Differentiated Strategies for Replicating Web Documents

Guillaume Pierre * Ihor Kuz **
Maarten van Steen * Andrew S. Tanenbaum *

* Vrije Universiteit, Division of Mathematics and Computer Science,
de Boelelaan 1081, 1081 HV Amsterdam, The Netherlands.
** Faculty of Information Technology and Systems, T.U. Delft,
Zuidplantsoen 4, P.O. Box 356, 2628 BZ Delft, The Netherlands.

Abstract

Replicating Web documents reduces user-perceived delays and wide-area network traffic. Numerous caching and replication protocols have been proposed to manage such replication while keeping the document copies consistent. We claim, however, that no single caching or replication policy can efficiently manage all documents. Instead, we propose that each document be replicated with a policy specifically tailored to it. We have collected traces on our university's Web server and conducted simulations to determine the performance such tailored policies would produce, as opposed to using the same policy for all documents. The results show a significant performance improvement with respect to end-user delays, wide-area network traffic and document consistency. We also present how these results can be used to build adaptive replicated Web documents, capable of automatically selecting the policy that best suits them.

1 Introduction

Every Web user has experienced slow document transfers. To reduce the access time, one possible solution is to replicate the documents. This balances the load among the servers and prevents repetitive transfers of documents over the same network links. However, after a document is updated, users should not access stale data; the replicated copies should be either destroyed or updated.

There are numerous protocols available to help achieve such consistency. In this paper, we assume that all updates to a document happen at the same location, which we call the *master*; the other locations (the replicas) are called *slaves*. Consistency policies for replicating Web documents generally fall into the "pull" or the "push" categories. Push strategies require the master (or the server hosting it) to keep track of all slaves, and to contact each slave when the document is updated. In such cases, it is possible to multicast the new version, or to request stale copies to be destroyed. Pull strategies require that slaves check the master to detect updates. Strategies differ in when to check for consistency: it can be done periodically or each time a copy is read. A commonly used variant is for a copy to destroy itself when it suspects it is out-of-date without even checking the master.

Another classification of replication strategies can be done regarding *replicas* and *caches*. A replica site always holds the document; a cache site may or may not hold it. Replica sites are sometimes called mirrors.

Which replication strategy is the best suited for Web documents? This is a difficult question, and much research is being done to answer it. The main obstacle to a good solution is the heterogeneity of documents. For example, document sizes, popularity, the geographical location of clients and the frequency of updates vary greatly from one document to another [19]. Most approaches try to find replication strategies that can deal with such diverse characteristics.

In this paper we take a different point of view. We claim that no single policy can be good enough in all cases. So, instead of designing some kind of "universal policy," we argue that several specialized policies should be used simultaneously. Depending on its characteristics, each document should be replicated using the best-suited policy for that particular document.

We will not discuss here how multiple replication strategies can be supported and integrated in the current World-Wide Web. This issue has already been addressed in papers about GlobeDoc [24]. Using

GlobeDoc, Web pages (or groups of pages) are encapsulated into distributed objects. This encapsulation allows one to easily associate an object with any replication policy [23]. A specialized proxy can then act as a gateway between the HTTP protocol used by the browsers, and the distributed-object protocols used by the documents.

Although the mechanisms for associating custom replication policies with Web documents are up and running, the need for differentiated strategies has not been addressed so far. The aim of this paper is not to present a complete replication system, but rather to demonstrate that it makes sense to differentiate replication strategies. To do so, we monitored our university's Web server by keeping track of client requests as well as document updates. Then, for each document in these traces, we simulated how it would have behaved if replicated by one of several policies. We compared the resulting performance of "one-size-fits-all" strategies with custom strategies. In the first case, all documents were replicated using the same strategy; in the second case we chose the "best" strategy for each document, based on a perfect knowledge of future requests. The results show that custom strategies provide a clear performance improvement compared to any one-size-fits-all strategy.

Demonstrating that differentiating replication strategies makes sense does not solve the issue of how to find the best strategy for an individual document. This paper also addresses the question whether past access patterns can be used to predict which replication policy will be optimal in the near future. We propose using *adaptive* Web documents that are capable of selecting their own optimal policy. To that end, each adaptive document is implemented as a distributed object encapsulating its state as well as past access data for deciding on the best replication policy. The document adapts its current policy as appropriate.

We have simulated adaptive documents. The results show that the predictions are accurate in most cases. However, our traces do not contain fast traffic pattern changes commonly known as "flash crowds." Further work is needed on this subject.

This paper is organized as follows: Section 2 describes the configurations we worked with; Section 3 describes our experimental setup; Section 4 discusses the methods we designed for associating optimal strategies to documents and presents the simulation results; Section 5 presents related work. We conclude in Section 6.

2 System Configurations

In our experiment, the documents were hosted by one particular server (the Web server of our computer science department). Clients located worldwide retrieved the documents from the server, or from intermediate servers (caches or replicas). We investigated the effect of interposing replication protocols between the server and the intermediate servers on the quality of service perceived by the users.

2.1 System Model

2.1.1 Document Model

We assume that all document are updated at the main server. The only requirement is that the server can detect such updates in order to propagate appropriate information to the copies (if the replication policy needs it).

Although dynamic documents such as CGIs, ASPs and PHPs are easily embeddable in GlobeDocs, the simulation of replication for such documents is not straightforward. To simplify our experiments, we considered only static documents.

2.1.2 Placement of Intermediate Servers

To reliably simulate replication strategies, we first had to figure out how many document copies are necessary and to decide which client will use which copy. The extent to which this choice is actually realistic strongly determines the validity of the final results. Therefore, we wanted to take the actual network topology into account in order to let adjacent clients share a copy, minimize bandwidth, and so on.

We decided to group the clients based on the autonomous systems that hosted them. Autonomous systems (or *ASes*) are used to achieve efficient worldwide routing of IP packets [4]. Each AS is a group of nodes interconnected by network links. Its managers are responsible for routing inside their domain. They export only information about their relations to other ASes, such as which ASes they can receive packets from, and which ASes they can send packets to. Worldwide routing algorithms use this information to determine the optimal route between two arbitrary machines on the Internet.

An interesting feature of ASes from our point of view is that they generally consist of relatively large groups of hosts close to each other with respect to the network topology.[1] Therefore, we can assume that

[1]Note that host proximity in terms of network topology does not imply geographical proximity. For example, all of

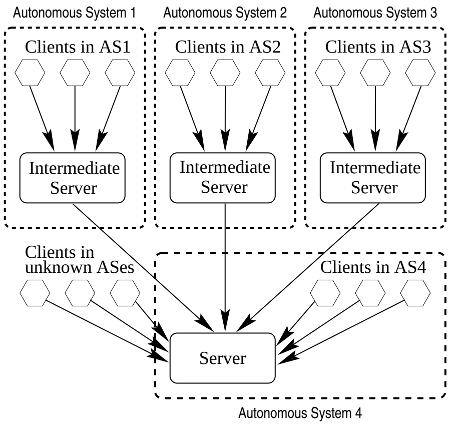

Figure 1: System model

the network connection performance is much better inside an AS than between two ASes.

This led us to decide to place at most one intermediate server (cache or replica) per AS, and to bind all users to their AS's intermediate server (see Figure 1). This rule has two exceptions. First, it would be pointless to create an intermediate server in the same AS as the master server: clients located in this AS can directly access the master as well. Second, we decided that the few clients for which we could not determine the AS also access the server directly.

2.2 Configurations

For each document, we consider a number of setups likely to optimize the access to that document. All configurations are based on the same system model; the only difference between them is the nature of the intermediate servers and the consistency policy they use.

The first configuration acts as a baseline configuration:

NoRepl: This configuration uses no caching or replication whatsoever. All clients contact the server directly, without any intermediate servers.

2.2.1 Caching Configurations

Caching configurations use proxy caches in place of intermediate servers. We have considered configurations where the caches use the following policies:

Check: When a cache hit occurs, the cache systematically checks the copy's consistency by sending an `If-Modified-Since` request to the master before answering the client's request.

Alex: When a copy is created, it is given a time-to-live proportional to the time elapsed since its last modification [7]. Before the expiration of the time-to-live, the cache can deliver copies to the clients without any consistency checks. At the expiration of the delay, the copy is removed from the cache.

In our simulations, we used a ratio of 0.2, as it is the default in the Squid cache [9]. That is:

$$\frac{T_{removed} - T_{cached}}{T_{cached} - T_{last\_modification}} = 0.2$$

AlexCheck: This policy is identical to Alex except that, when the time-to-live expires, the copy is kept in the cache with a flag describing it as "possibly stale." Any hit on a possibly stale copy causes the cache to check the copy's consistency by sending an `If-Modified-Since` request to the master before answering the client's request. This policy is implemented in the Squid cache [9].

CacheInv: When a copy is created, the cache registers it at the server. When the master is updated, the server sends an invalidation to the registered caches to request them to remove their stale copies. This policy is similar to the AFS caching policy [22].

2.2.2 Replica Configurations

An alternative to having a cache in an AS is to have a replica there. Replica servers create permanent copies of documents. There are a relatively low number of such servers, which allows us to apply strong consistency policies that would not be affordable in the case of caches.

The traces we collected involve clients from a few thousand different ASes, which led us to consider caching systems with as many caches as ASes. However, it would not be feasible to create so many replication servers. We decided to place replication servers in the autonomous systems where most of the requests came from. The rationale for this choice is that most requests come from a small number of ASes.

Figure 2 shows the number of incoming requests per AS (once the ASes were sorted by decreasing number of requests). In our case, the top 10 ASes issued 53% of the requests, the top 25 ASes issued

a company's offices worldwide may be topologically but not geographically close.

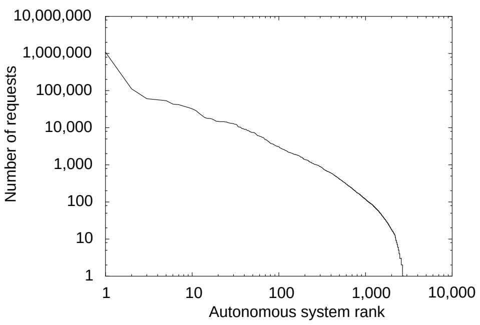

Figure 2: Number of requests per autonomous system

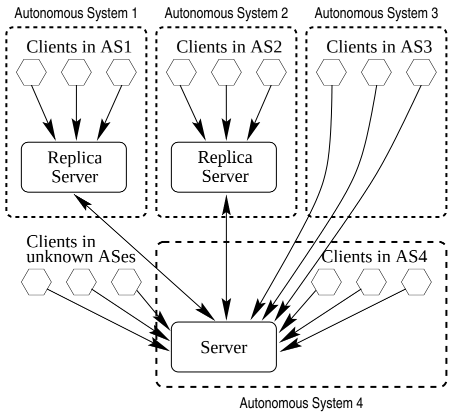

Figure 3: Replica configuration

62% of the requests, and the top 50 ASes issued 71% of the requests.

We decided to place replication servers only in the "top ASes." Clients located inside one of these ASes would be bound to their local replica. Other clients would send their requests directly to the server (see Figure 3).

We distinguished three replica configurations depending on the number of replicas created, which can be summarized as follows:

Repl10 (or **Repl25**, **Repl50:**) Replicas are created in the top 10 (or 25, 50) ASes. The consistency is maintained by pushing updates: when the master is updated, the server sends updated copies to the replica servers.

2.2.3 Hybrid Configurations

In the replica configurations presented in the previous section, many clients access the server directly (e.g., clients from autonomous system 3 in Figure 3). The AS of such clients generates only a few requests to our server, so it is not worthwhile installing a replica

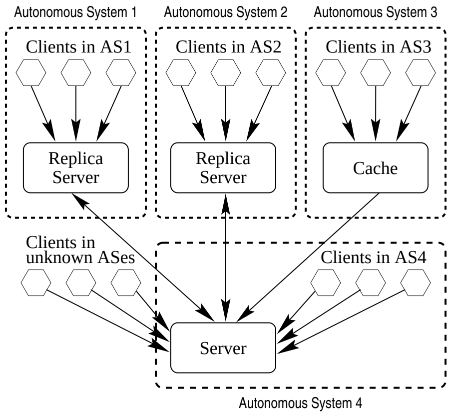

Figure 4: Hybrid configuration

there. However, it might benefit from a cache, which is cheaper to maintain than a replica. To take this into account, we created "hybrid configurations."

A hybrid configuration is similar to a replica configuration, but it includes a cache in each autonomous system which does not have a replica (see Figure 4). We defined two hybrid configurations depending on the consistency policy of the caches:

Repl50+Alex: Similar to **Repl50**, but the autonomous systems which have no replica server use an **Alex** cache instead.

Repl50+AlexCheck: Similar to **Repl50**, but the autonomous systems which have no replica server use an **AlexCheck** cache instead.

3 Experimental Setup

The experiment consisted of simulating the replication of each document with each of the ten configurations discussed above. We ran one simulation per document and per strategy, and measured (i) the delay at the clients, (ii) how many clients got stale copies, and (iii) the network bandwidth consumed. We then accumulated each of these values over all runs to determine the performance of any configuration over the entire set of documents.

We kept our simulations as close as possible to the real system. Therefore, they are not based on statistical traffic models, but rather on real traces and performance measurements.

3.1 Collecting Traces

To simulate the replication of documents, we needed to keep track of each event that can happen to a document: creation, update or request. The Web server

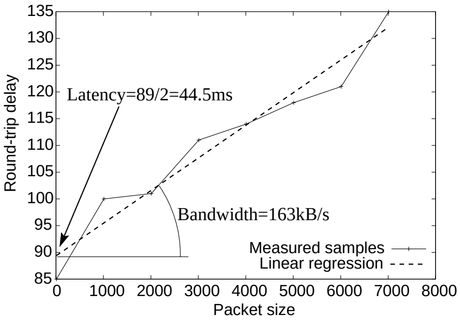

Figure 5: Determining the network performance to a host based on ping samples

logs gave us the necessary information about the requests for documents: request time and IP address of the clients. We also monitored the file system to detect any creation or update of a file located in the Web server directories. This way, we obtained information about the update times and the new sizes of documents. Document creation was handled as a special case of an update. We also measured the network performance between the server and each AS in our traces.

3.2 Measuring the Network Performance

To measure the network performance from our server to each AS in our experiment, we randomly chose 5 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 due to packet's size (see Figure 5). We assume symmetrical network performances: the performance from the server to any host is considered equal to the performance from this host to the server.

3.3 The Simulations

The simulations were based on a modified version of Saperlipopette, a discrete events simulator of distributed Web caches [18]. It allows one to simulate any number of caches, each cache being defined by its internal policies (replacement, consistency, cooperation) and its dimensioning. When given information about the network performance, Saperlipopette can replay trace files and calculate a number of metrics such as the cache hit rates, document access delays and the consistency of delivered documents. We extended this version to implement permanent replicas in addition to caches. We also added more consistency policies, such as invalidation.

3.3.1 Simulating Caching Configurations

The idea behind the caching configurations is, of course, not to deploy caches everywhere to access only our Web server. These caches are supposed to be used within the AS to access any Web server. As we reproduce only *part* of the traffic managed by each cache, we cannot simulate cache replacement policies; their behavior depends on the entire traffic seen by each cache. Therefore, we simulated caches without any replacement policy (i.e., caches of infinite size). To roughly reproduce the behavior of the replacement policies, we decided that a copy could not stay in a cache more than seven days, independent of any consistency consideration. This delay is a typical value of any document's time-to live inside a Web cache [16]. When the time-to-live value expires, the corresponding copy is removed from the cache.

3.4 Evaluation Criteria

Choosing a replication policy requires making tradeoffs. Replicating a Web document modifies the access time, the consistency of copies delivered to the clients, the master server load, the overall network traffic, etc. It is generally impossible to optimize all these criteria simultaneously. Therefore, evaluating the quality of service of the system should involve metrics that characterize the different aspects of the system's performance. We chose three metrics representing the access time, document consistency and global network traffic:

Total delay: This is the sum of all delays between the start of a client's request and the completion of the response (in seconds).

Inconsistency: This is the total number of outdated copies delivered to the clients.

Server traffic: This is the total number of bytes exchanged between the server and the intermediate servers or the clients. This metric measures all the inter-AS traffic, which we consider as the wide-area traffic; we do not take into account the traffic between the intermediate servers and the clients, as it is considered as "local."

One important remark is that all our metrics are additive: we can simulate each document separately

Table 1: Characteristics of the collected traces

Number of documents	17,368
Number of requests	2,118,572
Number of updates	9,143
Number of unique clients	107,386
Number of different ASes	2,785

and add the resulting values for each document in order to get the quality of service of the complete system. This would not be possible if the metrics were average values, for example.

4 Results

The result of our experiment is presented as follows: Section 4.1 gives a brief overview of the traces we collected; Section 4.2 shows the quality of service obtained when the same strategy is associated to all documents; Section 4.3 discusses methods for associating each document with its most-suited replication policy; Section 4.4 demonstrates the performance improvement such methods provide. Finally, Section 4.5 discusses the use of this method for building adaptive Web documents.

4.1 Collected Traces

We collected traces from Sunday, 29 August 1999 to Saturday, 3 October 1999 (i.e., 5 weeks). Table 1 shows some statistics about the resulting trace. We can see that our server handles medium-size traffic, and that documents are not updated very frequently (the average life-span is 67 days). We expect large servers, such as electronic-commerce servers, to have more heterogeneous document sets than ours. Therefore, they should benefit more than us from the ability to choose the replication strategies per document.

4.2 One-size-fits-all Strategies

Table 2 shows the resulting performance when the same strategy is applied to all documents (one-size-fits-all strategies). As we expected, the **NoRepl** strategy has bad results compared to the others in terms of delay and traffic. On the other hand, it provides perfect consistency.

Most policies are good with respect to one or two metrics, but none of them optimizes on all three metrics. For example, **Repl50+Alex** and **Repl50+AlexCheck** provide excellent delays. On the other hand, they are not so good with respect to inconsistency and traffic. Other configurations have similar problems.

Table 2: Performance of the one-size-fits-all strategies.

Configuration	Delay (hours)	Incons. (no.)	Traffic (GB)
NoRepl	219.0	0	43.91
Check	229.2	0	23.60
Alex	96.4	5211	23.51
AlexCheck	96.6	4821	23.23
CacheInv	93.7	0	23.18
Repl10	177.4	0	43.60
Repl25	145.0	0	48.06
Repl50	121.9	0	55.55
Repl50+Alex	67.5	966	46.93
Repl50+AlexCheck	67.6	941	46.86

4.3 Assigning Optimal Strategies to Documents

Is it possible to find a configuration that provides good performance with respect to all metrics at the same time? To answer this question, we propose that each document has its own replication strategy. We first describe a method to assign a strategy to each document. We then compare the performance of custom configurations to those of one-size-fits-all configurations.

4.3.1 Assigning a Strategy to a Document

For a given document, finding the best replication strategy consists of deciding which strategy provides the best compromise between different metrics. We prefer a strategy that is relatively good with respect to all metrics rather than a strategy that is very good in one metric and very bad in the others.

We proceed as follows: based on the simulation results, each strategy is given a score. For a given document, the strategy with the lowest score is declared "optimal." The score of a strategy is a weighted sum of the evaluation metrics:

$$score = \frac{delay}{\alpha} + \frac{incons}{\beta} + \frac{traffic}{\gamma}$$

Choosing values for α, β, and γ allows one to determine the relative weight of each metric. The larger a weight is, the less the associated metric will influence the final result. Because different metrics are expressed in different units, the factors α, β and γ

are expressed such that a score is always dimensionless.

This method is used to assign a strategy to each document. Using it with the same parameter vector (α, β, γ) leads to what we call an *arrangement*: a parameter-specific set of *(document,strategy)*-pairs. Thus for a given (α, β, γ), each arrangement has an associated value, which is expressed as a vector $\langle total(metric_1), \ldots, total(metric_n) \rangle$ where $total(metric_k)$ denotes for metric k the value accumulated over all documents in the arrangement.

An evaluation of our assignment strategy can be found in an extended version of this paper [17].

4.4 Comparing One-size-fits-all to Custom Policies

Comparing arrangements is somewhat difficult because the values of arrangements actually form a partially ordered set. We prefer comparing each arrangement with an ideal *target point*. This point corresponds to the best achievable delay (obtained by selecting the policy providing the smallest delay for each document) and the best achievable traffic (obtained by selecting the policy providing the smallest traffic for each document). Of course, for a given document, the best policy in terms of delay and the best policy in terms of traffic are not always the same. Therefore, the target point is generally impossible to reach. Nevertheless, this point acts as an upper bound: it is impossible to obtain a better performance than the target. We can also use the target point to compare the arrangements: the closer we get to that point, the better the arrangement is.

To simplify matters, we chose to give β a very small value, making the optimization of consistency an absolute requirement. By subsequently modifying the relative weights of delay and traffic, we obtain a number of arrangements which implement various delay/traffic trade-offs.

Figure 6 shows the performance of arrangements, in terms of total delay and server traffic. Each point corresponds to the performance of one particular arrangement. Arrangements where all documents are given the same strategy are represented by a point. Custom arrangements provide a set of points, each point being obtained with one particular set of weights (α, β, γ).

Among the one-size-fits-all arrangements, some have good performance with respect to delay, but poor performance with respect to traffic; some others behave the other way round. However, none of them gets very close to the target. On the other hand,

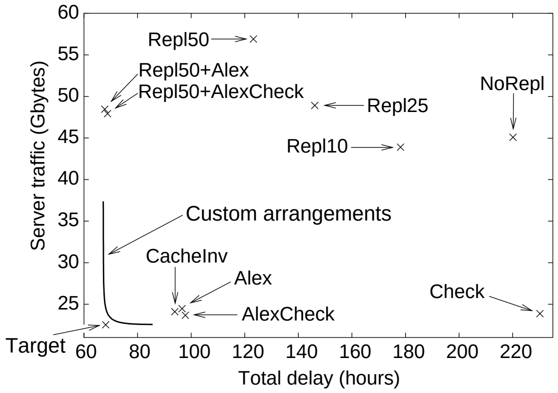

Figure 6: Performance of arrangements vs. one-size-fits-all configurations

we can see that custom arrangements are very close to the target if we compare them to one-size-fits-all configuration. This means that selecting replication strategies on a per-document basis provides a significant performance improvement over any one-size-fits-all configuration.

4.5 Towards Adaptive Web Documents

The results presented so far are based on a post-mortem analysis: knowing the past access pattern of a document we can determine which replication policy would have been optimal. Now, can we use our knowledge about past accesses to determine which policy will be optimal in the near future? Our hypothesis is that the traffic characteristics do not change very fast. Therefore, a strategy known to have been optimal in the recent past should stay close to the optimum in the near future. In this section, we consider "adaptive replicated documents." Such a document works as follows:

- The author of the document must choose one particular parameter vector (α, β, γ). The goal of the adaptive object will be to select the strategy which optimizes the score function, defined in section 4.3.1.

- When creating a document, we do not know anything about its future access pattern. The document uses a default replication policy (in our simulations, **Alex**). It also collects traces about the requests it receives.

- Periodically, the document has the opportunity to change its replication policy. Based on the

Table 3: Performance of adaptive documents

Configuration	Period 1			Period 2			Period 3		
	Delay (hours)	Incons. (no.)	Traffic (GB)	Delay (hours)	Incons. (no.)	Traffic (GB)	Delay (hours)	Incons. (no.)	Traffic (GB)
NoRepl	68.2	0	12.30	78.0	0	15.40	72.8	0	16.20
Check	71.4	0	6.63	81.4	0	7.95	76.4	0	8.03
Alex	30.9	25	6.66	33.8	1320	8.30	29.9	2197	8.16
AlexCheck	30.9	25	6.61	33.8	1214	8.20	29.9	2257	7.93
CacheInv	30.3	0	6.61	35.0	0	8.53	29.3	0	8.04
Repl10	56.4	0	10.80	62.1	0	14.06	58.9	0	15.45
Repl25	46.1	0	9.90	50.8	0	13.72	48.1	0	16.22
Repl50	39.2	0	9.32	43.5	0	12.83	39.2	0	16.96
Repl50+Alex	22.6	6	6.61	22.7	292	9.81	22.6	484	14.13
Repl50+AlexCheck	22.6	6	6.59	22.7	295	9.79	22.5	547	14.07
Adaptive	30.9	25	6.66	25.8	386	7.41	26.1	949	7.65
Optimal	22.6	0	5.34	22.8	0	6.47	23.5	0	7.12
Distance from optimal	37%	-	25%	13%	-	15%	11%	-	7%

traces it has collected, it simulates each replication strategy, calculates the score for each strategy, and determines which one would have performed the best. It then changes its replication policy to use this "optimal" one instead of the current one. Finally, it deletes the traces collected so far and starts collecting new ones.

We decided to use a period of 12 days between adaptations, so that our traces can be divided into three periods. Table 3 shows the performance of adaptive documents, compared with one-size-fits-all configurations. The "optimal" line in the table represents the performance of the optimal arrangement, based on *a posteriori* analysis. The closer the adaptive document performance is to the post-mortem performance, the more accurate the *a priori* choice of policies in the adaptive documents has been. The "distance from optimal" line shows the accuracy of the predictions.

During the first period, all adaptive documents use the default **Alex** replication policy. Their performance is therefore equal to that of the Alex policy. However, it is far from optimal (e.g., the total delay is 37% higher than the optimum). The "Period 2" column shows the performance after the first adaptation. We can see that the delay and traffic obtained by the adaptive documents are much closer to the optimum (the delay is only 13% higher than the optimum). During period 3, the distance from the optimum is only 11% from the optimum. Traffic figures have a similar behavior. We can conclude that adaptive documents are actually able to predict which policy will achieve the best delay/traffic tradeoff in the near future.

The inconsistency figures, unfortunately, do not converge to the optimal values. This is due to the low update rate in our traces. Being infrequent, updates are very hard for the adaptive objects to predict. In fact, most inconsistencies are due to documents that are updated only once in the entire trace. Such isolated updates are obviously impossible to predict from the past traces, leading to suboptimal policy choices.

One can argue that a few hundred inconsistencies are acceptable, if compared to the total number of requests (more than two million). In most cases, this should not be much of an issue. However, even in the case where consistency is considered very important, our research remains valid: it suffices to remove the weak consistency policies from the set of available policies. Adaptive objects would then be able to select the most suited policy among a set of strong consistency policies.

Another important remark is that our trace does not contain high load peaks commonly known as "flash crowds." The problem for handling flash crowds is to react as quickly as possible to a sudden traffic increase. The adaptive documents presented here would react only at the end of the period when the flash crowd happened; in most cases, it would be too late to take any measures. However, simple changes may be enough to solve the problem: for example, instead of adapting the documents at regular intervals, we can decide to adapt every time a document receives a given number of requests or if the request rate changes dramatically. This way, an adaptation would be started soon after the beginning of the flash crowd.

5 Related Work

A number of proposals have been made in the past to help improve the quality of service of the Web by means of better caching policies. Particularly relevant is the design of scalable cache consistency policies. As an alternative to the traditional Alex [7] and TTL policies, it has been shown that invalidation policies can lead to significant improvement of maintaining consistency at relatively low cost in terms of delays and traffic [14]. Several variants have been proposed. It is possible to propagate invalidations via a hierarchy of Web caches [25] or by using multicast [26]. Another possibility is for the server to piggyback information about recent document updates when caches contact it for a request [13] or to combine invalidation with leases [10].

Caches are an essential part of the Web infrastructure, but their efficiency has limits. Moreover, it seems that this efficiency decreases due to the long-term evolution of access patterns [3]. One solution to this problem is to systematically create document replicas. Based on a good knowledge of the access patterns, it is possible to place replicas close to the clients, thereby reducing delays [2, 5]. Such document distribution can be done by the server itself, as in push caches [11] and RaDaR [20], or by external services such as Akamai [1] and Sandpiper [21]. All these proposals tend to design a single policy which works well in all cases. In contrast, we prefer choosing one policy per document, depending on that document's access patterns.

All the policies cited here are good candidates for being incorporated in the set of differentiated strategies this paper advocates. However, most of them require implementing specific mechanisms. Invalidation protocols need various types of callback interfaces, replica distribution systems need to push document copies to the replica servers, and so on. One could think of incorporating such mechanisms in existing protocols. For example, many primitives for cache management were incorporated in HTTP during the design of version 1.1 [12]. However, such protocol modifications take time to be widely used. In addition, they often increase the protocol's complexity [15].

This paper advocates the simultaneous use of a large number of replication policies for different documents. In some cases, an author should even be allowed to develop a policy specially designed for a specific document. Therefore, we need a way to implement policies without having to modify HTTP or to build a specific infrastructure each time. The solution consists of separating transport and replication issues, by associating code with a document that can manage its replication. Such an approach has been taken in a number of projects. The active caches associate code with a document to enable caching of dynamic documents [6]. This proposal can be used to allow cached documents (dynamic or not) to manage their own consistency. In the W3Object system, highly visible caching mechanisms are proposed that can be modified by end users [8].

6 Conclusion

Our experiment demonstrates that no single replication policy is optimal for all Web documents. Instead, associating the most suited replication policy on a per-document basis leads to significant performance improvement. In addition to this, we showed that it makes sense to build replicated Web documents capable of individually adapting their replication policy, based on past-trace analysis.

The experiment presented was conducted over a large set of caching, replica and hybrid configurations. However, this set must be viewed only as a first example: a lot of other caching or replication policies could be added as well. We expect that increasing the number and diversity of policies will improve the resulting performance.

In the future, we plan to deploy a set of GlobeDoc servers and use the method presented in this article to decide on optimal replication policies. In particular, we aim to conduct studies on the traffic from other Web servers to see how the specifics of each server influence the optimal arrangements. A university Web server such as ours will likely not require the same strategies as a commercial server, for example.

Finally, this work is to be extended to the replication of other types of objects. We plan to investigate how the methods presented in this article can be applied to the replication of dynamic Web documents. Such results would lead us to a more general solution for choosing the replication policies of distributed objects.

References

[1] Akamai, `http://www.akamai.com/`.

[2] Michael Baentsch, Lothar Baum, Georg Molter, Steffen Rothkugel, and Peter Sturm, *Enhancing the Web infrastructure – from caching to replication*, IEEE Internet Computing **1** (1997), no. 2, 18–27.

[3] Paul Barford, Azer Bestavros, Adam Bradley, and Mark E. Crovella, *Changes in Web client access patterns: Characteristics and caching implications*, World

Wide Web (special issue on Characterization and Performance Evaluation) **2** (1999), no. 1-2, 15–28.

[4] Tony Bates, Elise Gerich, Laurent Joncheray, Jean-Michel Jouanigot, Daniel Karrenberg, Marten Terpstra, and Jessica Yu, *Representation of IP routing policies in a routing registry*, RFC 1786, March 1995.

[5] Azer Bestavros and Carlos Cunha, *Server-initiated document dissemination for the WWW*, IEEE Data Engineering Bulletin **19** (1996), no. 3-11.

[6] Pei Cao, Jin Zhang, and Kevin Beach, *Active cache: Caching dynamic contents on the Web*, Proceedings of the 1998 Middleware conference, September 1998, pp. 373–388.

[7] Vincent Cate, *Alex – a global file system*, Proceedings of the USENIX File System Workshop (Ann Arbor, Michigan), May 1992, pp. 1–11.

[8] Steve J. Caughey, David B. Ingham, and Mark C. Little, *Flexible open caching for the Web*, Computer Networks and ISDN Systems **29** (1997), no. 8-13, 1007–1017.

[9] Anawat Chankhunthod, Peter Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell, *A hierarchical Internet object cache*, Proceedings of the 1996 Usenix Technical Conference (San Diego, CA), January 1996, pp. 153–163.

[10] John Dilley, Martin Arlitt, Stéphane Perret, and Tai Jin, *The distributed object consistency protocol*, Tech. Report HPL-1999-109, Hewlett-Packard Laboratories, September 1999.

[11] James Gwertzman and Margo Seltzer, *The case for geographical push-caching*, Proceedings of the HotOS '95 Workshop, May 1995, pp. 51–55.

[12] Balachander Krishnamurthy, Jeffrey C. Mogul, and David M. Kristol, *Key differences between HTTP/1.0 and HTTP/1.1*, Computer Networks and ISDN systems **31** (1999), no. 11-16, 1737–1751.

[13] Balachander Krishnamurthy and Craig E. Wills, *Piggyback server invalidation for proxy cache coherency*, Computer Networks and ISDN Systems **30** (1998), no. 1-7, 185–193.

[14] Chengjie Liu and Pei Cao, *Maintaining strong cache consistency in the World-Wide Web*, IEEE Transactions on Computers **47** (1998), no. 4, 445–457.

[15] Jeffrey C. Mogul, *What's wrong with HTTP (and why it doesn't matter)*, Invited talk at the Usenix Technical Conference, June 1999.

[16] *NLANR caches "vital statistics"*, `http://www.ircache.net/Cache/Statistics/Vitals/`.

[17] Guillaume Pierre, Ihor Kuz, Maarten van Steen, and Andrew S. Tanenbaum, *Differentiated strategies for replicating Web documents*, Technical Report IR-467, Vrije Universiteit, Amsterdam, November 1999.

[18] Guillaume Pierre and Mesaac Makpangou, *Saperlipopette!: a distributed Web caching systems evaluation tool*, Proceedings of the 1998 Middleware conference, September 1998, pp. 389–405.

[19] James E. Pitkow, *Summary of WWW characterizations*, Computer Networks And ISDN Systems **30** (1998), no. 1-7, 551–558.

[20] Michael Rabinovich and Amit Aggarwal, *RaDaR: A scalable architecture for a global Web hosting service*, Proceedings of the 8th International World-Wide Web Conference, May 1999.

[21] Sandpiper, `http://www.sandpiper.com/`.

[22] Mahadev Satyanarayanan, *Scalable, secure, and highly available distributed file access*, IEEE Computer **23** (1990), no. 5, 9–18.

[23] Maarten van Steen, Philip Homburg, and Andrew S. Tanenbaum, *Globe: A wide-area distributed system*, IEEE Concurrency (1999), 70–78.

[24] Maarten van Steen, Andrew S. Tanenbaum, Ihor Kuz, and Henk J. Sips, *A scalable middleware solution for advanced wide-area Web services*, Distributed Systems Engineering **6** (1999), no. 1, 34–42.

[25] Jian Yin, Lorenzo Alvisi, Mike Dahlin, and Calvin Lin, *Hierarchical cache consistency in a WAN*, Proceedings of the 1999 Usenix Symposium on Internet Technologies and Systems (USITS'99), October 1999.

[26] Haobo Yu, Lee Breslau, and Scott Shenker, *A scalable Web cache consistency architecture*, Proceedings of the ACM SIGCOMM'99 Conference, September 1999.

Vitæ

Guillaume Pierre is a post-doc researcher at the Vrije Universiteit in Amsterdam. He holds an engineer degree from the "Institut d'Informatique d'Entreprise" and a Ph.D. in Computer Science from INRIA and the University of Evry-val d'Essonne. His main interests are flexible computing systems and Internet performance optimization.

Ihor Kuz has an M.Sc. in Computer Science (1996) from the Vrije Universiteit in Amsterdam. He is currently a Ph.D. student at the Delft University of Technology, doing research in the field of worldwide scalable distributed Web services. His research interests include operating systems, scalable distributed systems, and Web-based technologies.

Maarten van Steen is associate professor at the Vrije Universiteit in Amsterdam. He received an M.Sc. in Applied Mathematics from Twente University (1983) and a Ph.D. in Computer Science from Leiden University (1988). He has worked at an industrial research laboratory for several years in the field of parallel programming environments. His research interests include operating systems, computer networks, (wide-area) distributed systems, and Web-based systems. Van Steen is a member of IEEE Computer Society and ACM.

Andrew S. Tanenbaum has an S.B. from M.I.T. and a Ph.D. 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. Prof. 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, Tanenbaum is the author of five books and over 80 refereed papers. He is a Fellow of ACM, a Fellow of 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.