Globule: A Collaborative Content Delivery Network

Guillaume Pierre   Maarten van Steen

Abstract

We present Globule, a collaborative content delivery network developed by our research group. Globule is composed of Web servers that cooperate across a wide-area network to provide performance and availability guarantees to the sites they host. We discuss the issues involved in developing and setting up a large-scale collaborative CDN and provide solutions for many of its unique problems.

Introduction

Internet users become increasingly aware of the potentials that it has to offer. In particular, we can observe a trend in which end users move from being only content browsers to also being content providers. This trend is exemplified by peer-to-peer file sharing systems, the many Web logs, and the many personal Web sites. Furthermore, the development of broadband Internet connections allows an increasing number of users to maintain their own permanent Internet node, running various servers such as those for E-mail, Web content, file transfer, remote login, etc.

In this paper, we concentrate on individuals and organizations who wish to maintain a Web site. A common approach is to place the site under the regime of a commercial Web hosting service. In this case, the user merely provides Web documents, which are then uploaded to the service provider. These services are generally available at a moderate monthly charge.

Although this approach generally works fine, matters become complicated when the request load increases and quality-of-service demands rise. For example, an organization may want to have guarantees concerning the availability and reliability of its site. Likewise, it may require that the client-perceived latency be minimized and that enough storage space and bandwidth be available. In many cases, the resources necessary to meet these QoS demands fluctuate: high bandwidth is not always needed, but only when many clients are simultaneously accessing the Web site. Similarly, the resources necessary for high availability may vary depending on the status of the other servers in the system.

In all these cases, it is important to correctly balance the dimensioning of resource usage and the required performance. A common solution is to make use of the services of a Content Delivery Network (CDN) [4]. In effect, a CDN provides many resources that can be dynamically allocated to the sites it is hosting. CDNs are commonly operated on a commercial basis. However, it is not obvious why Web hosting for increased QoS should be outsourced, as in many cases the necessary resources are already available elsewhere in the system. Certain users or organizations, such as a retail chain, may have multiple computers online, so the extra resources needed are generally already available in-house. Alternatively, several users can decide to build their own CDN and join their respective resources in order to average everyone's resource needs over all the provided servers.

We thus come to a simple model in which providing some local resources to others may allow the provisioner to increase its resource usage elsewhere, in turn improving its overall QoS. Another way of looking at this, is that donating local resources to a community of nodes allows each member to ensure itself of sufficient resources elsewhere when needed. The difficulty, however, is to effectively *share* those resources.

Especially for demanding end users, and in general groups and organizations not willing or capable to make use of commercial CDNs we present an alternative, namely to join a **collaborative CDN** (CCDN). A collaborative CDN is an overlay network composed of end-user machines that operate in a peer-to-peer fashion across a wide-area network. Members offer resources in the form of storage capacity, bandwidth, and processing power. In return, their Web content is transparently and automatically replicated according to QoS demands regarding performance, availability, and reliability. In this way, a collaborative CDN offers the same service as a commercial CDN, but at virtually no extra costs beyond what a member is already paying for Internet connectivity.

In this paper, we discuss the issues involved in developing and setting up a large-scale collaborative CDN and provide solutions for many of its unique problems. Our discussion is based on our experience with Globule, a CCDN that has been developed by our group. Globule has been fully operational since approximately December 2003. It has been developed for UNIX-based systems, but has also been ported to Windows machines. We use Globule as a platform to identify and solve problems related to deploying, managing, and operating fully decentralized collaborative end-user systems.

Issues

We assume that a large collection of interconnected nodes are willing to interact. As nodes are voluntarily provided by end-users, we assume that they are very diverse with respect to the resources they can or are willing to provide, their operating systems, and their degree of availability. For example, we expect that many nodes are connected through (asymmetric) DSL or cable modem lines. Likewise, end-user machines cannot be expected to be highly available. Various peer-to-peer studies indicate that many users have their machines available for at most a few hours. At best, we should expect machines from organizations to improve on this by an order of magnitude. Moreover, we do not wish to exclude special computers, such as laptops, from participating in a CDN. The resulting *high churn*, as it is called, is a problem unique to overlay networks composed of end-user machines and deeply affects their design.

The goal of a CDN is to replicate Web content to increase performance and availability [7]. The need for replication immediately raises the following issues, which are naturally also relevant to collaborative CDNs:

1. A CDN should appear to its clients as a single powerful and reliable server. In particular, this excludes imposing clients to use any special component besides a standard Web browser, such as plugins or daemons.

2. Each client should be automatically *redirected* to the replica server that suits it best. How this is to be done is another issue that requires attention.

3. If documents are replicated for performance, they should preferably be placed close to their clients. As a consequence, we need a notion of *network proximity*.

4. Mutable replicated documents require *consistency management*, i.e., we need to decide when and how replicas are to be kept consistent.

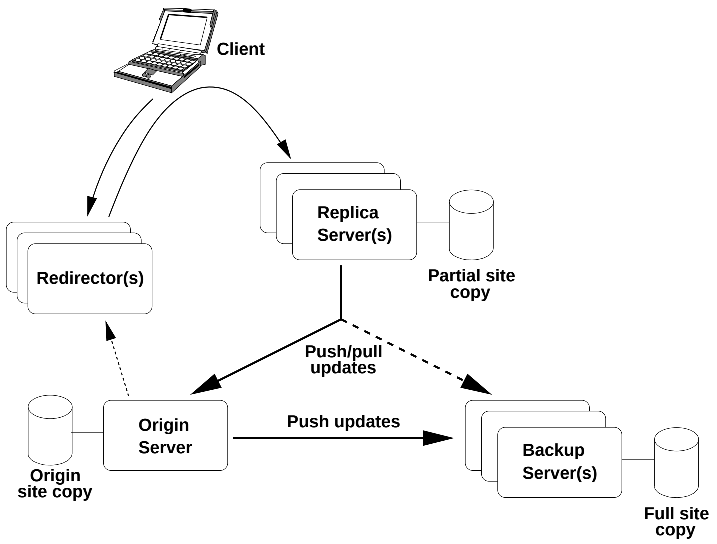


Figure 1: Globule Model

In addition to these requirements, the fact that we are dealing with a collaborative system built from end-user machines implies that we cannot expect specific expertise to be available for managing the collaboration. In particular, we need to address the following issues as well:

5. A server needs to have some knowledge concerning the organization of the CDN so that it can identify the places where to replicate its documents. In particular, we may need a *brokerage* system to permit automated resource discovery and allocation.

6. Nodes will need to collaborate, implying that resource usage and provisioning should be *fair* among the nodes.

7. The procedure required to join the collaborative CDN (including installing the necessary software and configuring it correctly) should be simple and require no specific technical knowledge. For example, a simple Web-based registration system could be used to allow users to register their machine and resources, make changes, and also to end membership.

8. Security should be enforced amongst members so that malicious users cannot attack the system.

Of course, most, if not all aspects that are related to the distribution and replication of Web content across the CDN should be preferably transparent to a member. Being a member should be (nearly) as simple as having a Web site hosted on a single machine.

Model

Globule makes a strong distinction between a *site* and a *server*. A site is defined as a collection of documents that belong to one specific user (the site's owner). A server is a process running on a machine connected to a network, which executes an instance of the Globule software. Each server may host one or more sites, that is, be capable of delivering the site's content to clients.

To be effective, a collaborative CDN must realize three main tasks. (i) It must be able to distribute the contents of a hosted site among multiple servers and maintain its consistency in the presence of updates; (ii) it must redirect client requests to the best available server; and (iii) it must efficiently deliver the site's contents to the clients. These three tasks need to be operating even when some servers are down. These constraints lead us to the following system model, as illustrated in Figure 1.

Each site is hosted by a modest number of servers belonging to the collaborative CDN (typically in the order of a dozen servers). One of them is called the *origin* server. It contains the authoritative version of all documents of the site, and is responsible for distributing contents among other involved servers. The origin server typically belongs to the site's owner.

The origin server of a given site should normally be reachable by other servers at all times. However, as this server cannot be assumed to be always available, it is helped out by any number of *backup* servers. These servers maintain a full up-to-date copy of the hosted site. The goal of the backup servers is to guarantee the availability of the site. When the origin is unavailable, it is sufficient that one of the backups be available for the site to work correctly.

In addition to backup servers, a site can be hosted by any number of *replica* servers. Unlike backups, the goal of replicas is to maximize performance. Depending on its request load and quality of service requirements, a site may have any number of replica servers, preferably located across the network so that there is a replica close to each potential client. A replica server for a site is typically operated by a different user than its origin, so the replica's administrator may impose restrictions on the amount of resources (disk space, bandwidth, etc.) that the hosted site can use on this machine. As a result, each replica server typically contains only a partial copy of its hosted site. Similar to a caching proxy, when requested for a document not present locally, a replica server fetches the document from its origin before delivering it to the client.

Finally, a site must have one or more *redirector* servers, whose task is to redirect client requests to the replica server that can serve them best. In Globule, redirectors can use either HTTP-based or DNS-based redirection. Redirectors monitor the availability of the origin, backup and replica servers so that they always redirect client requests to an available server. Similar to backup servers, the site will be functioning correctly as long as one of the redirectors is available.

It should be clear that the distinction between origin, replica, backup and redirector servers refers only to the role that a given server takes with respect to any given site. The same server may for example simultaneously act as the origin and one of the redirectors for its owner's site, as a backup for a few selected friend's sites, as a replica for other sites, and as a redirector for yet other sites.

Content Distribution

When replicating documents for performance, a CDN should strive to place replicas close to where clients are. Such a placement generally leads to low access times. Proximity between Internet nodes can be measured according to different metrics such as the number of hops in the shortest route and round-trip delays. In Globule, we take internode latency as our proximity measure, and use this metric to optimally place replicas close to clients, and also to redirect clients to an appropriate replica server.

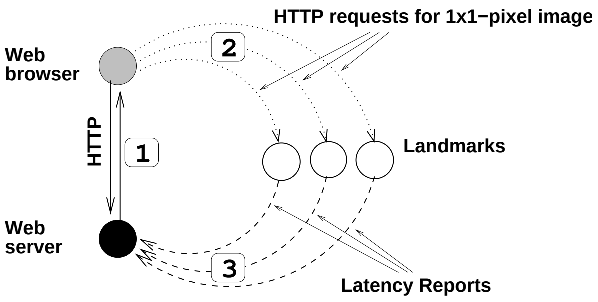


Figure 2: Positioning a new node in Globule.

Computing proximity

Estimating internode latencies in a large overlay network is a formidable task. In principle, to compute optimal replica placement we would need to measure the latency from each potential client machine to each potential replica server. However, the large numbers of machines involved clearly make these measurements impossible. To solve this problem, latencies in Globule are estimated by positioning nodes in an M-dimensional geometric space, similar to what is done by GNP [3] and Vivaldi [1]. The latency between any pair of nodes is then estimated as the Euclidean distance between their corresponding M-dimensional coordinates. A typical value for M is 6. The coordinates of node X are calculated based on the measured latencies between X and m designated "landmark" nodes, where m is slightly larger than M. Consequently, estimating pairwise latencies between N nodes requires much fewer measurements ($N \cdot m$) than in the naive approach ($N(N-1)/2$). Additional techniques such as node clustering allow reducing the number of necessary latency measurements even further.

Latency measurements can be made totally transparent to the clients, which complies with our requirement that no special software should be installed at the client machines. As shown in Figure 2, when a Web browser accesses a page (step 1) it is requested to download a 1x1-pixel image from each of the landmarks (step 2). The client-to-landmark latency is measured passively by the landmark during the TCP connection phase, and reported back to the origin server where the node's coordinates are computed (step 3).

Latency estimations derived from this method are reasonably good: with a space dimension of 6, 90% of the latency estimations fall in the interval $[\frac{2}{3}L, \frac{3}{2}L]$, where L is the actual latency between two nodes. This accuracy is enough to handle the placement of replicas and to redirect clients to their best replica server.

Replica placement

When clients of a specific site have been localized, we need to identify areas in the M-dimensional space where many clients are located, and place replicas there. A very simple algorithm consists of partitioning the space into cells of identical size, and ranking the cells according to the number of nodes each of them contains. By placing a replica in each of the k highest ranked cells, we thus minimize the overall client-perceived latency. We have proved that, provided that cells overlap and the cell size is chosen carefully, this algorithm provides placements almost as good as the best algorithm known to date. On the other hand, the computational complexity of our algorithm is much lower. This is crucial for large systems, where near-optimal replica placement can now be computed within minutes instead of days [8].

Client redirection

In a system where replicas of a given site may be created or deleted dynamically, and where few assumptions are made on their availability, we cannot expect clients to manually select the closest server where requests should be issued. Instead, we need to provide automatic client redirection.

A redirector is a component that monitors the availability of replica servers of a given site and instructs client browsers on where that site should be accessed. We support HTTP as well as DNS redirection. With HTTP redirection, a redirector can decide on a per-page basis which replica server is to handle the request. To this end, the redirector returns the URL of the replicated page to the client. The drawback of HTTP redirection is the loss of transparency and control: as the client is effectively returned a modified URL, it can decide to cache that URL for future reference. As a consequence, removing or replacing a replica may render various cached URLs invalid.

As an alternative, we also support DNS redirection. In this case, redirection is based entirely on a site's host name and the client's location. When the client resolves the site's hostname, the redirector returns the IP address of the replica server closest to the client. In this case, redirection is done on a per-site basis as the DNS redirector has no means to differentiate between individual pages. On the other hand, DNS redirection is mostly transparent to the client, allowing for better control of replica placement.

A redirector should also implement a redirection policy, which is an algorithm to dictate where each client should be redirected. The default policy redirects each client to the closest replica in terms of estimated latency. However, other factors can be introduced to influence the choice, such as the respective load of each replica server.

Content Availability

As we already mentioned, a CCDN will typically run on a collection of end-user machines whose availability cannot be relied on. Moreover, when the number of servers taking part in hosting a given site increases, the probability that at least one server is unreachable grows quickly. We therefore need to make sure that a site will remain functional even if a fraction of its hosting servers fail.

The first issue to address is the availability of a redirector at the time of a client request. When using DNS redirection, this issue is easily solved. A DNS redirector is simply a DNS server responsible for the site's host name, and which returns responses customized to each client. The DNS protocol allows multiple redundant servers to be registered for the same name; if one server fails, then the other ones are automatically queried instead.

The second issue is to make sure that redirectors always direct clients to a working replica server. To this end, redirectors periodically probe the availability of the replica servers. Whenever one replica server becomes unreachable, redirectors will stop redirecting clients to it, and direct them to the second best server instead.

The last issue related to content availability is to ensure that any replica server is able to obtain fresh up-to-date copies of the requested documents if it does not have them already. As previously mentioned, an origin server should always maintain at least one backup server, which by definition always has a full and up-to-date copy of the whole site. If a replica server cannot obtain a copy of a document from the origin, it automatically fetches it from one of the backups.

Management

A collaborative CDN relies on multiple users to cooperate for hosting each other's sites. This raises two specific issues. First, we must secure the system against malicious users. Second, we must provide the means by which users locate suitable available servers where to place their replicas.

Security

In a CCDN, servers will often host content that does not belong to their own administrator. In such situations, most administrators would demand guarantees that potentially malicious content cannot damage the server, by means of excessive resource usage, access to confidential information, or any other type of misbehavior. This threat is particularly present when hosting dynamic content, where arbitrary code can be executed to generate documents. This is a well-known problem, however, which is usually addressed by means of sandboxing techniques.

A more difficult security-related issue is that a content owner expects guarantees that replica servers will actually perform their assigned task faithfully. Indeed, a malicious replica server could, for example, reject incoming connections (creating a denial-of-service attack) or deliver modified versions of the original content. Our current solution consists of instrumenting certain clients so that they inform the origin server of the way their requests to the replicas were handled. The origin server can thus detect unexpected behavior from the replicas, and cease cooperation with them if they are found unreliable. This technique, however, requires a fraction of Web clients to be instrumented with Globule-related code, which contradicts our goal of keeping clients unmodified. For this reason, we are currently exploring approaches which would allow origin servers to obtain cryptographic proofs of the behavior of its replicas, without involving clients at all.

Brokerage

The effectiveness of a CCDN depends to a large extent on the number of users who take part in it. For example, a relatively large server base allows to better absorb significant variations in request rates addressed to individual sites; it also increases the probability that replica servers are available near the optimal locations computed by the replica placement algorithm. We therefore need to offer a service by which users can locate each other's servers.

When users install Globule, they are directed to a Web site known as the "Globule broker service" where they can register their new server(s). Users can also specify policies which define who is authorized to use a given server for replica, backup or redirection purposes. Finally, they are proposed a number of servers willing to host replicas of their content. Future Globule versions will also allow origin servers to automatically query the broker for potential servers located close to a given location. As any other Web site, a broker is likely to be replicated, allowing brokerage to scale.

Configuration

Establishing replication between an origin and a replica server requires that the two servers authenticate each other. They must therefore agree on a shared password and update both configurations before the replication can take place. To facilitate this procedure, the broker service generates the appropriate configuration files automatically. Globule servers can periodically fetch a fresh copy of their own configuration file from the broker, which fully automates the configuration update process.

```
# Origin server's configuration file
<VirtualHost *>
  ServerName   131-38-194-67.mysite.globeworld.net
  ServerAlias  mysite.globeworld.net
  DocumentRoot "/var/www/mysite.globeworld.net/htdocs"
  <Location "/">
    GlobuleReplicate on
    GlobuleReplicaIs    http://131-38-32-118.mysite.globeworld.net/  RWkVh6l31hHoi
    GlobuleRedirectorIs http://131-38-199-246.mysite.globeworld.net/ vccXwYA5V8Eas
  </Location>
</VirtualHost>
```

```
# Replica server's configuration file
<VirtualHost *>
  ServerName   131-38-32-118.mysite.globeworld.net
  ServerAlias  mysite.globeworld.net
  DocumentRoot "/var/www/mysite.globeworld.net/htdocs"
  <Location "/">
    GlobuleReplicaFor http://131-38-194-67.mysite.globeworld.net/ RWkVh6l31hHoi
  </Location>
</VirtualHost>
```

Figure 3: Configuration files of an origin server and its corresponding replica

Figure 3 displays example configuration files generated by the broker[1]. They show the configuration of a replicated site called `mysite.globeworld.net`. This site is hosted by an origin server (`131.38.194.67`, also known as `131-38-194-67.mysite.globeworld.net`) and a replica server (`131.38.32.118`, also known as `131-38-32-118.mysite.globeworld.net`). Since we are using DNS redirection, both servers are also registered under the same name `mysite.globeworld.net`. Finally, the origin server has an entry to define a DNS redirector running at `131-38-199-246.mysite.globeworld.net` (whose configuration is not shown for lack of space).

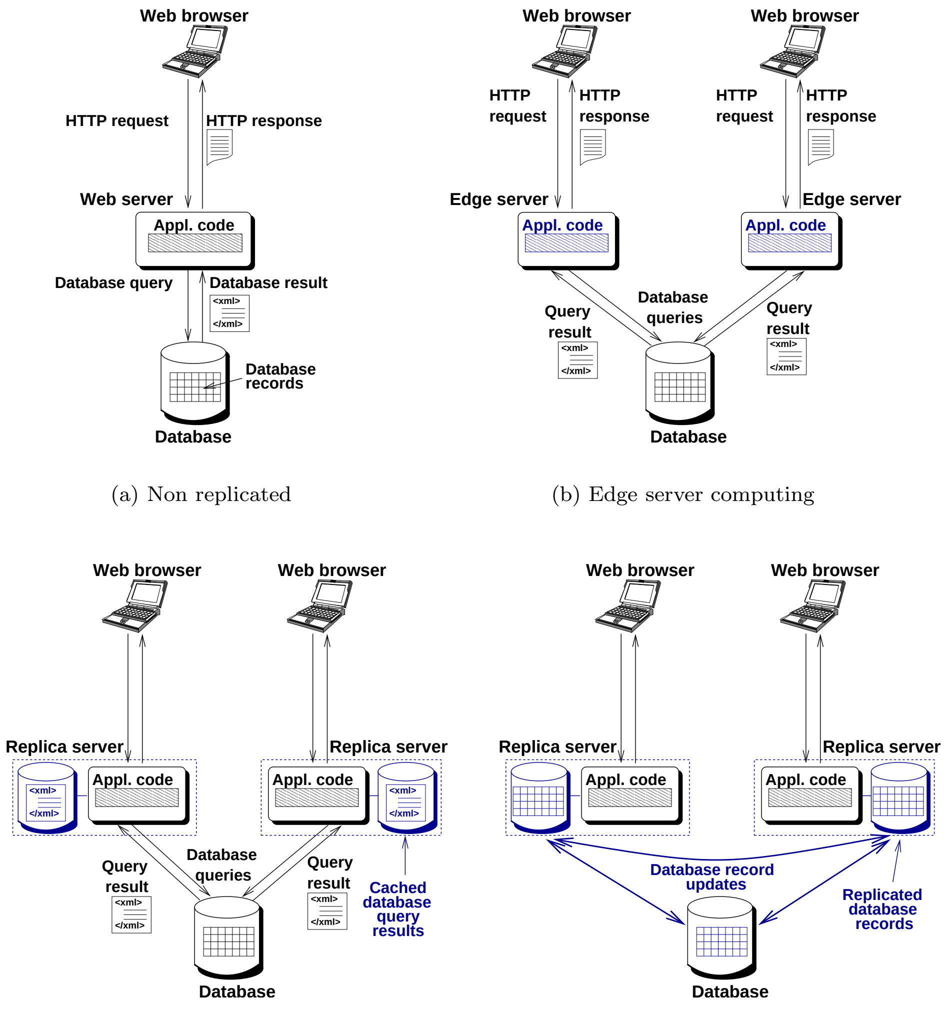
Supporting Web applications

The discussion so far applies to hosting both static and dynamically generated content. In practice, however, both document types must be handled differently. Replicating static content, even in the presence of updates, is relatively simple. On the other hand, a large amount of Web content is generated dynamically. Web pages are being generated upon request using applications that take, for example, individual user profile and request parameters into account when producing the content. These applications often run on top of databases. When a request arrives, the application examines the requests, issues the necessary read or update queries to the database, retrieves the data, and composes the page that is sent back to the client (see Figure 4(a)). For example, an e-commerce application can provide links to "best-sellers lists," which effectively require a search through the customer order database.

A simple approach to handling dynamic documents in content delivery networks is edge-server computing, which distributes the application code at all replica servers and keeps the database centralized (Figure 4(b)). Dynamic documents can then be generated at the edge servers. However, each database query must be issued across a wide-area network, thereby experiencing significant latency. Also, keeping the database centralized creates a performance bottleneck.

To overcome these limitations, we designed two complementary approaches to efficiently improve the scalability of dynamically generated documents. Even though

[1]Globule is implemented as a third-party module for the Apache Web server, which explains the recognizable Apache-like syntax of configuration files.



(a) Non replicated

(b) Edge server computing

(c) Database query caching

(d) Database record replication

Figure 4: Various Web application hosting techniques

the HTTP requests addressed to Web applications are often very different, we noticed that the database queries addressed from the application code to the database are often identical to each other. For applications with so-called high database query locality, we cache the results of database queries at the replica servers (Figure 4(c)). The application code can then be executed at the replica based on locally cached database query results, which provides significant performance improvements to the system.

When the database query locality is low, database query caching will not provide any improvement. In this case, it is more efficient to *replicate* the underlying database at each replica server, so that database queries can be issued to the local database (Figure 4(d)). The difficulty in this case is that each update to the data may lead to excessive communication to keep replicas consistent. This issue can be addressed by replicating each element of the database to only a fraction of all servers [6]. This allows significantly reducing the amount of communication required to maintain consistency between the databases, while maintaining high performance for generating documents.

Related work

Collaborative content delivery has been the subject of many research and development efforts. In particular, peer-to-peer overlays such as Kazaa and BitTorrent have recently received a lot of attention. These systems are best suited for large-scale distribution of large, static and immutable files such as music or video files. Web content, on the other hand, is much harder to deliver using peer-to-peer overlays because its content is updated frequently and is often generated dynamically. To our knowledge, no peer-to-peer overlay offers such features so far.

Systems such as Coral [2] and CoDeeN [9] address these issues by building large-scale collaborative Web caches on top of a peer-to-peer overlay that regular Web browsers can access. This architecture allows handling large amounts of traffic with reasonable performance. However, it also makes it very hard to select replica placement and to handle dynamically-generated content.

Two systems allow for collaborative hosting of dynamic Web content, but they make strong assumptions on the characteristics of the applications. ACDN assumes that the applications do not modify their underlying data [5]; DotSlash assumes that the database is not the performance bottleneck of Web applications, and uses edge-server computing techniques [10].

Conclusions

We believe that our work on Globule contributes to providing end users with state-of-the-art Web hosting functionalities. Globule has been in use in our group since December 2003; more recently, it has been used to host `www.minix3.org`, which served about 600,000 Web pages and 12,000 CD-ROM images to 50,000 clients during the first 24 hours following the release of the Minix3 operating system. Globule software is available for public use under an open-source licence at `www.globule.org`.

We consider Globule as a vehicle for research: building it leads us to address a wide range of difficult research topics, such as client localization, replica placement and Web application replication. Future research topics derived from our experience with Globule include the enforcement of fair resource usage among participants, the detection of and reaction to flash crowds, and the design of systems that aggregate many low-end machines into the abstraction of a few high-end reliable servers.

Naturally, the results of our research are initially designed with respect to specific issues in Globule. However, most of it also applies to other types of large-scale distributed systems, such as peer-to-peer networks and computing grids.

References

[1] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. In *ACM SIGCOMM Conference*, Portland, OR, August 2004.

[2] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with Coral. In *1st Symposium on Networked Systems Design and Implementation*, pages 239–252, San Francisco, CA, March 2004.

[3] T. S. Eugene Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *IEEE INFOCOM Conference*, New York, NY, June 2002.

[4] Michael Rabinovich and Oliver Spatscheck. *Web Caching and Replication*. Addison Wesley, Reading, MA, USA, 2002.

[5] Michael Rabinovich, Zhen Xiao, and Amit Aggarwal. Computing on the edge: A platform for replicating internet applications. In *8th International Web Caching Workshop*, Hawthorne, NY, September 2003.

[6] Swaminathan Sivasubramanian, Gustavo Alonso, Guillaume Pierre, and Maarten van Steen. GlobeDB: Autonomic data replication for web applications. In *14th International World-Wide Web Conference*, Chiba, Japan, May 2005.

[7] Swaminathan Sivasubramanian, Michał Szymaniak, Guillaume Pierre, and Maarten van Steen. Replication for web hosting systems. *ACM Computing Surveys*, 36(3):291–334, 2004.

[8] Michał Szymaniak, Guillaume Pierre, and Maarten van Steen. Latency-driven replica placement. In *IEEE Symposium on Applications and the Internet*, Trento, Italy, January 2005.

[9] Limin Wang, KyoungSoo Park, Ruoming Pang, Vivek S. Pai, and Larry Peterson. Reliability and security in the CoDeeN content distribution network. In *Usenix Annual Technical Conference*, Boston, MA, June 2004.

[10] Weibin Zhao and Henning Schulzrinne. DotSlash: Handling web hotspots at dynamic content web sites. In *IEEE Global Internet Symposium*, Miami, FL, March 2005.