Towards Scalable Web Documents

Anne-Marie Kermarrec
IRISA/University of Rennes
e-mail: akermarr@irisa.fr

Ihor Kuz
Delft University of Technology
e-mail: i.kuz@cs.tudelft.nl

Maarten van Steen (contact)
Andrew S. Tanenbaum
Vrije Universiteit, Faculty of Mathematics & Computer Science
De Boelelaan 1081a, 1081 HV, Amsterdam, The Netherlands
tel: +31 20 444 7784 / fax: +31 20 444 7653
e-mail: {steen,ast}@cs.vu.nl

Internal report IR-452, October 1998

This paper is a complete revision of "A Framework for Consistent, Replicated Web Objects," published by the same authors in the Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS), May 1998, Amsterdam, pp. 276–284. An electronic version of the ICDCS paper can be found at http://www.cs.vu.nl/~steen/globe/publications.html.

Article summary. The current Web is running into serious scalability problems. The standard solution is to apply techniques like caching, replication, and distribution. Unfortunately, as the variety of Web applications continues to grow, it will be impossible to find a single solution that fits all needs.

The authors advocate a different approach to tackling scaling problems. Instead of seeking a general-purpose solution, they argue that it makes more sense to look at each Web document separately. For each document, three issues need to be addressed: placement of replicas, required coherence, and best coherence protocol. The authors examine each of these issues, and identify the alternatives.

However, forcing developers to decide on the best alternatives will turn the Web into an unworkable system. Therefore, a number of possible ways to reduce complexity is indicated. Also, the authors briefly discuss a wide-area infrastructure that can be used as a flexible basis for developing per-document solutions.

Keywords: *distributed systems, replication, coherence, World-Wide Web, worldwide scalable systems*

With the continuing growth of the Web's popularity, we are increasingly confronted with its limited scalability. In general, scalability problems can be alleviated by means of caching and replication.[1] In the Web, much attention has been paid to caching. Recently, it has been recognized that caching alone is not enough. In particular, replication techniques by which updates are pushed to clients are needed as well.[2]

Caching and replication inherently lead to *consistency problems*: when a page is cached or replicated, a modification of one copy makes that copy different from the others. In the Web, consistency is managed by means of a simple protocol: whenever a page is retrieved from a cache, the cache checks when that page was last updated at its server. If the page was updated at the server subsequent to its being cached at the client, the cache entry is refreshed by fetching the page from the server. Otherwise, the currently cached version is handed over to the client. Variations exist in which a cache entry is refreshed only after some time has expired leading to weaker consistency, but in general better performance.

Unfortunately, proposals for both replication and coherence protocols in the Web have in common that they treat all pages alike. In other words, they assume that one particular form of replication or consistency is required for all pages. Moreover, there are only very few consistency protocols to choose from. With the large variety of Web pages currently existing, and the increasing alternative applications of Web technology, it is questionable whether such an approach makes any sense even now, let alone in the near future.

Consider, for example, a personal Web document. (A **Web document** is a collection of logically related pages, including their icons, images, sounds, applets, etc. A document can also contain interactive parts, such as in the case of whiteboards or distributed spreadsheets.) In general, although it may be worthwhile to let an individual browser cache pages of such a document, site-wide caching by a Web proxy is less likely to improve performance. In addition, temporarily having a stale copy of personal home pages will generally not matter. In contrast, Web documents that are dynamically updated with stock market data will have much stronger consistency requirements. Moreover, if the client site is populated by brokers, site-wide caching may improve performance considerably, especially if combined with push technology.

Any large system that has to support a wide variety of users, must differentiate between many use cases to operate efficiently and effectively. At the same time it may be acceptable to somewhat limit its user in their behavior to guarantee good overall performance. In this context, the Web is not providing as much flexibility as it should, which is now leading to serious scaling problems.

It is clear that we have a difficult problem at hand. Alleviating scalability problems requires that we replicate Web documents. This means that we first have to decide on where and when replicas are to be placed. As a second step, we have to find solutions to solve inconsistencies between replicas. As no single good solution exists, we need to identify minimal coherence requirements for each Web document. Third, we have to decide on the best coherence protocol, and this will vary from document to document. Again, many choices can be made and each choice will have its effect on performance. In the following, we take a look at each of these issues.

Deciding on Replication

The first issue that needs to be resolved, is what we refer to as **replication management**: deciding on where, when, and by whom replicas are to be placed, and which Web documents (or parts thereof) must be replicated.

Caching represents a form of on-demand replication managed by clients, exploiting temporal locality (given the access patterns of the Web, spatial locality is not always relevant). There are many situations in which caching has shown to be effective, but poor hit rates also show that, in general, it cannot be the only solution to solve scalability problems.[3]

Replication by *prefetching* is also managed by clients. Traditionally, prefetching consists of loading pages succeeding the current one before a request for those pages has been made. It works well when accessing linearly structured data. Prefetching may also work for Web documents, provided we take into account how an individual document is organized, and possibly combine that with knowledge on reading habits of its clients. Finding a single prefetching strategy that can be used efficiently for all Web documents is doomed to fail.

Finally, *server replication*, by which a document is replicated across multiple servers, aims at load distribution and increase of performance and availability. There are many different criteria for server replication, even within a single document. Consider, for example, a national electronic newspaper having local editions. Many pages will be of interest to all readers and should be replicated nationwide, whereas other pages should be replicated and distributed only within certain regions. Likewise, some readers may have subscriptions that exclude, for example, sports pages or news reports in foreign languages.

To account for various forms of replication management, we adopt a model in which the files that contain the pages and other elements of a Web document are kept at three different kinds of **stores**:

Permanent stores implement persistence of a Web document. This means that if there is currently no client accessing the document, the document's content will be kept at its associated permanent stores. A (mirrored) Web site is an example of a permanent store. We assume a document's set of permanent stores changes only rarely, and can be considered more or less fixed.

Document-initiated stores are (dynamically) installed as the result of the document's replication management policy. Replicas are managed independently of clients. A typical example of a document-initiated store is a push cache, as proposed by Gwerztman and Seltzer.[4]

Client-initiated stores are comparable to caches. They are placed independently of the replication management of the document and fall under the regime of the client processes that read and write the document. A site-wide cache at a Web proxy is an example of a client-initiated store.

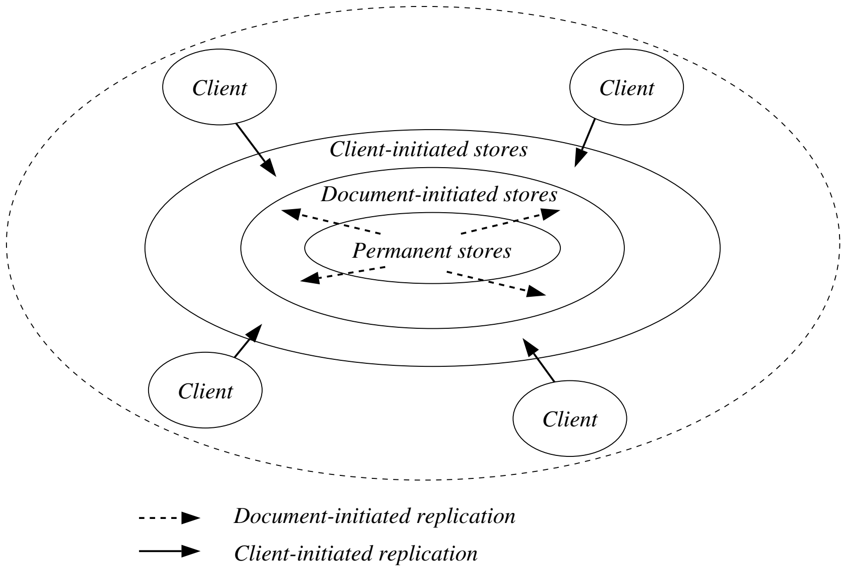

Figure 1: A system model for distributed and replicated Web documents.

Global storage is organized in a layered fashion as shown in Figure 1. The layers represent different spheres of replication management. The model allows us to separate management by servers (permanent and document-initiated stores) from that by clients (client-initiated stores). We note that in our model, clients are allowed to request Web documents directly from the permanent and document-initiated stores. In such cases, a client effectively decides not to make use of a local, private cache.

Deciding on a Coherence Model

After having decided on the placement of replicas, we need to decide on how replicas are to be kept consistent. A **coherence model** describes the effect of read and write operations by different clients on a possibly replicated document, as viewed by clients of that document. A **coherence protocol** describes an implementation of such a model. We make a distinction between coherence as offered by a document to its set of clients (document-centric coherence models), and coherence as required by a client (client-centric coherence models).

Document-centric Coherence Models

Current Web cache coherence protocols assume pages of a Web document are modified only by their owner. They provide no support for shared documents that allow concurrent updates by different clients. However, it is important that we can express the coherence of a document that is shared by concurrently operating clients. Many studies, especially in the distributed shared memory systems community, have addressed the definition of such coherence models. In the context of the Web, we believe the **document-centric coherence models** as shown in Table 1 are particularly relevant. These models are implemented by the permanent and document-initiated stores discussed before, and may be adopted by a client to client-iniated stores if required.

Table 1: Document-centric coherence models

Name	Description	Example
Sequential coherence	Global total ordering of operations on a document.	Collaborative editing
PRAM coherence FIFO coherence	The writes of each client appear on every store in the order in which they have been issued by that client	Web page replacement
Causal coherence	Ordering of operations must be guaranteed only between causally related operations.	Discussion groups
Eventual coherence	Updates are propagated such that eventually everyone has the latest version.	Personal home page

The sequential coherence model[5] is hard to implement efficiently in the Web as it requires global coordination. However, many documents such as, for example, those that are manipulated through collaborative editing, will actually need it.

The PRAM coherence model[6] does not require global coordination between neither stores or clients. The model can be implemented efficiently by tagging every update with a local sequence number. This model is particularly useful in the case of incremental updates to a document. For example, a conference Web document may be updated by successive replacement of one of its pages. It is important that the order in which replacements take place is maintained. However, in this case it is allowed to publish an update even if a previous one has been missed. In this **FIFO coherence model**, a write request from a client is honored if it is *more recent* than the latest write from that same client. Otherwise, the request is simply ignored.

The causal coherence model[7, 8] is used for documents in, for example, discussion groups, where posting a participant's reaction makes sense only if the client has received the message that triggered the reaction.

Finally, the eventual coherence model imposes only very limited ordering constraints. The issue is that if no updates take place for a long time, all replicas gradually converge to the same state as dictated by the last update.

Client-centric Coherence Models

A **client-centric coherence model** allows a client to express his own coherence requirements, more or less independent from the coherence as supported by the document, and independent from other clients. Such a model describes the effect of read and write operations by a *single* client on a replicated document, as viewed by that client. The underlying assumption is that a client may possibly access different replicas while browsing and interacting with a Web document. Client-centric coherence is useful when document-centric coherence is either too weak or too strong from a client's point of view. Client-centric coherence is applied separately for each client, possibly in combination with the document-centric coherence model.

Not much research has been conducted on client-centric models, except for the session guarantees developed in the Bayou system.[9] Bayou provides mobile users support for weak consistency in a replicated database. In the context of offering scalable Web documents, the Bayou models (shown in Table 2) can be retained with the following two extensions. First, we believe that a solution for scalable Web documents should *guarantee* coherence rather than only check whether the coherence requirements are satisfied, as is done in Bayou. Second, it should be possible to combine client-centric models with document-centric ones.

Table 2: Client-centric coherence models

Name	Description	Example
Client-PRAM coherence Client-FIFO coherence	Writes made by one specific client (eventually) appear at every replica in the same order in which they have been issued	Message streams
Client-causal coherence	For each replica, an operation by a specific client is carried out after all updates on which that operation causally depends, have taken place.	Posting a reaction to a bulletin board article, and ensuring that others see to what you react
Read-Your-Writes coherence	The effects of every write by a client are visible to all subsequent reads by that client	Updates on Web pages are always immediately visible in the local browser
Monotonic-Reads coherence	Two subsequent reads, possibly at different stores, are based on copies updated by all writes preceding the first read operation	Successive read operations at two Web browsers at different machines give the same result

Note that client-PRAM and client-causal coherence deal with propagating updates to different replicas. The other two models impose ordering constraints on what a specific client reads.

To illustrate the Read-Your-Writes coherence model, consider a Webmaster who writes directly to a Web server whereas all reads are performed from his cache. When the Web master updates a document, he must be able to check whether the write has been done correctly, that is the update must be immediately propagated to his cache. As he is the only writer of the document, he is the only client having this requirement.

As an illustration of Monotonic-Reads coherence, consider a Web page replicated at two different stores, one in Amsterdam and the second one in Paris. If a client first reads the page from the Amsterdam store and later again from Paris, then the second copy should be the same as the one read in Amsterdam, or an updated version thereof, but not an earlier version.

Discussion

Deciding on document-centric and client-centric coherence models are different issues, and adds complexity to solving scalability problems. In those cases that a client communicates with a permanent or document-initiated store, not every combination of the two types of coherence models makes sense. For example, if the document offers sequential consistency, then it automatically realizes every client-centric model as well. On the other hand, if only PRAM consistency is offered, a client may additionally decide to impose the Monotonic Reads model. Note, however, that in general a client-centric coherence model cannot be enforced: the document should be willing to support it as well.

Deciding on Coherence Protocols

Having decided on coherence requirements leaves open how to actually meet those requirements. A coherence protocol is a specific implementation of a coherence model. There may be several protocols for a single model. Which protocol is the best may depend on issues such as read/write ratios, the number of clients simultaneously accessing a document, etc. We have identified a set of **protocol parameters** that can be used to classify protocols. An overview of the most common parameters is shown in Table 3. It is primarily up to the developers of a Web document to decide on the protocol to be used for the document's coherence model.

To illustrate, at present, coherence for pages in the Web is generally maintained by a page's Web server in combination with caches at a client's Web browser and additional Web proxies. The current protocol can be characterized by the parameter values shown in Table 4.

A Web server is a permanent store that is not concerned with the state of proxy or browser caches, qualifying it as being passive. In general, a Web proxy can check the consistency of a cached page whenever it receives a read request. If necessary, it refreshes its cache entry by pulling in the entire page from the server. Browser caches are often configured by default to respond in a lazy fashion. A

Table 3: Parameters for classifying coherence 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 updated 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.
Store responsiveness	- immediate - lazy (e.g., periodic) - passive	Describes how quickly a store reacts when it notices it is no longer consistent with the other replicas. A passive store will do nothing.
Store reaction	- pull - push	Describes what a (non passive) store will do when it notices it is inconsistent with the other replicas. It will either push or request update messages.
Write set	- single - multiple	This parameter gives the number of writers that may simultaneously access the Web document.
Coherence group	- permanent only - permanent and document-initiated - all layers	Describes who implements the document-centric coherence model: permanent and/or document-initiated stores.

user can have some influence here, though. For example, many browsers can pull a page in either on each read, once per session, or never.

The choice of protocol parameter values is important since it may have a large effect on performance. For example, if a highly replicated Web document is often modified, it may be more efficient to implement a lazy server that aggregates several updates instead of a server that immediately responds to each update. In contrast, if the Web document is seldom modified a better solution may be to combine passive servers with client caches that respond immediately by requesting updates to be pulled in on each access.

We believe that the (default) choice for coherence protocols should be made

Table 4: Classification of the Web's general coherence protocol

Parameter	Web server	Proxy cache	Browser cache
Change distribution	full state		
Store responsiveness Store reaction	passive —	immediate pull	lazy pull
Write set	single		
Coherence group	Web server only		

by the developers of a document since they have the most knowledge about the semantics of the document. Whenever (part of) a document is cached by a user, the cache adopts the document's coherence protocol by default. Of course, the protocol parameters can be overridden by a user's preferences.

Controlling Complexity

We believe it is important that users are aware that many alternatives for implementing and accessing Web documents exist. However, the Web's success is entirely based on its elegant simplicity and it should be kept that way. Consequently, the Web's casual users cannot be made responsible for making the right choices. Realizing that any choice they make influences the attainable performance, we have to find ways of keeping complexity to a minimum.

In our model, we make a distinction between the *developers* of a Web document and the *clients* of that document. Developers are responsible for decisions concerning permanent and document-initiated stores, and document-centric coherence and its implementation. Clients, in principle, need to decide on local caching (i.e. client-initiated stores), and client-centric coherence and its implementation.

Controlling Complexity for the Developer

To a certain extent, one may expect from developers that they understand how their document could possibly be replicated and distributed, what type of coherence it requires, how it will be used, etc. In principle, the (professional) developer needs to make the following subsequent decisions:

1. Decide on what has to be replicated, and how replicas are to be managed
2. Choose appropriate coherence models for the document
3. Decide on the best coherence protocols

As we argue below, deciding on replication management and coherence protocols can be highly automated if we succeed in developing dynamically adaptive solutions. Understanding and deciding on coherence models may not be that easy, as has also been experienced by the developers of Isis.[10] A possible solution lies in a classification system for Web documents, and setting appropriate defaults for replication management and coherence protocols for each class.

Classifying Web Documents

A classification system for Web documents will help developers decide on an appropriate coherence model. In addition, for each class we need to find reasonable defaults for replication management and coherence protocols that can dynamically converge to optimal settings.

There are a number of obvious classes. For example, many Web documents are passive information sources, such as home pages of individuals, organizations, and special-interest groups. Assuming we know nothing a priori about the usage pattern, having only a single permanent store is a reasonable default. Client-side caching can initially be disabled. These information sources are generally maintained in such a way that updates never conflict. In other words, there is conceptually only a single writer. Moreover, we may assume that the update frequency is relatively low. Consequently, when in the course of time the document is stored at several places, an eventual document-centric coherence model should be sufficient. In addition, because a client will not be writing, no client-centric coherence is needed. Other defaults are shown in Table 5.

Table 5: Defaults for passive information sources

Replication	**Coherence**	**Protocol parameters**
Single permanent store	*document:* eventual *client:* none	*change distribution:* state differences *store responsiveness:* immediate *store reaction:* pull *write set:* single *coherence group:* all layers

Specific work in this area has also been done on electronic media distribution,[11] and collaborative editing for whiteboards [12] and text documents.[13] For example, many shared whiteboards tend to be transient, that is, exist only during a collaboration session. As soon as a client enters a session, he is required to store a copy of the whiteboard locally. There is no need for permanent and document-initiated stores. Also, global ordering is generally not needed, as users are not allowed to modify each other's drawings, making the PRAM coherence sufficient. Drawing operations can be immediately multicast to the current session members. These characteristics correspond to the parameter setting shown in Table 6.

Table 6: Defaults for transient documents with cooperating, independent writers

Replication	**Coherence**	**Protocol parameters**
Caching only	*document:* PRAM *client:* PRAM	*change distribution:* operation *store responsiveness:* immediate *store reaction:* push *write set:* multiple *coherence group:* all layers

Other classes easily come to mind. For example, collaborative text editing defines a class of persistent documents with cooperating writers that may change each other's work. This requires a sequential coherence model. A distinction should be made between documents that are circulated, and those that are simultaneously modified. In the latter case, we may also wish to distinguish cases that support only global modifications from those in which each participant works on a specific section of the document.

Dynamic Adaptations

Deciding on the best coherence model is something that perhaps only the developers and users of a Web document can determine. However, many alternatives may be initially ignored if it is possible to dynamically converge a configuration towards a good solution, starting at a reasonable default.

For example, suppose a system administrator initially choses to immediately update a site-wide proxy cache when noticing that its content was no longer valid. Also assume the coherence protocol initially uses a pull mechanism by which the cache requests a permanent store to provide it with the update. In the course of time, the frequency of updates may increase, and perhaps also the number of users sharing the proxy cache. In that case, a periodic update combined with pushing data to the cache is more efficient. We can avoid asking a developer or user to predict such changes, if we can be detect them automatically and dynamically adapt the protocol. In many cases, such an approach is definitely feasible, and in fact necessary, considering that, for example, system administration generally not even notices changes in usage patterns. The classification of Web applications could help setting the appropriate thresholds before a change in strategy is initiated.

Dynamic coherence protocols are still very much an open research issue. However, if we are to build a worldwide scalable Web, it is essential that such protocols are to be developed. Recent studies, such as conducted by the Oceans group,[14] show promising results.

Controlling Complexity for the Client

The majority of users is formed by those simply accessing existing Web documents. For this group of people, usage of the Web should be kept as simple as possible. Therefore, they inherit the default strategies specified by a document's developer. However, they should be allowed to change these defaults, for example, by disabling or enabling caching facilities, and imposing stronger or weaker client-centric coherence. Changing the defaults is also essential to avoid abuse by developers. Note that clients have no direct influence on the permanent and document-initiated stores, but are in full control of the client-initiated stores such as their caches.

Technical Support for Scalable Web Documents: Globe

An important starting point for research is providing an infrastructure that will allow us to at least attach distribution strategies and implementations to individual documents. Unfortunately, even this feature is missing in the current Web. What is needed is a mechanism that will (1) allow us to fully encapsulate a distribution policy inside a document, and (2) allow for several distribution strategies to coincide within a single document. The latter is needed to let users adopt their own client-centric coherence model, while at the same time other replicas maintain document-centric coherence.

As a first step in this direction, we are developing Globe,[15] a wide-area distributed system that is being implemented as a middleware layer on top of the Internet. At the core of Globe lies the concept of a **distributed shared object**. Such an object is physically distributed across multiple machines, meaning that an object's state may be partitioned and replicated across multiple machines at the same time. Each object fully encapsulates its own distribution policy. In other words, all details concerning, for example, the distribution, replication, and migration of the object's state, are completely hidden from the object's clients.

A restricted version of Globe is being developed to support Web documents.[16] A Globe Web Document (called a GlobeDoc) consists of a collection of HTML pages, together with files for images, audio, video, icons, applets, etc., which are called **page elements**. The hyperlinked structure as normally provided by Web pages is maintained in a GlobeDoc. When a client accesses a GlobeDoc, a local implementation of the document is installed at the client. Such an implementation is called a **local object**. The local object offers a standard interface to the client, as shown in Table 7.

Table 7: Interfaces offered by each GlobeDoc

Interface	**Contains methods for** ...
Document interface	Listing, adding, and removing page elements
Content interface	Reading and writing the content of a page element
Attribute interface	Attributes of page elements: type, last modification date, etc.

Our model of distributed shared objects is shown in Figure 2. A local object resides in a single address space and communicates with local objects in other address spaces. It forms a particular implementation of an interface of the distributed shared object. For example, in the case of GlobeDocs, a local object may implement an interface by forwarding all method invocations to a central location where the page elements of the document are stored, as in RPC client stubs. However, a local object in another address space may implement that same interface through operations on local replicas of those elements. Such implementation details are transparent to the client processes: they see only the interface to the distributed object as offered by the local object. Each local object is composed of several subobjects, and is itself again fully self-contained as also shown in Figure 2. A minimal composition consists of the following four components.

Semantics subobject. This is a local subobject that implements (part of) the actual

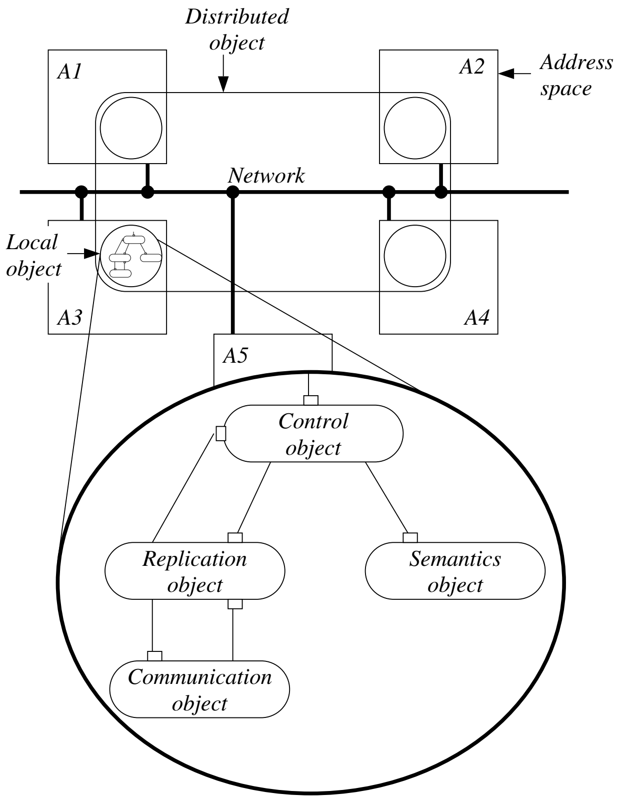

Figure 2: Example of an object distributed across four address spaces.

semantics of the distributed shared object.

Communication subobject. This is generally a system-provided subobject. It is responsible for handling communication between parts of the distributed object that reside in different address spaces.

Replication subobject. The replication subobject is responsible for keeping the replicated semantics subobjects consistent according to some (per-object) coherence model. Different distributed objects may have different replication objects, using different replication algorithms.

Control subobject. The control subobject takes care of invocations from client processes, and controls the interaction between the semantics subobject and the replication subobject. Incoming invocation requests are also handled by the control subobject.

An important observation is that communication and replication subobjects are unaware of the methods and state of the semantics subobject. Replication subobjects of different local objects are independent and each local object within a distributed object is able to implement a different coherence model.

Globe, and in particular its Web-based version GlobeDoc, is currently being implemented in Java. We initially developed a simple prototype version to test the feasibility of our approach. We are now working towards an implementation that will allow us to do large-scale experiments on the Internet.

Conclusions

If we are to build a scalable Web, we have to provide support for replication, different coherence models, and different coherence protocols. Equally important, we need an infrastructure that allows us to tailor distribution to each indidividual Web document. In that sense, current Internet research is still only in a preliminary stage when it comes to finding truly scalable solutions.

We have identified and analyzed the issues that need be addressed for building a scalable Web. The main problem we have to deal with is that replication and coherence must be taken into account by those that develop and use Web documents. This is an unfortunate situation as it may turn the Web into an intricate, hard-to-use system. Therefore, besides having an infrastructure that can support distribution on a per-document basis, we need to find the means to reduce the many alternatives to choose from. We advocate that more research needs to be targeted towards classifying Web documents with the aim to provide reasonable default distribution policies. At the same time, effort should be put into developing dynamically adaptive coherence protocols.

References

1. B. Neuman. "Scale in Distributed Systems." In T. Casavant and M. Singhal, (eds.), *Readings in Distributed Computing Systems*, pp. 463–489. IEEE Computer Society Press, Los Alamitos, CA., 1994.
2. M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm. "Enhancing the Web's Infrastructure: From Caching to Replication." *IEEE Internet Comput.*, 1(2):18–27, Mar. 1997.
3. A. Bestavros, B. Carter, M. Crovella, C. Cunha, A. Heddaya, and S. Mirdad. "Application-level Document Caching in the Internet." In *Proc. Second Int'l Workshop on Services in Distributed and Network Environments*, pp. 166–173, Whistler, Canada, June 1995.
4. J. Gwertzman and M. Seltzer. "The Case for Geographical Push-Caching." In *Proc. Fifth HOTOS*, Orcas Island, WA, May 1996. IEEE.
5. L. Lamport. "How to Make a Multiprocessor Computer that Correctly Executes Multiprocessor Programs." *IEEE Trans. Comput.*, C-29(9), Sept. 1979.
6. R. Lipton and J. Sandberg. "PRAM : A Scalable Shared Memory." Technical Report CS-TR-180-88, Princeton University, Sept. 1988.
7. M. Ahamad, M. Raynal, and G. Thia-Kime. "An Adaptive Protocol for Causally Consistent Distributed Services." In *Proc. 18th Int'l Conf. on Distributed Computing Systems*, pp. 86–93, Amsterdam, May 1998. IEEE.
8. P. Hutto and M. Ahamad. "Slow Memory: Weakening Consistency to Enhance Concurrency in Distributed Shared Memories." In *Proc. Tenth Int'l Conf. on Distributed Computing Systems*, pp. 302–311. IEEE, 1990.
9. D. B. Terry, A. J. Demers, K. Petersen, M. J. Spreitzer, M. M. Theimer, and B. B. Welsh. "Session Guarantees for Weakly Consistent Replicated Data." In *Proc. Third Int'l Conf. on Parallel and Distributed Information Systems*, pp. 140–149, Austin, TX, Sept. 1994. IEEE.
10. K. Birman and R. van Renesse, (eds.). *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, Los Alamitos, CA., 1994.
11. J. Donnelley. "WWW Media Distribution via Hopwise Reliable Multicast." *Comp. Netw. & ISDN Syst.*, 27(6):781–788, 1995.
12. S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang. "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing." *IEEE/ACM Trans. Netw.*, 5(6):784–803, Dec. 1997.
13. C. Ellis, S. Gibbs, and G. Rein. "Groupware – Some Issues and Experiences." *Commun. ACM*, 34(1):38–58, Jan. 1991.
14. A. Bestavros. "WWW Traffic Reduction and Load Balancing through Server-Based Caching." *IEEE Concurrency*, 5(1):56–67, Jan. 1996.
15. M. van Steen, P. Homburg, and A. Tanenbaum. "The Architectural Design of Globe: A Wide-Area Distributed System." *IEEE Concurrency*, 7(1), Jan. 1999. Scheduled for publication.
16. M. van Steen, A. S. Tanenbaum, I. Kuz, and H. J. Sips. "A Scalable Middleware Solution for Advanced Wide-Area Web Services." In *Proc. Middleware '98*, pp. 37–54, The Lake District, England, Sept. 1998. IFIP.