Globule: a Platform for Self-Replicating Web Documents

Guillaume Pierre and Maarten van Steen

Vrije Universiteit Amsterdam
Department of Mathematics & Computer Science
{*gpierre,steen*}*@cs.vu.nl*

Abstract. Replicating Web documents at a worldwide scale can help reduce user-perceived latency and wide-area network traffic. This paper presents the design of Globule, a platform that automates all aspects of such replication: server-to-server peering negotiation, creation and destruction of replicas, selection of the most appropriate replication strategies on a per-document basis, consistency management and transparent redirection of clients to replicas. Globule is initially directed to support standard Web documents. However, it can also be applied to stream-oriented documents. To facilitate the transition from a non-replicated server to a replicated one, we designed Globule as a module for the Apache Web server. Therefore, converting Web documents should require no more than compiling a new module into Apache and editing a configuration file.

1 Introduction

Large-scale distributed systems often address performance and quality-of-service (QoS) issues by way of caching and replication. In the Web, attention has traditionally concentrated on caching and in much lesser extent to replication. Recently, Web hosting services such as Akamai and Digital Island have started to emerge as the solution to achieve scalability through replication. In this approach, content is replicated to places where user demand is high. Content itself can vary from simple static pages to bandwidth-demanding video streams.

Web hosting services have the advantage over static mirroring of Web sites in that decisions on *what* content to replicate, and *where* replicas should be placed can be made automatically. However, current solutions generally do not differentiate how copies are to be kept consistent. In other words, once the decision is made to replicate, the same replication protocol is applied in all cases. In essence, no distinction is made between the type of content, nor are different access patterns taken into account. In many cases, static HTML documents are treated the same as large files containing, for example, audio- or videodata.

Many protocols have been proposed to achieve caching or replication, each of which presents specific advantages and drawbacks. However, as we have shown in a previous article, there is no single policy that is best in all cases [11]. This statement is true even for simple Web documents that are constructed as a static collection of logically related files. Typically, files in such documents contain HTML text, images, icons, and so on. When dealing with complex documents, such as those also containing streaming data,

or which are (partly) generated on request, differentiating policies becomes even more important.

As a consequence, we can expect to see a myriad of protocols that need to co-exist in a single system. One approach is to build multiprotocol servers, preferably following techniques that allow future extension. However, we believe such an approach is not sufficient. As an alternative, we propose to establish integration by following an object-based approach, in which a document is encapsulated in an object that is fully responsible for its own distribution. In other words, an object should not only encapsulate its state and operations, but also the implementation of a replication or distribution policy by which that state is delivered to clients.

To examine the feasibility of our approach we have been concentrating on Web documents, which we represent as distributed, self-replicating objects. Our approach allows a document to monitor its own access patterns and to dynamically select the replication policy that suits it best. When a change is detected in access patterns, it can re-evaluate its choice and switch policies on the fly [12].

Although we have demonstrated the potential merits of self-replicating documents, we have not yet addressed their practical implementation. There are two problems that need to be addressed. First, Web servers need to be adapted so that they can support adaptive per-document replication policies. Second, servers need to cooperate to allow replicas to be dynamically installed and removed, and to redirect clients to the nearest replica. One additional requirement that we feel is justified, is that adaptations should fit into the current Web infrastructure, requiring minimal modifications to existing servers and no modification at all to clients.

This paper presents the design of Globule, a platform for hosting adaptive Web documents that encapsulate their own distribution and replication protocol. It is designed as a module for the popular Apache server. Making use of our approach should require no more than compiling a new module into Apache and editing a configuration file. Globule handles all management tasks: discovering and negotiating with remote sites for hosting replicas; replicating static and some dynamic documents; and transparently redirecting clients to their closest replica.

The paper is structured as follows: Section 2 describes our document and server models; Section 3 details the architecture of the system, and Section 4 shows how such a system can be implemented as an Apache module. Finally, Section 5 presents related work and Section 6 concludes.

2 General Model

Our system is made of servers that cooperate in order to replicate Web documents. This section describes our document and server models.

2.1 Document Model

In contrast to most Web servers, we do not consider a Web document and its replicas only as a collection of files. Instead, we take a more general approach and consider

a document as a physically distributed object whose state is replicated across the Internet. The fact that the object's state is distributed is hidden from clients behind the object's interfaces. There is one standard interface containing methods such as `put()` and `get()` to allow for delivering and modifying a document's content. Special interfaces may be provided as well. For example, a document containing multimedia data may offer an interface for setting a client's QoS requirements before delivery takes place.

Our system is based on Globe, a platform for large-scale distributed objects [18]. Its main novelty is the encapsulation of issues related to distribution and replication *inside* the objects. In other words, an object fully controls how, when, and where it distributes and replicates its content. We have even designed documents that can dynamically select their own replication policy. Our "documents-are-objects" model is also a key to replicating dynamic or streaming documents.

Adaptive Replicated Web Documents We have shown in previous papers that significant performance improvements can be obtained over traditional replicated servers by associating each document with the replication strategy that suits it best [11, 12]. Such per-document replication policies are made possible by the encapsulation of replication issues inside each document.

The selection of the best replication policy is realized internally to each document by way of trace-based simulations. Replicas transmit logs of the requests they received to their master site. At startup or when a significant access pattern modification is detected, the master re-evaluates its choice of replication strategy. To do so, it extracts the most recent trace records and simulates the behavior of a number of replication policies with this trace. Each simulation outputs performance metrics such as client retrieval time, network traffic and consistency. The "best" policy is chosen from these performance figures using a cost function. More details about these adaptive replicated documents can be found in [12].

Replicating Dynamic Web Documents Many documents are not made from static content, but are generated on the fly. For each request, a Web server executes a request-specific piece of code whose output is delivered to the client. This code can in turn access external resources such as databases, execute shell commands, and issue network requests for generating a *view* of the dynamic document.

Replicating dynamic documents requires replicating the code as well as all data necessary for its execution (databases, etc.). This can be done by encapsulating the code and the databases in replicated distributed objects. Every request sent to the database is intercepted by the encapsulating object before being executed. Following the object's replication policy, if the request is likely to modify the internal state of the database, then it is propagated and applied to the other replicas in order to maintain a consistent state [18].

Our object model does not differentiate between static and dynamic documents. Therefore, dynamic documents are considered as objects implementing the same interface as static documents, but they differ in their implementation: static documents

use always the same implementation to access various internal states, whereas dynamic documents differ in both their internal states and method implementations.

The main issue with respect to dynamic documents arises when converting existing dynamic documents into objects. The Web server must be able to determine automatically which resources, files or databases are accessed by each dynamic document in order to include them inside the object. This can be difficult for documents such as CGIs, where the server delegates request handling to an arbitrary external program. However, a large portion of dynamic documents such as PHPs and ASPs are in fact scripts interpreted by the Web server itself. In this case, the server knows the semantics of the document, and can often automatically detect which resources are required by the document.

Dynamic documents that the server can not analyze cannot be encapsulated into objects. Therefore, they are not replicated in our approach. In these cases, client requests are directed to the master site.

2.2 Cooperative Servers

One important issue for replicating Web documents is to gain access to computing resources in several locations worldwide (CPU, disk space, memory, bandwidth, etc.). On the other hand, adding extra resources locally is cheap and easy. Therefore, the idea is to trade cheap local resources for valuable remote ones. Servers automatically negotiate for resource peering. The result of such a negotiation is for a "hosting server" to agree to allocate a given amount of its local resources to host replicas from a "hosted server." The hosted server keeps control on the resources it has acquired: it controls which of its clients are redirected to the hosting server, which documents are replicated there and which replication policies are being used.

Of course, servers may play both "hosting server" and "hosted server" roles at the same time: a server may host replicas from another server, and replicate its own content to a third one. We use these terms only to distinguish roles within a given cooperation session.

Servers communicate with each other in an NNTP-like fashion. Each server is configured to communicate with a few other servers, which jointly communicate with more servers. This server network allows information to be spread efficiently among the whole group: messages are propagated from host to host through the entire network. Each message carries a unique identifier, so that duplicate message transmission can be avoided.

This communication channel allows servers to broadcast data such as their name, location and publicly available resources such as available storage capacity, network bandwidth, available Apache modules and database software.[1] When a server searches for a new hosting server in a certain area, it uses its local list of server locations to choose a site which would be suitable. It then contacts it directly and starts a negotiation phase. This negotiation can, for example, determine the price asked for accessing the required resources [3, 19].

[1] Note that this channel only propagates servers' meta-information. Actual content delivery is realized by means of separate document-specific channels.

Other types of data that flow through the network of servers are information about disconnected nodes (which may be caused by a failure or simply by a server deciding to leave the group). That way, hosted servers are informed when their hosting servers become unreachable, so they can stop redirecting clients to them.

2.3 Security Issues

The proposed system presents obvious security risks. A malicious server could accept to host Web document replicas, and deliver modified versions to the users. It could also accept replicas and refuse to answer any request directed to them, causing a "denial of service." More subtle attacks could be, for example, to allocate less resources to replicas than was negotiated.

We propose to solve this problem by using a peer-to-peer trust model. Each server associates a recommendation to a number of nodes, representing the trust it has that these sites are honest. Recommendations can be either positive, meaning that the recommended site is probably honest, or negative, meaning that the site is probably malicious. Recommendations are made public using the information propagation technique presented above. Based on the full set of recommendations, each node autonomously decides whether it trusts or distrusts each of the cooperative servers.

Although this scheme cannot provide complete certainty about a node being good or bad, it generally allows to discriminate good nodes from bad ones. It also tolerates a relatively high number of wrong recommendations (good recommendations for a bad node or *vice versa*). Discussing the full details of our trust model is beyond the scope of this paper, and can be found in [13].

Another issue is to protect servers against malicious dynamic documents, which could be used to run unauthorized code on remote platforms. This problem will be solved with classical techniques such as executing remote code in a sandbox or a remote playground [8].

3 System Architecture

Figure 1 shows Globule's architecture. It provides several distinct features: negotiating with remote servers for resource allocation and resource management, document replication and consistency, and automatic client redirection.

3.1 Delegated Resource Management

When a server notices that replicating the content of a specific document would improve the quality of service for that document's clients, it identifies one or more locations where replicas would be needed. Those locations are determined, for example, as the Autonomous Systems from which most of the requests originate.[2] It then locates suitable hosting servers in those locations and negotiates resource allocation, as was discussed in Section 2.2.

[2] Usually, a small number of origin Autonomous Systems account for a large fraction of the requests, as shown for example in [11].

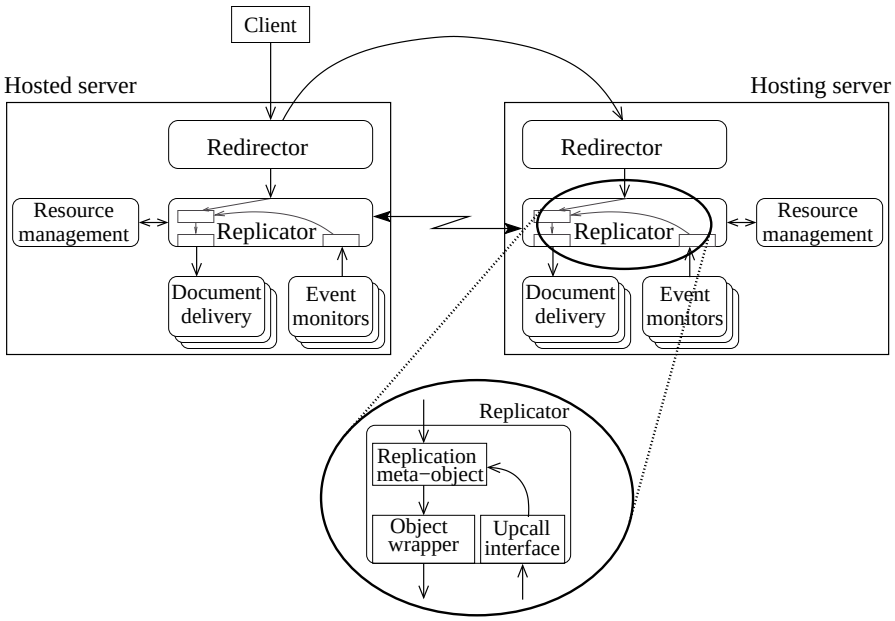

Fig. 1. General Architecture

When a hosted server has acquired appropriate resources on a hosting server, it can use those resources to create a replica and redirect clients to that replica. Its only obligation is to not exceed the amount of resources that it has been allocated. This rule is enforced by the hosting server: to prevent excessive use of storage space, a replacement module similar to those of caches is associated to the set of replicas from this hosted server. If a hosted server tries to use more resources than it has been allocated, the hosting server will automatically delete other replicas from the same server.

A hosting server, having allocated resources for several different hosted servers, manages each resource pool separately. Therefore, it must have several replacement module instances, each of which enforces resource limitations to one of the hosted servers. This mechanism is similar to those of partitioned caches [9].

Even though resource limitation is enforced by the hosting server, the hosted server remains in control of its allocated resources. It does so by attaching priority flags to its documents, indicating how important each replica is. When making replacement decisions, the hosting server takes these priorities into account in addition to standard parameters such as the frequency of requests and replica size. The more important a replica, the less likely it is to be removed.

Similar mechanisms can be setup to enforce limitations in resources such as bandwidth usage (for which Apache modules are readily available). Such limitations are of primary importance when considering multimedia documents, as their requests may easily prohibit processing requests for other documents at the hosting server.

Basically, one can associate bandwidth usage limitations to the replicas from a given hosted server. When the request stream for a specific set of documents from a hosted server exceeds the negotiated bandwidth, data transmission to the requesting clients is slowed down forcing those clients to share the allocated bandwidth. This algorithm is very similar to handling reserved bandwidth in routers.

This bandwidth management technique seems adequate to smooth occasional load peaks. However, slowing down connections is acceptable only as long as it remains exceptional. If a server must slow down connections on a regular basis, it will trigger a re-evaluation of the placement and replication policy, possibly leading to the creation of more replicas in its neighborhood (see Section 2.1).

3.2 Document Replication

As is also shown in Figure 1, a replica is made of two separate local objects: a document's content which is available in the form of delivery components capable of producing documents, and a replication meta-object which is responsible for enforcing the document's replication policy. Each object can either reside in memory or be marshaled to disk. State transitions between memory and disk are dictated by the per-hosted-server resource limitation module described in Section 3.1.

All replication meta-objects implement a standard interface, but they can have various implementations depending on the replication policy they represent. They maintain 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 to a document, the server transmits the characteristics of the request to the replication meta-object. Based on its implementation, the meta-object responds by indicating how to treat the request: reply immediately based on the local replica, or require to first check for freshness, etc. The server is in charge of actually performing the operation.

Once the replication meta-object has authorized the request, the Web server uses one of its standard document delivery modules to respond. These can be modules that deliver static documents, or modules that generate a document on request.

Certain replication policies require taking actions at other times than request time, such as prefetching large documents, periodically checking for a document's freshness, sending invalidations to replicas when the master copy is updated, and processing incoming invalidations. To do so, meta-objects can register to local services for being invoked when certain events take place. For example, a meta-object can request to be woken up periodically or when a given file is updated.

3.3 Client Redirection

Each document is assumed to have a home server. This server is responsible for automatically redirecting clients to their most suitable replica. Knowing the location of clients and replicas, such a selection can be reduced to a shortest path problem [14]. Each incoming client is associated with its Autonomous System. Knowing the locations of replica sites and the map of network connections between autonomous systems, each

client is directed to a server such that the number of autonomous systems on the path between the client and the replica server is minimized. If necessary, more criteria can be used in the selection of a replica site, such as the load of each replica site and the intra-AS network distances.

Mechanisms to effectively redirect clients to replicas can be classified in two categories [2]:

- HTTP redirection: when it receives an HTTP request, the server sends a redirection response, indicating from which URL the document should be retrieved. This scheme is very simple, but it is not transparent. That is, browsers display the URL of the mirror site instead of the home site. This may become a problem if, for example, a user bookmarks a mirror page. Later on, when he tries to access the page again, this mirror may have been removed.
- DNS redirection: before accessing a page, a browser needs to request the DNS to convert the server's name into an IP address. After locating an authoritative server for the given domain, the client's DNS server contacts it for actually resolving the name. DNS redirection requires the authoritative server to send customized responses depending on the location of the client [17]. Small TTL values are associated to responses, so that client DNS caches are updated often. A customized authoritative DNS server is necessary, but no other DNS server needs to be modified. This method is fully transparent to the user, since URLs do not need to be modified. On the other hand, it has a coarse granularity: it is not possible to replicate only part of a server, since all requests to this server will be sent to the mirrors.

We decided to use DNS redirection, as did most of the content distribution networks, such as Akamai. However, our system's architecture does not depend on this decision; we may later decide to use another mechanism, for example when the HTTP standard will feature more powerful redirection mechanisms.

As we discuss in next section, we will implement a customized DNS server *inside* the Globule Apache module. This allows for easier administration as well as for tight coupling between the replication and redirection modules.

4 Integration into the Apache Web Server

In order to allow for easy deployment, we decided to develop Globule as an Apache module. This way, turning a normal Apache server into a replication-aware server would require only compiling a new module into the server and editing a configuration file.

The Apache Web server is built from a modular design, which enables one to easily add new features [7]. It decomposes the treatment for each request into several steps, such as access checking, actually sending a response back to the client, and logging the request.

Modules can register handler functions to participate in one or more of these steps. When a request is received, the server runs the registered handlers for each step. Modules can then accept or refuse to process the operation; the server tries all the handlers registered for each step until one accepts to process it.

Many third party modules have been developed to extend Apache in a number of ways, such as the PHP server scripting language and a streaming MP3 delivery module.

The architecture of Apache provides us all the tools necessary to implement Globule: a replication module can, for example, intercept a request before being served by the standard document delivery modules to let the replication meta-objects check for consistency. Likewise, servers can communicate with each other by HTTP.

Although Apache has been originally designed to handle only the HTTP protocol, its newest version allows one to write modules that implement other protocols. An appropriate module could therefore turn Apache into a DNS server. We plan to use this feature for redirecting clients to mirrors.

5 Related Work

5.1 Content Distribution Networks

Many systems have been developed to cache or replicate Web documents. The first server-controlled systems have been push-caches, where the server was responsible of pushing cached copies close to the users [5]. More recently, content distribution networks (CDNs) have been developed along the same idea. These systems rely on a large set of servers deployed around the world. These servers are normal caches, configured as surrogate proxies. The intelligence of the system is mainly concentrated in the DNS servers which are used to direct clients to a server close to them. Consistency is realized by incorporating a hash value of document's content *inside* its URL. When a replicated document is modified, its URL is modified as well. This scheme necessitates to change hyperlink references to modified documents as well. In order to deliver only up-to-date documents to users, this system cannot use the same mechanism to replicate HTML documents; only embedded objects such as images and videos are replicated.

Globule presents three major differences with CDNs. First, since its consistency management is independent from the document naming scheme, it can replicate all types of objects. Second, contrary to CDNs which use the same consistency policy for all documents, Globule selects consistency policies on a per-document basis so that each document uses the policy that suits it best. This is of particular interest when replicating heterogeneous sets of documents such as static and dynamic pages, and streaming documents. Finally, the system does not require one single organization to deploy a large number of machines across the Internet: Globule users automatically trade resources with each other, therefore incrementally building a worldwide network of servers at low cost.

5.2 Peer-to-peer Systems

Globule can be seen as a peer-to-peer system [10]. Most of these systems, such as Gnutella, are used to locate files at user's locations without the need for a centralized server. Other peer-to-peer applications allow, for example, for distribution of parallel computations on users' machines, such as the SETI@home project.

Globule qualifies as a peer-to-peer system because it makes nodes from different administrative domains cooperate to provide a service that no single server could ever

achieve. Moreover, its architecture is entirely decentralized, which means that no central server is necessary. The protocol for propagating meta-information among the network of servers is also similar to the protocol for searching a file among a Gnutella network.

The main difference between Globule and other peer-to-peer systems relies in the nature of the inter-node cooperation. Resources being exchanged in Gnutella are files whereas Globule servers exchange storage space and network bandwidth.

5.3 Streaming Media Replication

Replicating streaming documents is a very different issue than replicating regular Web documents. Although we do not consider this class of documents as our primary application target, we believe that our approach could be beneficial in this area as well.

On the one hand, consistency is often not a major issue, since these documents are hardly ever updated. On the other hand, since they are quite big (typically between 1 MB and 100 MB), partial replication often offers better cost-benefit ratio than total replication. Prefix caching (i.e., replicating only the beginning of each file) can help reducing the startup delay and smooth the subsequent bandwidth requirement between the server and the cache [4, 16]. The utility of such prefix caching is reinforced by the fact that many users stop movie playback after only a few seconds [1]. Another possibility for partial replication is video staging, where only the parts of the variable-bit-rate video stream that exceed a certain cut-off bandwidth are replicated [20]. Replication of layered encoded video can also be realized by replicating only the first N layers [6]. However, the authors note that this last technique is of interest only if at least some clients use the ability of viewing a low-quality version of the video.

Redirecting clients to appropriate replicas is more difficult with streaming documents than with Web documents. Streaming sessions last longer and use more resources at the replica site, in particular in terms of network bandwidth and disk I/O. This leads to complex scheduling and resource reservation problems at the server or at the replicas. However, many solutions have been proposed that range from clever scheduling policies to mechanisms for redirecting a client to another server during a streaming session [15].

Clearly, streaming documents place different requirements on wide-area content delivery platforms than Web documents. Specific mechanisms are being developed to handle them efficiently. Moreover, these mechanisms do not apply to all situations, depending on the interactive or non-interactive nature of the documents, their compression formats, and the usage pattern of users.

We believe that our adaptive per-document replication scheme would be well suited to handle the variable requirements of video-on-demand applications. Future research will investigate the interest of our approach for wide-area delivery of streaming documents.

6 Conclusion

We have presented Globule, a platform for Web document replication. Globule integrates all necessary services into a single tool: dynamic creation and removal of repli-

cas, consistency management, and automatic client redirection. Globule will be implemented as a module for the Apache server.

The architecture presented in this article is still work in progress, but we hope to release a first prototype soon. Two problems have been left for future work. First, security: the administrator of a server would like to make sure that remote servers which accepted to host his replicas will do it without modifying documents, for example. We plan to use a trust model to solve this. Second, details of server-to-server negotiation still have to be figured out.

When the Globule project is completed, we expect to provide a free cooperative platform for Web document replication that will match the ever-increasing quality of service expectations that users have.

References

1. Acharya, S., Smith, B.: MiddleMan: A video caching proxy server. In: Proc. 10th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV), Chapel Hill, NC (2000)
2. Barbir, A., Cain, B., Douglis, F., Green, M., Hofmann, M., Nair, R., Potter, D., Spatscheck, O.: Known CDN request-routing mechanisms. Internet Draft (2001)
3. Buyya, R., Abramson, D., Giddy, J.: An economy driven resource management architecture for global computational power grids. In: Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, NA (2000)
4. Gruber, S., Rexford, J., Basso, A.: Protocol considerations for a prefix-caching proxy for multimedia streams. In: Proc. 9th International World Wide Web Conference, Amsterdam (2000)
5. Gwertzman, J., Seltzer, M.: The case for geographical push-caching. In: Proc. 5th Workshop on Hot Topics in Operating Systems (HotOS), Orcas Island, WA, IEEE (1996)
6. Kangasharju, J., Hartanto, F., Reisslein, M., Ross, K.W.: Distributing layered encoded video through caches. In: Proc. 20th INFOCOM Conference, Anchorage (AK), IEEE (2001)
7. Laurie, B., Laurie, P.: Apache: The Definitive Guide. 2nd edn. O'Reilly & Associates, Sebastopol, CA. (1999)
8. Malkhi, D., Reiter, M.: Secure Execution of Java Applets using a Remote Playground. IEEE Transactions on Software Engineering **26** (2000) 1197–1209
9. Murta, C.D., Almeida, V., Meira, Jr, W.: Analyzing performance of partitioned caches for the WWW. In: Proc. 3rd Web Caching Workshop, San Diego, CA (1998)
10. Oram, A., ed.: Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates, Sebastopol, CA. (2001)
11. Pierre, G., Kuz, I., van Steen, M., Tanenbaum, A.S.: Differentiated strategies for replicating Web documents. Computer Communications **24** (2001) 232–240
12. Pierre, G., van Steen, M., Tanenbaum, A.S.: Self-replicating Web documents. Technical Report IR-486, Vrije Universiteit, Amsterdam (2001)
13. Pierre, G., van Steen, M.: A trust model for cooperative content distribution networks. Technical report, Vrije Universiteit, Amsterdam (2001) In preparation.
14. Qiu, L., Padmanabhan, V., Voelker, G.: On the placement of Web server replicas. In: Proc. 20th INFOCOM Conference, Anchorage (AK), IEEE (2001)
15. Schulzrinne, H., Rao, A., Lanphier, R.: Real time streaming protocol (RTSP). RFC2326 (1998)
16. Sen, S., Rexford, J., Towsley, D.: Proxy prefix caching for multimedia streams. In: Proc. 19th INFOCOM Conference, New York, NY, IEEE (1999)

17. Tang, W., Du, F., Mutka, M.W., Ni, L.M., Esfahanian, A.H.: Supporting global replicated services by a routing-metric-aware DNS. In: Proc. 2nd International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, San Jose, CA (2000) 67–74
18. van Steen, M., Homburg, P., Tanenbaum, A.S.: Globe: A wide-area distributed system. IEEE Concurrency **7** (1999) 70–78
19. Wolski, R., Plank, J.S., Brevik, J., Bryan, T.: Analyzing market-based resource allocation strategies for the computational grid. Technical Report CS-00-453, University of Tennessee, Knoxville (2000)
20. Wang, Y., Zhang, Z.L., Du, D.H., Su, D.: A network-conscious approach to end-to-end video delivery over wide area networks using proxy servers. In: Proc. 18th INFOCOM Conference, San Francisco, CA, IEEE (1998) 660–667