VRIJE UNIVERSITEIT AMSTERDAM

VU Research Portal

Adaptive Replicated Web Documents.

Pierre, G.E.O.; Kuz, I.; van Steen, M.R.

2000

document version
Publisher's PDF, also known as Version of record

Link to publication in VU Research Portal

citation for published version (APA)
Pierre, G. E. O., Kuz, I., & van Steen, M. R. (2000). *Adaptive Replicated Web Documents.* (IR; No. 477). Computer Systems.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

E-mail address:
vuresearchportal.ub@vu.nl

Download date: 15. Oct. 2021

Adaptive Replicated Web Documents

Guillaume Pierre
Ihor Kuz
Maarten van Steen

Internal report IR-477
September 2000

Abstract

Caching and replication techniques can improve latency of the Web, while reducing network traffic and balancing load among servers. However, no single strategy is optimal for replicating all documents. Depending on its access pattern, each document should use the policy that suits it best. This paper presents an architecture for adaptive replicated documents. Each adaptive document monitors its access pattern, and uses it to determine which strategy it should follow. When a change is detected in its access pattern, it re-evaluates its strategy to adapt to the new conditions. Adaptation comes at an acceptable cost considering to the benefits of per-document replication strategies.

Faculty of Mathematics and Computer Science

1 Introduction

Most Web users suffer from slow document transfers. The reasons for such high latencies include distance between the user and the document, and load of the intermediate network. One common solution is to maintain copies of documents across the Internet. A user's requests are directed to a nearby copy, which reduces user-perceived latency and wide-area network load. Several strategies can be used for copying documents. *Caching strategies* create a copy when a user first requests the document; subsequent requests can be resolved locally. Cached copies can be destroyed at any moment, for example to reclaim storage space. *Replication strategies* create copies *a priori* as opposed to cached copies. Such replicas are intended to persist for a long time.

When a document is modified, its copies must either be updated or destroyed so that users do not access stale data. Most caching policies apply diverse heuristics to guess when the original document has been updated. When a cache suspects that a copy is stale, it can destroy it or check for its validity. Replicas use a different strategy: they assume that the main server sends a notification when an update occurs.

Which of these strategies is the most efficient for minimizing latency and network usage, while keeping copies consistent? In a previous paper we have shown that no single strategy is optimal in all cases. Instead, each document should select the strategy that suits it best, depending on the requests it receives [15]. Our method for selecting a policy for a given document relies on trace-based simulations. First, we collect traces of every request to the document as well as every update. We then replay the trace in a simulator to reproduce the behavior of each candidate policy; each simulation outputs the cumulative latency, the consumed wide-area network bandwidth and the number of stale copies delivered to clients. Finally, we evaluate each policy by way of a cost function; the "best" policy being the one with the lowest cost.

We have shown that differentiating strategies provides a significant performance improvement over any one-size-fits-all strategy [15]. The strategy selection method relies on *a posteriori* analysis of traces. That is, it allows one to determine what the optimal strategy for a document would have been. However, we have also shown that a selected policy is stable, in the sense that a choice for a policy does not change as long as the usage pattern for a document remains the same. In other words, we can use past traces to determine an optimal policy for the near future.

These results suggest that it is possible to build adaptive replicated documents. However, we did not yet address the question of how to realize dynamic strategy selection in practice. This paper makes up for this omission and proposes an architecture for adaptive replicated Web documents. Based on its own access traces, an adaptive document selects the policy that currently suits it best. A document continuously monitors changes in the access pattern and periodically re-evaluates its choice of policy. If needed, it dynamically changes its replication strategy.

The paper is structured as follows: Section 2 describes the general model of an adaptive document; Section 3 details the practical issues of adaptation; Section 4 describes the architecture of an adaptive document; Section 5 provides a performance evaluation. Finally, Section 6 presents some related work, and Section 7 concludes.

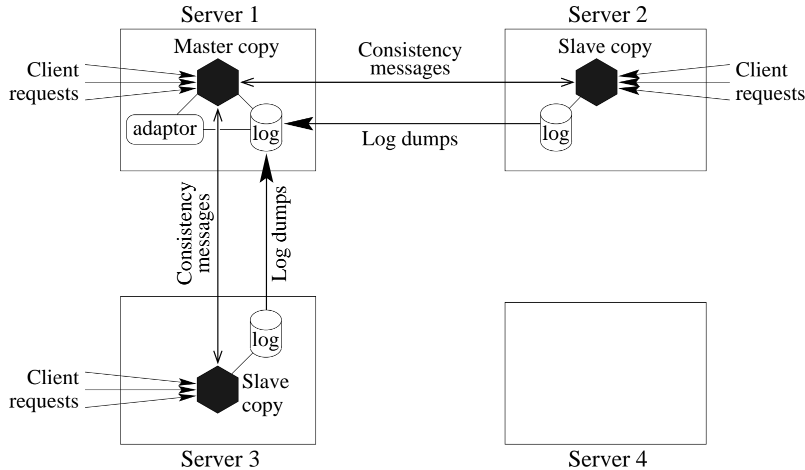

Figure 1: Structure of a Distributed Web Document

2 General Model

Most current Web research considers caching and replication as an operation that one applies to a document. In other words, caching and replication are *external* to the documents. Such an approach makes document-specific replication policy decisions difficult to implement. We take a different point of view: we consider that a document and all its copies across the Internet constitute a single entity, called a *distributed Web document*. Considering all instances together as a whole eases taking per-document decisions, such as the choice of a replication policy.

A distributed Web document is composed of several copies located at different servers. It follows a master/slave organization: there is one master copy, whose location is stable. The author of the document directly accesses it to edit its content. Slave are read-only copies that are created and deleted depending on the replication strategy. Figure 1 shows the structure of a simple distributed Web document. The master is located at server 1, with slaves at servers 2 and 3.

Clients can send requests to any copy. A location service allows clients to find the closest copy of a document [18].

Copies communicate with each other to maintain consistency. This communication includes downloading a document's content when creating a copy, `If-Modified-Since` requests, invalidations, etc. The details of such communication depend on the replication policy being used by the document.

Figure 1 also shows logs and an adaptor component. The adaptor is responsible for selecting the most appropriate replication strategy and for deciding when the current strategy should be changed. It bases its decision on logs that are collected at every copy and merged at the master.

2.1 The Adaptor

From time to time, the master decides to re-evaluate its choice of replication policy. It does so by looking at its most recent log data and by simulating several alternative

replication policies. A cost function applied to the simulation results helps it to decide on a new policy. Details about evaluating simulation results can be found in [15].

When a decision has been made, the master must propagate that decision to every slave it knows about (e.g., those that have registered for invalidation). Slave copies that the server does not know about continue using the previous policy; however, each time one contacts the master, it receives information about which policy it should currently follow. This scheme allows for progressive dissemination of the new policy.

2.2 Log File Management

The copies of a given document must collect traces of their activity for two reasons: first, we believe that the owner of a document should be allowed to obtain logs of every request, independently of which instance treated it. This may encourage the use of caching and replication, even for sites whose revenue depend on their popularity [16]. Second, access traces must be centralized at the master site of the document to enable replication policy selection.

Each copy of a document keeps a log of the requests it receives. Periodically, the tail of the log is sent to the master. Sending a log entry to the master is delayed at most 10 minutes, which guarantees that the master's view of the logs is at most 10 minutes late. We estimate that this limit is adequate for piggybacking log entries while allowing responsive adaptations of the current policy as access patterns change. However, copies can send log data more often if they wish, for example to reclaim storage space.

The master writes the log data it receives directly to disk. Since it can receive log data from several sites, the master log file is not sorted in chronological order. We believe that this is not a serious issue for the document owner. However, the simulator used for re-evaluating the current policy requires its input traces to be sorted. Therefore, when a re-evaluation takes place, the first operation is to re-read the last N requests from the master log file and sort them into a trace usable by the simulator.

3 Making Adaptation Responsive

Given an infrastructure capable of collecting traces and deciding which replication policy should be used, we still need to decide when a document should re-evaluate its strategy. Doing so too often would waste computing resources, while re-evaluating too rarely would decrease performance.

3.1 Deciding When to Adapt

The simplest scheme for adaptation is to re-evaluate the replication strategies at fixed time intervals, such as once a week. However, this approach does not allow a document to react quickly to sudden changes in access patterns. It would be more efficient to adapt as soon as an access pattern changes.

To decide when to adapt, the system monitors a number of variables such as frequency of requests and average response time. Significant variation of these variables is a sign that something is currently changing in the system and that adaptation may be necessary.

Variables are computed using a standard technique: at startup, each variable V is initialized to the value from a first sample. Each time a new sample s is taken, V is updated using the following formula:

$$V := \alpha.s + (1-\alpha)V$$

The formula uses a parameter α, which controls the relative weight given to a new sample with respect to the previous sequence of samples.

Each time an adaptation takes place, low and high watermarks such as $V/2$ and $V\times 2$ are set up for each variable. If the value of V later reaches one of these watermarks, we estimate that the access pattern may have changed enough for the current policy not to be optimal any more. The current strategy must then be re-evaluated.

A problem that must be solved is where to monitor the variables. One possibility is that the master copy of a document does all the necessary computations. However, this would not be very practical, since variables can be computed only from the log data sent by copies. Since the master receives log data in non-chronological order, computing a sequential history of a variable would become quite complex. Instead, each copy computes variables locally, and transmits their value to the master together with the log data. The master does not compute a single value, but keeps the variables separate.

The master monitors all the variables received from its slaves. Because many of the variables account only for a small fraction of the overall traffic, one variable reaching its watermark does not necessarily mean that a significant change is occuring. On the other hand, if several variables reach their watermarks within a small time interval, it is likely that a real change in the access patterns has occured. To prevent "false alarms" from being triggered, the master waits until a sufficient number of variables reach a watermark before starting a re-evaluation.

3.2 The Case of a Flash Crowd

Sometimes, a copy cannot wait for the master to re-evaluate policies. For example, during a flash crowd there is a sudden and significant increase in the number of requests received by a document. Such events may occur, for example, when the document gets linked to by a popular information site [1]. In such cases, the load increase on a copy may deteriorate not only the response time of the document, but also that of every other document hosted by the same site. This is clearly not acceptable.

When a copy is requested often enough to significantly deteriorate the quality of service of its host site, it can decide to adapt by itself without requiring its master to re-evaluate policies. This allows fast responsiveness in case of a flash crowd.

The reaction consists of creating more copies to handle the load. Although sharing the load among several copies may solve the overload problem, such a trivial adaptation is likely not to be optimal. Therefore, an alarm is sent to the master requesting it to re-evaluate the overall replication strategy as soon as possible.

4 Encapsulating a Replication Policy in a Document

The architecture of the adaptive replicated documents is based on Globe, a system for large-scale distributed shared objects [19]. Physically, objects are distributed, with ac-

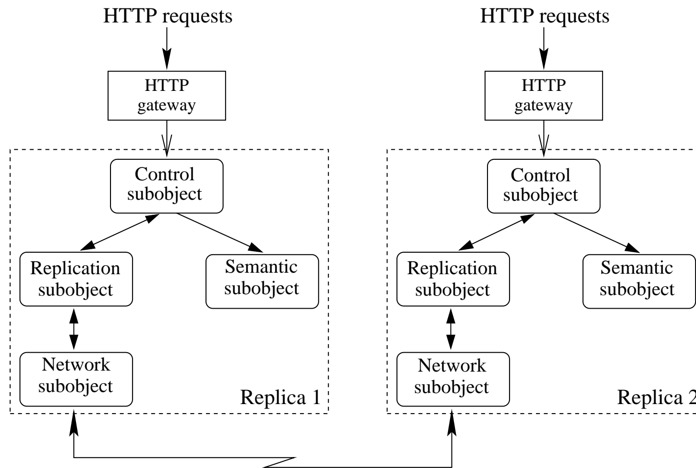

Figure 2: Architecture of a Globe Distributed Object

tive copies on multiple machines at the same time. Distributed copies use peer-to-peer communication: an application loads the object implementation in its address space to participate in the distributed object. Users may contact any copy to have methods performed, but they know nothing about the internal structure and protocols used inside the object. This scheme allows different objects to transparently use different algorithms for data partitioning, replication, consistency, and fault tolerance.

4.1 Basic Architecture

Replicated Web documents can be considered as a special case of Globe objects. Figure 2 shows the structure of a two-replica document. Each *semantic subobject* contains the state of the object, as well as meta-data such as a MIME type and a date of last modification. To access the document, invocations must be sent locally to the *control subobject*. Before performing the invocation, the control subobject notifies the *replication subobject* so that it can check for consistency before the request is honored. The replication subobject may use the *network subobject* to communicate with peer replicas. Finally, each document is made accessible by normal Web browsers using an HTTP gateway.

A worldwide directory service is also provided to locate servers in given regions of the network. It allows a document to discover replica servers close to the clients where the most requests come from. Describing the design of this service is beyond the scope of this article.

4.2 Building Adaptive Documents

The Globe object architecture allows one to select the replication policy of each document by choosing the proper replication subobject. However, an adaptive document must also be capable of collecting traces of past requests, evaluating candidate strategies, and dynamically loading the optimal one. To do so, the replication subobject is itself decomposed into three parts: one part is in charge of mechanisms (which are

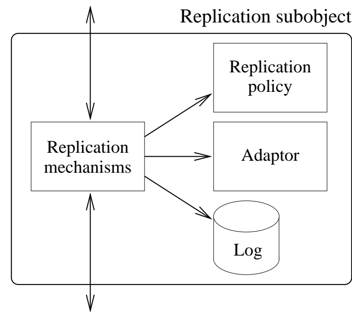

Figure 3: Internal Structure of a Replication Subobject

common to all replication strategies), another part implements the current replication strategy, and the third part is the adaptor presented in Section 2.1.

Figure 3 shows the internal structure of a replication subobject. The replication policy object maintains information about the object's consistency, such as the date of last modification and the date of the last consistency check. Each time a request is issued, the replication mechanisms object transmits the characteristics of the request to the policy object. Based on its implementation, the policy object responds by indicating how to treat the request: answer immediately, send an `If-Modified-Since` request to the server before answering, etc. The mechanisms object is in charge of actually performing the operation. The policy object can also directly receive incoming network messages, such as a notification that the document has been updated.

The mechanisms object is in charge of collecting log data and transmitting them to the master. It also transmits the monitoring variables to the adaptor. Based on these variables, the adaptor decides whether an adaptation should take place. It then sorts the most recently received logs, runs simulations, and informs the mechanisms object of the new optimal policy. Once a policy change has been decided, the mechanisms object replaces the policy object by one that implements the new policy.

Although adaptations take place at the master, each slave also has an adaptor module. This adaptor is used only to detect flash crowds and create new replicas to handle the sudden load.

5 System Evaluation

Compared to traditional Web documents, adaptive documents must perform additional operations: collecting logs, centralizing them at the master and running simulations. Although adaptive documents incur a cost, we expect this cost to remain small compared to the performance increase that per-document replication strategies will provide. In the following, we take a look at the overhead introduced by our approach.

5.1 Overhead Due to Trace Collection

The overhead due to trace collection is two-fold. First, it introduces additional computations at each copy; second, it generates network traffic for centralizing log data at the master.

Table 1: Adaptive Document Profiling

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

To evaluate the computational costs, we built a small prototype system for adaptive documents. We used it to replay trace files of accesses to our university's Web server. We emulated a complete Internet setup by running the prototype on a 200-node cluster of workstations [5]. Each node represented an Autonomous System in the Internet. Simulated clients located at these nodes sent requests to a number of adaptive replicated documents. We profiled each copy to get an idea of how much time is spent in each function.

Table 1 shows the amount of time that the program spends in its different modules. Network I/O operations account for most of the computation. Logging adds up to 9% of the processing time at each slave. As the master must log the requests that are addressed to it as well as the log data sent by slaves, it requires more time, up to 15%. We consider these figures to be acceptable.

Each log data message contained on average data about 12 requests, and contained about 25 bytes of data per logged request. However, one can expect that this figure is highly dependent on the number of requests that the document receives every 10 minutes.

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

5.2 Overhead Due to Simulations

Adapting the replication policy of a document requires running as many simulations as there are candidate policies. Simulations are trace driven, which means that they execute roughly in linear time compared to the number of requests in the trace. To save computing resources, traces should be kept as small as possible. On the other hand, short traces may not reliably represent the access pattern to a document. Increasing the length of the trace improves the accuracy of the prediction. Therefore, one must find a tradeoff between trace size and accuracy.

To evaluate the accuracy of the prediction, we used a trace collected at our university between 13 September 1999 and 18 December 1999. This trace contains every request received by documents in our Web server, as well as every document update. We selected only the 98 documents having received at least 5000 requests during this period. For each of these documents, we split the trace into chunks of N requests. We simulated each of the trace chunks with different replication policies. If the "best"

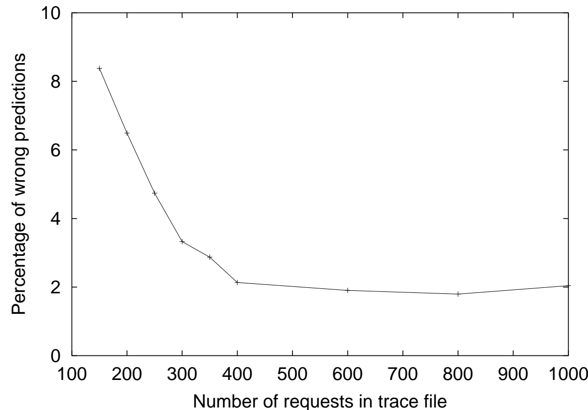

Figure 4: Proportion of Wrong Predictions vs. Trace Size

policy of chunk n is the same as the "best" policy of chunk $n+1$, then the prediction made at time n is assumed to have been correct.

Figure 4 shows the incorrect predictions when the chunk size varies. We can see that short trace files lead to many incorrect predictions. However, as trace size grows, the proportion of error decreases and stabilizes around 2%. Therefore, there is no point using long traces, since no additional accuracy will be gained. A reasonable trace size would be, for example, 500 requests.

We measured the computation time required by simulations on a 600 MHz Pentium-III workstation. Each simulated request took about 28 μs. So, for example, simulating a 500-request trace over 10 different configurations takes about 140 ms of CPU time.

6 Discussion and Related Work

Studying past access patterns to optimize the future behavior of the system is not a new idea in the Web community. Services such as prefetching, for example, rely on past access analysis to determine which documents are worth downloading [6, 10, 13]. Other systems dynamically organize the search path for a URL among a cache mesh based on a shared knowledge of caches' contents [11]. Finally, certain replacement algorithms base their decisions on temporal correlations between requests [7]. However, all these types of adaptation are fundamentally based on optimizing a single strategy. Instead, we propose to use several different policies, and to dynamically select the one most suited for each document.

We make a distinction between *flexible* and *adaptive* systems. We define a flexible system as being capable of dynamically loading new policies at run-time. An adaptive system is a flexible system which can, in addition, automatically determine which policy should be used.

Flexible systems have been developed in many domains. Flexible group communication systems, such as x-kernel [12] and Horus [17], split protocols into elementary modules that can be composed together to obtain the required features. Flexible replication systems allow one to choose which replication policy should be used [3, 8]. The

same principle has been applied for building routers [9], network traffic analyzers [14], etc.

Examples of adaptive systems are content distribution networks, such as Akamai [2] and Digital Island [4]. These systems adapt the number and location of document copies to provide copies close to the users' locations and to accommodate the load. However, the lack of available technical documentation prevents us from making any detailed comparison with our approach.

Our approach requires us to consider documents as objects, instead of data. This allows the encapsulation of replication policies *inside* each document. Of course, this fundamental change prevents current Web servers and caching proxies from hosting adaptive documents. A new platform is necessary. We have built a prototype of this platform but, since it does not support traditional Web proxying, using it at client sites would not be practical. Instead, we plan to use it as a base for a content distribution network. With all hosting sites being under the same administrative domain, deployment should be easier. Doing this will provide users with adaptive distributed documents in a transparent manner.

7 Conclusion

We have described an architecture for adaptive distributed Web documents. By considering a Web document and all its copies across the Internet as a whole, it becomes possible to follow document-specific policies. Document copies transmit their access logs to their master, which enables it to simulate several policies and to select the best one. We have described mechanisms to make documents adapt their policy as soon as a change in the access pattern is detected.

Adapting the replication policies allows one to exploit a document's access pattern to improve its performance. Adaptation comes at a cost that we consider low compared to the benefits of using per-document replication strategies.

References

[1] Stephen Adler, *The Slashdot effect – an analysis of three Internet publications*, `http://ssadler.phy.bnl.gov/adler/SDE/SlashDotEffect.html`, 1999.

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

[3] Georges Brun-Cottan and Mesaac Makpangou, *Adaptable replicated objects in distributed environments*, Tech. Report 2593, INRIA, Rocquencourt (France), May 1995, `http://www-sor.inria.fr/publi/RCMFGCCOSDS_rr2593.html`.

[4] *Digital Island*, `http://www.digisle.com/`.

[5] Henri E. Bal et. al, *The distributed ASCI supercomputer project*, `ftp://ftp.cs.vu.nl/pub/amoeba/orca_papers/das.ps.Z`, July 2000.

[6] Hiroyuki Inoue, Kanchana Kanchanasut, and Suguru Yamaguchi, *An adaptive WWW cache mechanism in the AI3 network*, Proceedings of the INET '97 conference, June 1997.

[7] Shudong Jin and Azer Bestavros, *GreedyDual* Web caching algorithms: Exploiting the two sources of temporal locality in Web request streams*, Proceedings of the 5th International Web Caching and Content Delivery Workshop, May 2000.

[8] Jürgen Kleinöder and Michael Golm, *Transparent and adaptable object replication using a reflective java*, Tech. Report TR-I4-96-07, Universität Erlangen-Nürnberg, September 1996, http://www4.informatik.uni-erlangen.de/TR/TR-I4-96-07.abs.html.

[9] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and Frans Kaashoek, *The Click modular router*, ACM Transactions on Computer Systems **18** (2000), no. 4.

[10] Evangelos P. Markatos and Catherine E. Chronaki, *A top 10 approach for prefetching the Web*, Proceedings of the INET '98 conference, July 1998.

[11] Scott Michel, Khoi Nguyen, Adam Rosenstein, Lixia Zhang, Sally Floyd, and Van Jacobson, *Adaptive Web caching: towards a new global caching architecture*, Computer Networks And ISDN Systems **30** (1998), no. 22-23, 2169–2177.

[12] Sean W. O'Malley and Larry L. Peterson, *A dynamic network architecture*, ACM Transactions on Computer Systems **10** (1992), no. 2, 110–143.

[13] Venkata N. Padmanabhan and Jeffrey C. Mogul, *Using predictive prefetching to improve World-Wide Web latency*, Proceedings of the ACM SIGCOMM '96 Conference (Stanford University, CA), July 1996.

[14] Simon Patarin and Mesaac Makpangou, *Pandora : A flexible network monitoring platform*, Proceedings of the USENIX 2000 Annual Technical Conference (San Diego), June 2000.

[15] Guillaume Pierre, Ihor Kuz, Maarten van Steen, and Andrew S. Tanenbaum, *Differentiated strategies for replicating Web documents*, Proceedings of the 5th International Web Caching and Content Delivery Workshop, May 2000.

[16] James Pitkow, *In search of reliable usage data on the WWW*, Proceedings of the Sixth International World-Wide Web Conference, April 1997.

[17] Robbert van Renesse, Kenneth P. Birman, and Silvano Maffeis, *Horus, a flexible group communication system*, Communications of the ACM **39** (1996), no. 4, 76–83.

[18] Maarten van Steen, Franz J. Hauck, Philip Homburg, and Andrew S. Tanenbaum, *Locating objects in wide-area systems*, IEEE Communications Magazine **36** (1998), no. 1, 104–109.

[19] Maarten van Steen, Philip Homburg, and Andrew S. Tanenbaum, *Globe: A wide-area distributed system*, IEEE Concurrency **7** (1999), no. 1, 70–78.