Differentiated Strategies for Replicating Web Documents

Guillaume Pierre
Ihor Kuz
Maarten van Steen
Andrew S. Tanenbaum

Internal report IR-467
November 1999

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 Web documents have very diverse characteristics, and that no single caching or replication policy can efficiently manage all documents. Instead, we propose that each document is replicated with its most-suited policy. We collected traces on our university's Web server and conducted simulations to determine the performance such configurations would produce, as opposed to configurations 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.

vrije Universiteit

Department of Mathematics and Computer Science

1 Introduction

Every Web user has experienced slow document transfers. To reduce the access time, one possible solution is to replicate the documents. Doing so 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 able to help enforce such consistency. Mostly, we can assume that all updates 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 for the master (or the server hosting it) to keep track of all copies, and to contact each slave when the document is updated. In such cases, it is possible to multicast the new version, or to request a stale copy to be destroyed. Pull strategies require for slaves to 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 A commonly used variant is for a copy to destroy itself when it suspects it is out-of-date, without even checking the master.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 mirror sites.

Which replication strategy is the most suited for Web documents? This is a difficult question, and much research is currently trying to answer it. The main obstacle to a good solution is the heterogeneity of documents. For example, sizes, popularity, the geographical location of clients and the frequency of updates are very different from one document to another [16]. 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 with the best-suited policy. Furthermore, the choice of a replication policy should be renegotiable in case circumstances change.

How multiple replication strategies can be supported and integrated in the current World-Wide Web is addressed by GlobeDoc [20]. Using GlobeDocs, Web pages (or groups of pages) are encapsulated into distributed objects. This encapsulation allows one to easily associate an object with any replication policy [19]. 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. In the future, we envision GlobeDoc-enabled browsers which could directly contact the replicated 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. To do so, we monitored our university's Web server[1] 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 are replicated using the same strategy; in the second case we chose the "best" strategy for each document. The results show that custom strategies provide a clear performance improvement compared to any one-size-fits-all strategy.

The rest of this paper is organized as follows: Section 2 describes the configurations we worked with, Section 3 describes our experimental setup, Section 4 shows the simulation results and discusses the methods we designed for associating optimal strategies to documents. Section 5 presents related

[1] `http://www.cs.vu.nl/`

work. We present our conclusions in Section 6.

2 System Configurations

The experiment consisted of studying the documents hosted by one particular server (the server of our computer science department). Clients located worldwide retrieved the documents from the server, or from intermediate servers (caches or replicas). Interposing replication protocols between the server and the intermediate servers would modify the user-level perceived quality of service.

We first describe our system model, after which we present various configurations for our experiment.

2.1 System Model

2.1.1 Document Model

We assume that all document updates are done at the master (which is always located at the server) by means of any document-editing facility. The only requirement is that the server can notice such updates to propagate the information to the copies (if the replication policy needs it).

To simplify our experiments, we considered only static documents. Although dynamic documents such as CGIs, ASPs and PHP are easily embeddable in GlobeDocs, the simulation of replication for such documents is not trivial. Static documents are self-contained, which allowed us to capture and reproduce all accesses they received (creation, update, consultation). Also, dynamic documents may require external sources of information such as files and databases for generating a new version of a document at each request. The nature of these external sources of information and the way they act upon the generated versions are very diverse, which makes such documents very difficult to simulate. Therefore, we excluded them from our experiment.

2.1.2 Placement of Intermediate Servers

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

We decided to group the clients based on the autonomous systems which hosted them. Autonomous systems (or *ASes*) are used to achieve efficient world-wide 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 in the Internet.

An interesting feature of ASes from our point of view is that they generally materialize relatively large groups of hosts close to each other with respect to the network topology.[2] Therefore, we can assume that the network connection performance is much better inside an AS than between two ASes.

[2]Note that host proximity in terms of network topology does not imply geographical proximity. For example, all of a company's offices worldwide may be topologically but not geographically close.

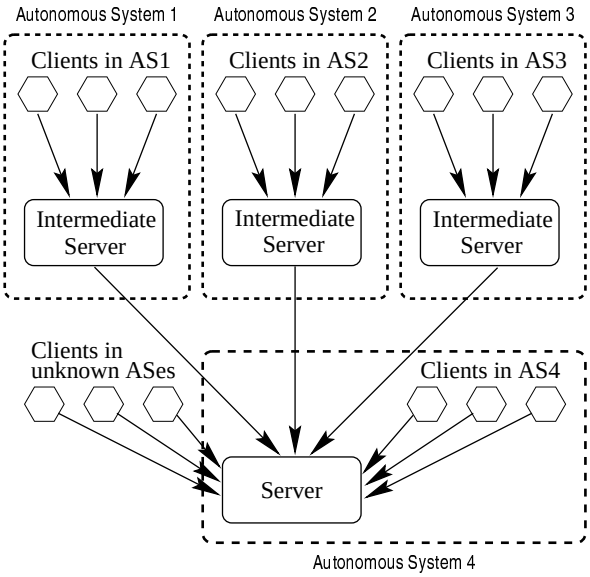


Figure 1: *System model*

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. The other exception is for the few clients for which we could not determine the AS. These clients also access the server directly.

2.2 Configurations

For each document, we consider a number of system 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 at all. All clients contact the server directly. There are no intermediate servers.

2.2.1 Caching Configurations

Caching configurations use proxy caches in place of the intermediate servers. We 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]. Therefore, a document updated very recently will be given a short time-to-live, as it is assumed that it will be updated soon again. On the other hand, documents whose last update is long ago are not assumed to be updated soon.

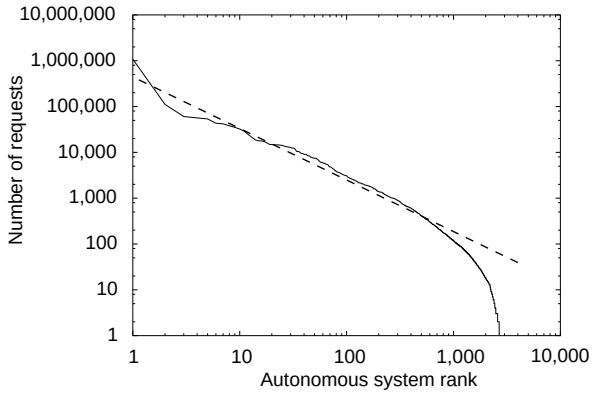


Figure 2: *Distribution of requests per autonomous system*

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. 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 caches [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 in order to request them to remove their stale copy. This policy is similar to the AFS caching policy [18].

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 them, which allows us to use 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 reasonable 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). The shape of the curve (with a logarithm/logarithm scale) is close to a straight line, which means that it follows a self-similar distribution. In our case, the 10 "top ASes" issued 53% of the requests, the 25 "top ASes" issued 62% of the requests, and the 50 "top ASes" issued 71% of the requests.

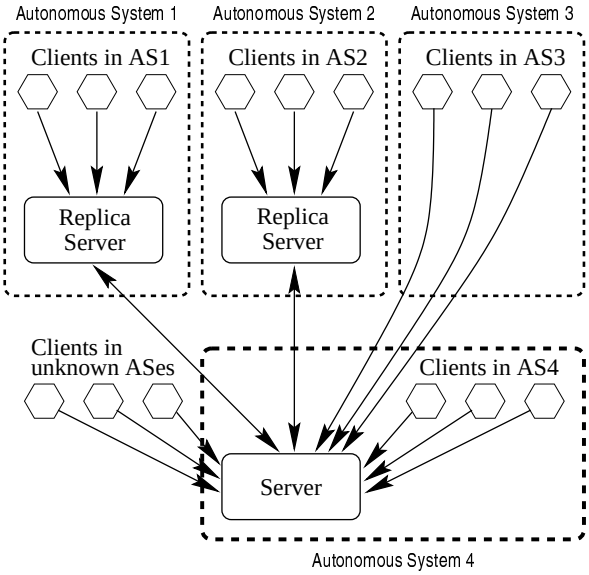


Figure 3: *Replica configuration with two replica servers*

We decided to place a number of replication servers in the "top ASes." Clients located inside one of these ASes are bound to their local replica. Other clients 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 10 (or 25, 50) "top ASes." The consistency is maintained by pushing updates: when the master is updated, the server sends updated copies to the replica servers. Note that this approach is different from **CacheInv**: in **Repl** systems, the updated documents are sent to the replicas, whereas **CacheInv** only informs the caches that they have a stale copy they should destroy.

2.2.3 Hybrid Configurations

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

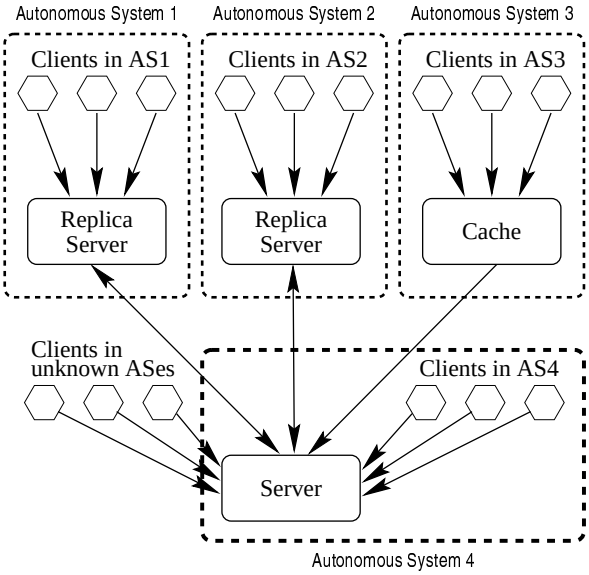


Figure 4: *Hybrid configuration*

3 Experimental Setup

The experiment consisted of simulating the replication of each document with each configuration. We ran one simulation per document and per strategy, and measured the delay at the clients, whether or not the returned copies were stale, and the consumed network bandwidth. 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 traffic models, but rather on real traces and performance measurements. Section 3.1 describes the trace collection; Section 3.2 discusses the network performance measurements, and Section 3.3 describes the simulation itself. Finally, Section 3.4 discusses the metrics we used to evaluate the configurations.

3.1 Collecting Traces

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

The Web server logs gave us the necessary information about the requests to 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.

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-

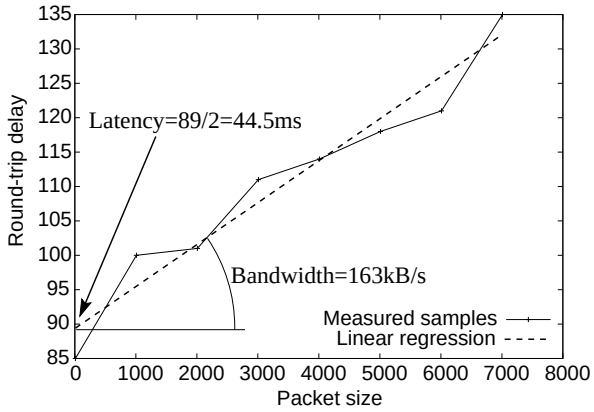


Figure 5: *Determining the network performance to a host based on ping samples*

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.

Measuring the network performance inside each AS is more complicated because we cannot send ping packets inside an AS. Therefore, we assume that the performance figures inside the AS where our server is located are representative for all intra-AS communications. We measured the network performance inside our AS. This is only a first approximation. We are currently in contact with other groups in order to measure the performance inside a number of other ASes. This should give us a better insight about typical intra-AS communication performance.

3.3 The Simulations

The simulations are based on a modified version of Saperlipopette, a simulator of distributed Web caches [15]. 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 topology and performance, Saperlipopette can replay trace files and calculate a number of metrics such as the cache hit rates, documents 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 in order 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 [14]. When the time-to-live value expires, the corresponding copy is removed from the cache.

Table 1: *Characteristics of the collected traces after filtering*

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

3.4 Evaluation Criteria

Choosing a replication policy is fundamentally based on 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 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. Finally, Section 4.3 discusses methods for associating each document with its most-suited replication policy, and demonstrates the performance improvement such methods provide.

4.1 Collected Traces

We collected traces from Sunday 29 August 1999 to Saturday 3 October 1999 (i.e., 5 weeks). Although our Web server hosts a large number of documents, most of them are rarely accessed. To prevent pollution of our results, we filtered the traces by removing the documents that had been accessed fewer than 10 times during our trace collections: documents that are hardly accessed obviously don't need to be replicated or cached.

Table 2: *Performance of the one-size-fits-all strategies. "Delay" stands for the sum of user-perceived delays. "Incons" is the number of outdated documents delivered to the clients. "Traffic" is the amount of network traffic between the master and the slave servers.*

Configuration	Delay (hours)	Incons. (nb)	Traffic (GB)
NoRepl	219.0	5	44.96
Check	229.2	5	24.16
Alex	96.4	5218	24.08
AlexCheck	96.6	4828	23.79
CacheInv	93.6	5	23.74
Repl10	177.4	6	44.64
Repl25	145.0	6	49.22
Repl50	121.9	7	56.88
Repl50+Alex	67.5	971	48.06
Repl50+AlexCheck	67.6	946	47.99

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 live-span is 67 days).

We expect servers of larger size such as electronic-commerce servers to have more heterogeneous document sets than us. 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 very good consistency: only five out-of-date documents delivered to the clients.

These five outdated documents do not mean that **NoRepl** generates any inconsistency (it obviously does not). In fact, they are caused by a side effect of the simulation. In some cases, an update and a request were issued at the same time on the same document. By chance, the start of the request was simulated before the update.[3] By the time the response reached the client, its version was considered out-of-date by the simulator. The same remark applies to other replication policies having a low number of inconsistency (5, 6 and 7).

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 behavior.

[3] Saperlipopette is a discrete event simulator and does not allow true parallelism between events.

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 follows its own replication strategy.

We first describe a number of candidate methods that can be used to assign a strategy to each document. We then compare the performance of such configurations to those of one-size-fits-all configurations.

4.3.1 Proposed Methods for 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 which is relatively good with respect to all metrics rather than a strategy which is very good in one metric and very bad in the others.

We designed three *methods* for assigning a strategy d to a document D. Each method M evaluates a strategy s by first giving it a *score* for its result with respect to a single metric. The scores per metric are then taken together leading to a final *grade* $G_M(s)$ for that strategy. The strategy with the highest grade is declared the best one for document D.

Running this algorithm once per document assigns an "optimal" strategy to each document.

The methods we discuss next differ only in the way they grade strategies and, in particular, in the way they give scores to each metric for a given strategy and document.

- **Ranking functions**: a strategy is given a score for each metric based on its rank with respect to other strategies. The strategy that performs best (in metric m) is given a score of 10 points for m. The strategy that performs second best in m is given a score of 5 points. The following three strategies receive scores of 3, 2, and 1 points, respectively, while all others get no points. The final grade $G_{ranking}(s)$ is obtained by simply adding the scores of each metric for a given strategy s.

  As presented so far, the ranking function considers all metrics equally important. However, we often value one metric higher than the others. For example, one may wish to first minimize inconsistencies and, given this constraint, to find a good trade-off between the other two metrics. To enable such a choice, we weighted the scores for each metric. It $score_s(m)$ is the score obtained by strategy s for metric m, then:

  $$\begin{aligned} G_{ranking}(s) = \quad & weight_1 \times score_s(m_1) \\ & + weight_2 \times score_s(m_2) \\ & + \ldots \end{aligned}$$

  Giving more weight to inconsistency (e.g., by multiplying its score by 2) will then favor its optimization.

  By modifying the relative weights of metrics, we thus obtain a *family* of ranking functions which represent different trade-offs between the metrics.

- **Range functions**: the family of ranking functions tries to select strategies that are as good as possible over all three metrics. However, it takes into account only the ranking of strategies and not *how much better* one strategy is compared to another. The family of range functions tries to solve this issue.

A strategy gets a score of 10 points with respect to a given metric, if that strategy ranks as the best one. The strategy that ranks as the worst with respect to that metric, is given a score of 0 points. Let $res_{best}(m)$ denote the result in metric m for the strategy that ranks best, and $res_{worst}(m)$ the result for the strategy that ranks worst. Then, each other strategy s receives a score (for the same metric m), relative to its result $res_s(m)$ compared to the best and worst obtained results:

$$score_s(m) = 10 \times \frac{res_{worst}(m) - res_s(m)}{res_{worst}(m) - res_{best}(m)}$$

Again, the scores for each metric are added to a final grade $G_{range}(s)$ and the strategy with the highest grade is declared best for document D. Similar to the family of ranking functions, we weighted the metric-specific elements of the range function again leading to a *family* of range functions.

- **Cost functions**: the family of cost functions is a variation of the family of range functions. However, instead of considering the relative position a strategy has based on how good or bad the value of a metric was compared to values of other strategies, it gives a score to a strategy by directly taking the value for each metric:

$$score_s(m) = \frac{res_s(m)}{C_m}$$

Because different metrics are expressed in different units, we introduced a correcting factor C_m such that $score_s(m)$ is always dimensionless.

Unlike the ranking and range functions, the "best" strategy for a given document is the one which *minimizes* the cost function, where we assume that for each metric, a low value is better than a high value.

4.3.2 Comparing the Methods

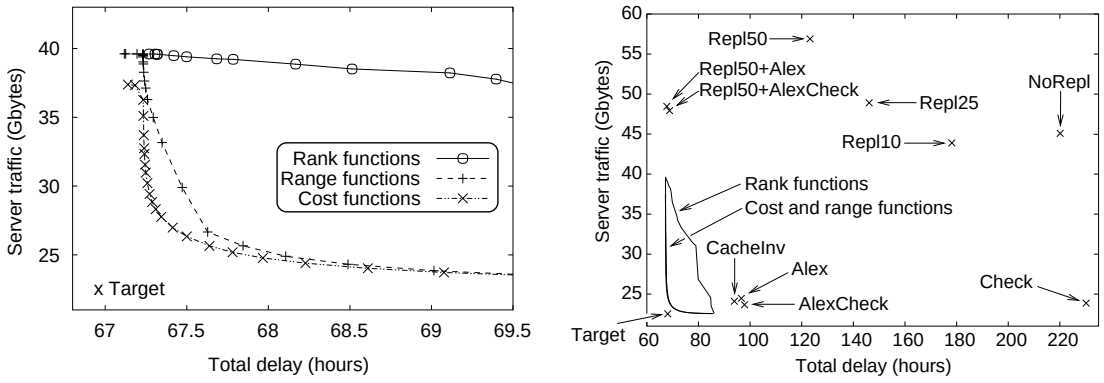
A method from a particular family as discussed above, is used to assign a strategy to each document. Using a single method for all documents, leads to what we call an *arrangement*: a method-specific set of *(document,strategy)*-pairs. Each arrangement has an associated value, which is expressed as a 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.

To compare the quality of a method with respect to how good it is in actually assigning the "best" strategy to a document, we simply need to compare the values of arrangements. Comparison is somewhat difficult because the values of arrangements actually form a partially ordered set.

To simplify matters, we decided, for each method, to assign a large weight to the inconsistency metric, making the optimization of consistency an absolute requirement. By subsequently modifying the relative weights of delay and traffic, we obtain, *per family of methods,* a number of arrangements which implement various delay/traffic trade-offs.

Figure 6(a) shows the performance of arrangements, in terms of total delay and server traffic. Each point on a curve corresponds to an arrangement obtained by running one of the methods over the set of documents, with one particular set of weights.

We compare each arrangement with an ideal *target point.* This point corresponds to the best achievable delay (obtained by selecting for each document the policy with the smallest delay) and the best achievable traffic (obtained by selecting for each document the policy with the smallest traffic). Of course, for a given document, the best policy in terms of delay and the best policy in terms of traffic



(a) *Close view* (b) *Wide view*

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

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.

It is important to note that, due to the partial ordering of arrangements, it is generally impossible to select one arrangement as the "best" one. The only case where we can compare two arrangements is if one has better performance than the other with respect to *all* metrics at the same time. In this sense, it can be seen from Figure 6(a) that the best method is always from the family of cost functions. Its curve is closer to the target than any method from either the family of ranking functions or that of range functions.

The points representing one-size-fits-all configurations all fall out of Figure 6(a). Figure 6(b) shows them in a wider view of the same graph. We can see that all our arrangements are very close to the target if we compare them to any one-size-fits-all configuration. This means that selecting replication strategies on a per-document basis provides a performance improvement over any one-size-fits-all configuration.

4.3.3 Optimality of the Cost-Function Method

Among the three methods for selecting per-document strategies, the family of cost functions is clearly the best. One can then wonder if we could imagine yet a better method that would lead to arrangements with better results.

We searched for the optimal arrangements using a brute-force method. We iterated over all possible arrangements and selected the optimal ones. We could do so only for a very small set of documents. As each document must be tested with 10 configurations, if we have d documents, that makes 10^d arrangements to test. Computing time limited us to a set of eight documents.

Many arrangements are obviously not optimal: each time one arrangement provided a better (performance) value than another with respect to all metrics, we discarded the second. The remaining arrangements are the optimal ones.

Figure 7 shows the performance of arrangements provided by the brute-force method, and by the

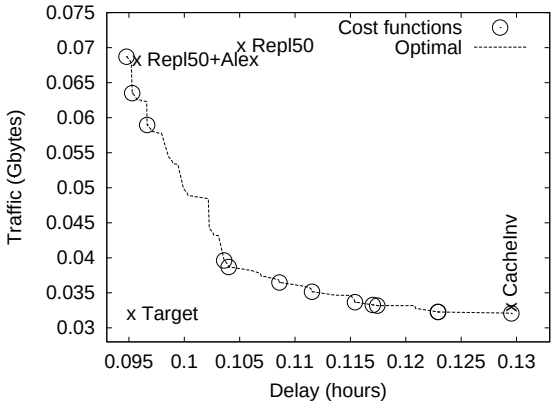


Figure 7: *Comparing the optimal mappings to the cost functions' ones*

family of cost functions. Somewhat surprisingly, the family of cost functions provided only a small number of arrangements, no matter how slowly we changed the weights on each metric. However, all cost function arrangements are located on the optimal curve, or very close to it.

We can then conclude that, at least with respect to this small example, the family of cost functions are not only better than the other two families of strategy-assignment methods, but actually provide truly optimal arrangements. On the other hand, not all optimal arrangements are provided by the cost functions.

4.3.4 Stability of Optimal Arrangements

Is it possible to rely on past access patterns to predict a good arrangement for the future? If the access pattern of a document changes very fast, an arrangement obtained from past data would most likely not be optimal any more at the time it is being used.

To answer this question, we split our trace file in two parts of equal duration (2.5 weeks each). We chose one particular cost function, and determined the arrangement it provides, based on the first trace. We then used this arrangement with the trace from the second part.

Table 3 compares this arrangement's performance (configuration "Obtained") with all one-size-fits-all configurations, as well as with the arrangement resulting from the cost function during the second period. We can see that the obtained performance is worse than the cost function arrangement, but still better than any of the one-size-fits-all configurations.

In particular, the number of outdated documents delivered to the clients by the "obtained" arrangement is high (490). This is due to the fact that our Web site was redesigned from scratch during the second period. Therefore, many documents which had not been modified during the first period were modified during the second one. The "optimal" policies for stable documents failed to maintain consistency when these documents were updated.

Between the two periods, the optimal strategy assignment has changed for 3,297 documents (over a total of 17,401 documents). In fact, most assignment changes were from one **Repl** strategy to another (including **Repl50+Alex** and **Repl50+AlexCheck**). This means that what did actually change were not the optimal replication *protocols* but rather the optimal *number* and *placement* of replicas. We can then expect replication strategies able to dynamically adapt the number and placement of replicas, such as push caches [10], to be good alternatives to our static replication policies.

Table 3: *Using the arrangement based on the first period during the second period*

Configuration	Delay (hours)	Incons. (nb)	Traffic (GB)
NoRepl	104.4	2	21.33
Alex	42.9	3885	10.13
CacheInv	42.1	2	10.03
Check	109.2	2	10.32
AlexCheck	43.0	3730	10.05
Repl10	89.3	3	19.87
Repl25	70.6	3	19.76
Repl50	59.0	3	19.28
Repl50+Alex	31.8	1068	14.88
Repl50+AlexCheck	31.8	1034	14.85
Cost function	32.0	2	10.60
Obtained	33.2	490	10.88

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 [13]. Several variants have been proposed. It is possible to propagate invalidations via the the hierarchy of Web caches [21] or by using multicast [22]. Another possibility is for the server to piggyback information about recent documents' updates when caches contact them for a request [12].

Caches are an essential part of the Web infrastructure. However, 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, therefore reducing delays [2, 5]. Such document distribution can be done by the server itself, as push caches do [10], or by external services such as Akamai [1] and Sandpiper [17].

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 to implement 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 have been incorporated in HTTP during the design of version 1.1 [11]. However, such protocol modifications take time to be widely used. In addition, they often increase the protocol's complexity.

This paper advocates the simultaneous use of a large number of replication policies. 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 to a doc-

ument 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 can be successfully applied to all Web documents. Instead, associating the most suited replication policy on a per-document basis leads to significant performance improvement. In addition to this, one document author can weight the cost function as to reflect the tradeoffs that (s)he considers preferable for these particular documents. This cost function provides an optimal mapping which implements the desired tradeoff.

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 would like to study the traffic from other Web servers to see how the specifics of each server influences the optimal arrangements. A university Web server such as ours will likely not lead to 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] BAENTSCH, M., BAUM, L., MOLTER, G., ROTHKUGEL, S., AND STURM, P. Enhancing the Web infrastructure – from caching to replication. *IEEE Internet Computing 1*, 2 (Mar. 1997).

[3] BARFORD, P., BESTAVROS, A., BRADLEY, A., AND CROVELLA, M. E. Changes in Web client access patterns: Characteristics and caching implications. *World Wide Web (special issue on Characterization and Performance Evaluation)* (1999).

[4] BATES, T., GERICH, E., JONCHERAY, L., JOUANIGOT, J.-M., KARRENBERG, D., TERPSTRA, M., AND YU, J. Representation of IP routing policies in a routing registry. RFC 1786, Mar. 1995.

[5] BESTAVROS, A., AND CUNHA, C. Server-initiated document dissemination for the WWW. *IEEE Data Engineering Bulletin 19*, 3-11 (Sept. 1996).

[6] CAO, P., ZHANG, J., AND BEACH, K. Active cache: Caching dynamic contents on the Web. In *Proceedings of the 1998 Middleware conference* (Sept. 1998), pp. 373–388.

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

[8] CAUGHEY, S. J., INGHAM, D. B., AND LITTLE, M. C. Flexible open caching for the Web. *Computer Networks and ISDN Systems 29*, 8-13 (1997), 1007–1017.

[9] CHANKHUNTHOD, A., DANZIG, P., NEERDAELS, C., SCHWARTZ, M. F., AND WORRELL, K. J. A hierarchical Internet object cache. In *Proceedings of the 1996 Usenix Technical Conference* (San Diego, CA, Jan. 1996).

[10] GWERTZMAN, J., AND SELTZER, M. The case for geographical push-caching. In *Proceedings of the HotOS '95 Workshop* (May 1995).

[11] KRISHNAMURTHY, B., MOGUL, J. C., AND KRISTOL, D. M. Key differences between HTTP/1.0 and HTTP/1.1. *Computer Networks and ISDN systems 31*, 11-16 (May 1999), 1737–1751.

[12] KRISHNAMURTHY, B., AND WILLS, C. E. Piggyback server invalidation for proxy cache coherency. *Computer Networks and ISDN Systems 30*, 1-7 (1998), 185–193.

[13] LIU, C., AND CAO, P. Maintaining strong cache consistency in the World-Wide Web. *IEEE Transactions on Computers 47*, 4 (Apr. 1998), 445–457.

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

[15] PIERRE, G., AND MAKPANGOU, M. Saperlipopette!: a distributed Web caching systems evaluation tool. In *Proceedings of the 1998 Middleware conference* (Sept. 1998), pp. 389–405.

[16] PITKOW, J. E. Summary of WWW characterizations. *Computer Networks And ISDN Systems 30*, 1-7 (Apr. 1998), 551–558.

[17] SANDPIPER. `http://www.sandpiper.com/`.

[18] SATYANARAYANAN, M. Scalable, secure, and highly available distributed file access. *IEEE Computer 23*, 5 (May 1990).

[19] VAN STEEN, M., HOMBURG, P., AND TANENBAUM, A. S. Globe: A wide-area distributed system. *IEEE Concurrency* (January-March 1999), 70–78.

[20] VAN STEEN, M., TANENBAUM, A. S., KUZ, I., AND SIPS, H. J. A scalable middleware solution for advanced wide-area Web services. *Distributed Systems Engineering 6*, 1 (Mar. 1999), 34–42.

[21] YIN, J., ALVISI, L., DAHLIN, M., AND LIN, C. Hierarchical cache consistency in a WAN. In *Proceedings of the 1999 Usenix Symposium on Internet Technologies and Systems (USITS'99)* (Oct. 1999).

[22] YU, H., BRESLAU, L., AND SHENKER, S. A scalable Web cache consistency architecture. In *Proceedings of the ACM SIGCOMM'99 Conference* (Sept. 1999).