACM Trans. Computer Systems,* vol. 16, no. 1, pp. 1-40, 1998.

[3] G. Banga, F. Douglis, and M. Rabinovich, "Optimistic Deltas for WWW Latency Reduction," *Proc. Usenix Ann. Technical Conf.,* pp. 289-304, Jan. 1997.

[4] T. Bates, E. Gerich, L. Joncheray, J.-M. Jouanigot, D. Karrenberg, M. Terpstra, and J. Yu, "Representation of IP Routing Policies in a Routing Registry," RFC 1786, May 1995.

[5] G. Brun-Cottan and M. Makpangou, "Adaptable Replicated Objects in Distributed Environments," Technical Report 2593, INRIA, May 1995.

[6] P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the World Wide Web," *IEEE Trans. Computers,* vol. 47, no. 4, pp. 445-457, Apr. 1998

[7] V. Cate, "Alex—A Global File System," *Proc. File Systems Workshop,* pp. 1-11, May 1992.

[8] A. Chankhunthod, P. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell, "A Hierarchical Internet Object Cache," *Proc. Usenix Ann. Technical Conf.,* pp. 153-163, Jan. 1996.

[9] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic Algorithms for Replicated Data Management," *Proc. Sixth Symp. Principles of Distributed Computing,* pp. 1-12, Aug. 1987.

[10] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy, "Adaptive Push-Pull: Disseminating Dynamic Web Data," *Proc. 10th Int'l WWW Conf.,* pp. 265-274, May 2001.

[11] V. Duvvuri, P. Shenoy, and R. Tewari, "Adaptive Leases: A Strong Consistency Mechanism for the World Wide Web," *Proc. 19th INFOCOM Conf.,* pp. 834-843, Mar. 2000.

[12] A. Fox, S.D. Gribble, Y. Chawathe, E.A. Brewer, and P. Gauthier, "Cluster-Based Scalable Network Services," *Proc. 16th Symp. Operating System Principles,* pp. 78-91, Oct. 1997.

[13] J.S. Gwertzman and M. Seltzer, "The Case for Geographical Push-Caching," *Proc. Fifth Workshop Hot Topics in Operating Systems,* pp. 51-55, May 1996.

[14] H. Innoue, K. Kanchanasut, and S. Yamaguchi, "An Adaptive WWW Cache Mechanism in the A13 Network," *Proc. INET '97,* June 1997.

[15] S. Jin and A. Bestavros, "GreedyDual* Web Caching Algorithm: Exploiting the Two Sources of Temporal Locality in Web Request Streams," *Computer Comm.,* vol. 24, no. 2, pp. 174-183, 2001.

[16] J. Kleinöder and M. Golm, "Transparent and Adaptable Object Replication Using a Reflective Java," Technical Report TR-14-96-07, Computer Science Dept., Univ. of Erlangen-Nürnberg, Sept. 1996.

[17] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek, "The Click Modular Router," *ACM Trans. Computer Systems,* vol. 18, no. 3, pp. 263-297, 2000.

[18] B. Krishnamurthy and J. Wang, "On Network-Aware Clustering of Web Clients," *Proc. SIGCOMM Conf.,* pp. 97-110, Aug. 2000.

[19] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat, "Providing Availability Using Lazy Replication," *ACM Trans. Computer Systems,* vol. 10, no. 4, pp. 360-391, 1992.

[20] F.T. Leighton and D.M. Lewin, "Global Hosting System," US Patent No. 6,108,703, Aug. 2000.

[21] E.P. Markatos and C.E. Chronaki, "A Top 10 Approach for Prefetching the Web," *Proc. INET '98,* July 1998.

[22] P.R. McManus, *A Passive System for Server Selection within Mirrored Resource Environments Using AS Path Length Heuristics,* 1999.

[23] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd, and V. Jacobson, "Adaptive Web Caching: Towards a New Global Caching Architecture," *Computer Networks & ISDN Systems,* vol. 30, nos. 22-23, pp. 2169-2177, 1998.

[24] B.C. Neuman, "Scale in Distributed Systems," *Readings in Distributed Computing Systems,* T. Casavant and M. Singhal, eds., pp. 463-489, Los Alamitos, Calif.: IEEE CS Press, 1994.

[25] S.W. O'Malley and L.L. Peterson, "A Dynamic Network Architecture," *ACM Trans. Computer Systems,* vol. 10, no. 2, pp. 110-143, 1992.

[26] V.N. Padmanabhan and J.C. Mogul, "Using Predictive Prefetching to Improve World-Wide Web Latency," *Proc. SIGCOMM '96,* July 1996.

[27] S. Patarin and M. Makpangou, "Pandora: A Flexible Network Monitoring Platform," *Proc. Usenix Ann. Technical Conf.,* June 2000.

[28] G. Pierre, I. Kuz, M. van Steen, and A.S. Tanenbaum, "Differentiated Strategies for Replicating Web Documents," *Computer Comm.,* vol. 24, no. 2, pp. 232-240, 2001.

[29] G. Pierre and M. Makpangou, "Saperlipopette!: A Distributed Web Caching Systems Evaluation Tool," *Proc. 1998 Middleware Conf.,* pp. 389-405, Sept. 1998.

[30] G. Pierre and M. van Steen, "Globule: A Platform for Self-Replicating Web Documents," *Proc. Sixth Int'l Conf. Protocols for Multimedia Systems,* pp. 1-11, Oct. 2001.

[31] J.E. Pitkow, "In Search of Reliable Usage Data on the WWW," *Proc. Sixth Int'l WWW Conf.,* Apr. 1997.

[32] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal, "A Dynamic Object Replication and Migration Protocol for an Internet Hosting Service," *Proc. 19th Int'l Conf. Distributed Computing Systems,* pp. 101-113, June 1999.

[33] Y. Saito, "Optimistic Replication Algorithms," technical report, Univ. of Washington, Aug. 2000.

[34] M. Satyanarayanan, "Scalable, Secure, and Highly Available Distributed File Access," *Computer,* vol. 23, no. 5, pp. 9-21, May 1990.

[35] B. Urgaonkar, A. Ninan, M. Raunak, P. Shenoy, and K. Ramamritham, "Maintaining Mutual Consistency for Cached Web Objects," *Proc. 21st Int'l Conf. Distributed Computing Systems,* Apr. 2001.

[36] R. van Renesse, K. Birman, and S. Maffeis, "Horus: A Flexible Group Communication System," *Comm. ACM,* vol. 39, no. 4, pp. 76-83, 1996.

[37] M. van Steen, P. Homburg, and A.S. Tanenbaum, "Globe: A Wide-Area Distributed System," *IEEE Concurrency,* vol. 7, no. 1, pp. 70-78, 1999.

[38] J. Whitehead and Y.Y. Goland, "WebDAV: A Network Protocol for Remote Collaborative Authoring on the Web," *Proc. Sixth European Conf. Computer Supported Cooperative Work,* pp. 291-310, Sept. 1999.

[39] O. Wolfson, S. Jajodia, and Y. Huang, "An Adaptive Data Replication Algorithm," *ACM Trans. Database Systems,* vol. 22, no. 4, pp. 255-314, 1997.

[40] H. Yu and A. Vahdat, "Design and Evaluation of a Continuous Consistency Model for Replicated Services," *Proc. Fourth Symp. Operating System Design and Implementation,* Oct. 2000.

Guillaume Pierre holds the engineer degree from the "Institut d'Informatique d'Entreprise" and the PhD degree in computer science from INRIA and the University of Evry-val d'Essonne. He is a postdoctoral researcher at the Vrije Universiteit in Amsterdam. His main interests are flexible computing systems and Internet performance optimization.

Maarten van Steen received the MSc degree in applied mathematics from Twente University (1983) and the PhD degree in computer science from Leiden University (1988). He is currently a professor at the Vrije Universiteit in Amsterdam. He worked at an industrial research laboratory for several years before returning to academia. His research interests include operating systems, computer networks, (wide-area) distributed systems, and Web-based systems. He is a member of the IEEE and the ACM.

Andrew S. Tanenbaum holds the SB degree from the Massachusetts Institute of Technology and the PhD degree from the University of California at Berkeley. He is currently a professor of computer science at the Vrije Universiteit in Amsterdam and dean of the interuniversity computer science graduate school, ASCI. Professor Tanenbaum is the principal designer of three operating systems: TSS-11, Amoeba, and MINIX. He was also the chief designer of the Amsterdam Compiler Kit. In addition, he is the author of five books and more than 80 refereed papers. He is a fellow of the ACM, a fellow of the IEEE, and a member of the Royal Dutch Academy of Sciences. In 1994, he was the recipient of the ACM Karl V. Karlstrom Outstanding Educator Award and, in 1997, he won the SIGCSE award for contributions to computer science.

▷ **For more information on this or any computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.