Adding Advanced Caching and Replication Techniques to the Apache Web Server

Joachim Marder, Steffen Rothkugel, Peter Sturm
University of Trier
D-54286 Trier, Germany
Email: marder@jam-software.com, sroth@uni-trier.de, sturm@uni-trier.de

Abstract

In this project replication based on invalidation messages has been added to the Apache web server by developing three new Apache modules: mod_replication, mod_cgr and mod_wlis. For each document, the module mod_replication keeps track about existing replication servers. It checks autonomously for changes in documents and reports them to the replication servers by sending invalidation messages coded into a HTTP request. Additionally it adds the information about replication servers to the HTTP header of each delivered document. The module mod_cgr can be installed on a proxy server which should be upgraded to a replication server. It controls the selection of the replicated documents on the base of statistical data derived from the proxy log. The module registers itself as registration server the originating server of each replicated document and controls the updating of invalidated documents. Currently three different replication types are available, which result in an immediate or delayed update of an invalidated document. Last but not least the module mod_wlis extracts the information about replication servers from the HTTP header of each document, which passes the proxy server, and collects them in a database. This information can be used to address an alternative server if the desired server cannot be reached or is too slow. In this paper, the module implementation and the limitations of the Apache API for modules is discussed in detail.

Keywords: Proxy, Replication Server, Apache, Modules, Apache Module API, Document Caching, Document Replication, Invalidation, Name Service

Table of Contents

1. Introduction
2. The Apache Web Server and its API
   2.1 Introduction to the Apache server
   2.2 The Apache API
   2.3 Limitations of the API
3. The new Apache modules
   3.1 Overview
   3.2 The module mod_replication
   3.3 Changes to the proxy module
   3.4 The module mod_cgr
   3.5 The module mod_wlis
4. Further Work
5. Conclusions
References

1. Introduction

Caching is one of the primary measures to utilize the available bandwidth of the Internet more efficiently. The basic idea is to improve access time by storing copies of frequently used data as close to the consumer as possible. Performance gains can only be obtained up from the second request for the same data. A prerequisite for a successful deployment of document caching is a sufficiently high reference locality. Several evaluations of proxy log files like in [1] have shown, that this reference locality exists for web documents too: only a very small subset of documents incorporates a high number of references while most documents are accessed seldom or never.

Today all web browsers have a built-in cache. A proxy server, which acts as a mediator between the user's machines and the the servers outside the local network, has an ideal position to include another cache. All documents, which are requested by the clients, must pass the proxy server and can then be placed in a cache. Because this cache is shared among several users, the probability of a document to be accessed more than once increases. Another approach is the mirroring of complete servers. In this case, popular sites are mirrored on different servers all over the world. Periodically (usually at night) the mirror servers are synchronized with the original server. This technique has been used long before the existence of the World Wide Web (WWW) for FTP servers.

Caching and mirroring suffer from the problem of document staleness. Cached documents may change on the originating server but caching proxies and mirror servers are not aware of these changes. Users, accessing a cached or mirrored copy of a document, can never be sure if it is the most recent version. In order to minimize document staleness in the cache of proxy servers, several approaches are used today:

- In the header of each document transferred by the Hypertext Transfer Protocol (HTTP, [3]) an expiration date for the document can be included. Unfortunately this
Implementing Advanced Caching and Replication Techniques in the Apache Web Server

possibility is used by only very few web servers today, although this approach is very suitable for periodically changing sites like online newspapers. One reason for this is, that many content providers are interested in the total number of document downloads and don't want their documents to be cached.

- Another way to eliminate document staleness in a cache are conditional request. A proxy server may add the field If-Modified-Since (IMS) to the header of a HTTP request. The time indicated in the IMS header designates the last time a cache obtained a new copy from the originating server. The web server will only send the whole document if it has been changed after the given date, otherwise it will only send a short status message. The disadvantages of this approach are additional HTTP requests and relatively high latency times.
- A third approach is client polling. Here a cache periodically checks on the originating servers if the cached documents are still valid. Due to the obvious disadvantages this approach it is only rarely used.

In today's proxy servers usually a combination of the first two approaches is used: If a web server doesn't supply an expiry date, the cache calculates an expiry date based on the experience, that documents that have not changed recently will probably not change in the near future. If a cached document is requested after this date, the cache sends a conditional request to the originating server and generates a new expiry date.

Another approach to eliminate document staleness are invalidation messages which are sent to caches as soon as the original document has changed. Since there doesn't exist a standard for invalidation messages, they are not widely used in the Internet, although they have numerous advantages compared to other approaches. The necessary traffic overhead is low. The cache immediately gets informed about changed documents, and there is no increase in latency for valid documents. It is possible to update invalidated documents at times of low traffic (most likely at night), which results in a more equal use of the available bandwidth.

As stated above, usually only a very small subset of documents is very popular and incorporates a high number of references. So one could think about determining these documents regularly and to prevent them from being replaced in the cache. So if these documents are kept in the cache permanently, the next logical step would be to replicate them. Replication means, that there is always a current version of the document in the cache, and that the cache gets informed if a replicated document has changed on the originating server. With a current version of a document available on different servers not only load balancing is possible, it also results in a geographical distribution of the document.

It has been shown that the combined use of caching, replication, and invalidation messages has several advantages in practical use ([11], [2]), as there are:

- Increased document availability
- Reduced document retrieval latency
- Reduced amount of transferred data
- Higher transfer rates
- More equal use of the available bandwidth
- Low overhead for additional traffic necessary for synchronization
- Load balancing for popular servers
- Elimination of document staleness

Most former work investigating the impact of replication in the world wide web was based on simulations or stochastical models. Our aim was to implement the concept of replication in a real web and proxy server in order to test advanced caching and replication features in a more realistic environment. We have decided to base our work on the Apache web server V1.3.4, which can also act as proxy server. The Apache server is currently the by far most popular web server and its functionality can be extended by modules without the need to change the server itself. This would make it easier to add replication feature to existing web servers in the Internet.

The aim was to transform a previously only passively caching server into a replication server, which actively duplicates parts of other servers. Besides the conversion of existing caches to replication servers, dedicated replication servers can be set up as well. A replication server may also act as a primary server, which allows to create a hierarchical network of replication servers. The selection of which documents should be replicated on a server can either be done manually or automatically based on appropriate heuristics. An additional aim was to make the replicated documents available under a consistent address.

2. The Apache Server and its API

2.1 Introduction to the Apache Server

Apache is currently the most popular web server on the Internet. The May 1999 WWW server site survey by Netcraft [4] found that over 56% of the web sites on the Internet are using Apache. The Apache server is developed and maintained by the Apache Group [5] and is available as open source software for all major operating systems.

Internally, the server is built around an API (application programmer interface) which allows third-party programmers to add new server functionality by linking new modules directly to the server executable. Indeed, most of the server's visible features (logging, authentication, access control, CGI, proxy and so forth) are implemented as modules, using the same API available to third parties. The API covers functionality of file and memory management, sockets, string processing, dynamic and associative arrays, HTTP request processing and subprocesses. It is implemented by the so called server core, which abstracts from the current operating system.

The Apache server is multithreaded and distributes incoming requests to several subprocesses. A multithreaded server is able to process several requests at a time. The number of server processes may vary according to the server load.

Compared to CGI scripts, modules have one major advantage: performance. The use of CGI machinery has fairly substantial associated costs. The most obvious of these is that the server must spawn a separate process to handle each CGI request. More significant for complex applications is, that a CGI script must initialize itself on each request, e.g. configuration files must be read and database connections must be established each time the script is called. Using a server API, the initialization needs to be done only once and no separate process is needed. A connection to a database for example can be kept alive until the end of the server process, which results in faster response times. Another advantage of using a server API is the access to internal functions of the web server, like access control, authentication, smart remapping of URLs or assignment of MIME types to different documents, to which the CGI protocol doesn't give you access.

2.2 The Apache API

Apache breaks down request handling into a set of phases, which are executed successively for each incoming request. A module can select to handle any of these phases. When writing a module for the Apache web server, you have to define some global API structures in your source code which establish the connection to the server core during linking. In these structures you can define configuration directives, which can be used in the config file of the server to customize your module, and a handler function for each phase (also called hooks) of the request processing. Each handler is a C function and must return an integer status code which indicates the server core that the phase has been processed successfully, that an error occurred and the processing should be aborted, or that the handling of this request is declined and control should be passed to the next module.
Currently, the request processing of the Apache server is divided into the following ten phases (see figure 1). If no handler is registered for a phase or every handler declines to handle the request, a default handler in the server core will be called.

1. **Post-read-request**: Handlers of this phase will get called before the actual processing of the request starts. The proxy module for example uses this hook to identify proxy requests and to prevent other modules from processing it.

2. **URL Translation**: In this phase the file or script is determined to which the current request refers.

3. **Header parser**: In this rarely used phase the HTTP header information should be parsed and stored in the internal data structures to make them available to the following phases.

4. **Access Control**: Here a module can check if the client is allowed to access the requested document.

5. **Authentication**: If an authentication is necessary for the requested document, a module can ask for user name and password in this phase.

6. **Authorization**: If an authentication has been performed, a module can check in this phase if it was valid.

7. **MIME-type-checking**: This step derives a preliminary guess of the requested document's MIME type. The decision may be based on the document's file extension, the name of its file, or the document's location in the document tree.

8. **Fixups**: This rarely used phase starts immediately before the response phase. It may for example be used to generate additional HTTP header lines for the response.

9. **Response**: The response handler (content handler) may adjust the HTTP response header and the MIME type to suit its needs and will provide the data which is sent to the client.

10. **Logging**: Finally the processed request may be logged.

Additionally a module may declare handlers which get called during the initialization of modules, virtual servers, subprocesses, and per-directory configuration structures.
Implementing Advanced Caching and Replication Techniques in the Apache Web Server

Each handler of a phase will be passed a data structure called `request_rec` which contains and collects all relevant information about the current request.

### 2.3 Limitations of the API

The Apache API is in particular organized/designed to process HTTP-requests. The modules get only activated during the processing of a request and most API calls require the above mentioned `request_rec` as one parameter. All these facts make it difficult to perform concurrent tasks within Apache. If a process is spawned during the initialization of a module, many of the powerful API functions cannot be used because their code would not be executed in the context of a request and so no valid `request_rec` structure would be available.

It is currently not possible to add new protocols to the Apache server without doing major changes on the server core. Since it was necessary for our project that two Apache server can exchange information, we have decided to encode this information into special URLs which are recognized by a server if our modules are installed. The Apache API also doesn't enable different modules on a server to communicate with each other. We have solved this problem by sending HTTP requests to the own server with the communicated information encoded in the URL.

### 3. The New Apache Modules

#### 3.1 Overview

In order to incorporate advanced caching and replication features in the Apache server, the following three Apache modules have been implemented:

1. The first module extends the proxy functionality of the Apache server with the ability to replicate documents from the world wide web. For storing and managing the replicated documents, the existing infrastructure of the proxy cache has been reused in parts. The new module works close together with the Apache proxy module and requires its installation. It is able to receive and process invalidation messages from other servers. These messages can be sent by a primary server in order to inform about changed or deleted documents, if the enhanced proxy server is registered as a replication server. Combining the paradigms of caching and replication has been called "Caching goes Replication" (CgR) in [2], that's why we named this module `mod_cgr`.

2. For the primary servers a new module has been created, that accepts registrations of replication servers, which want to replicate a certain document of the primary server. This module, which got the name `mod_replication`, periodically checks for changed and deleted documents in the file tree. If necessary, it sends invalidation messages to all registered replication servers. These messages will be received and processed by these servers in the module `mod_cgr`. The module extends the header of every HTTP response with a list of all known replication servers of the requested document. Optionally this list may be added as hyperlinks to the bottom of each HTML document.

3. The information about replication servers, as part of the extended HTTP header, will be extracted and saved by the third new module. By storing this information, replicated documents are available although the primary server cannot be reached. Due to the automatic extraction of the information, the data base is continuously growing. This approach has been called "Web Location and Information Service" (WLIS, pronounced "Willis") in [2], giving the module its name `mod_wlis`. The automatic collection of information about replication servers can only work on a proxy server, because only here external documents are passing the server.

Additionally, some minor changes on Apache's proxy module were necessary. The modules `mod_replication` on the primary server and `mod_cgr` on the replication server have to communicate with each other, e.g. in order to register as a replication server for certain documents or to send invalidation messages. Due to the problems with the Apache API mentioned in the previous chapter, it was difficult to implement this communication as a new protocol or to make the modules listen on a special TCP port. So we decided to send commands to the different modules encoded into the URL of HTTP requests. An Apache server which is enhanced with the new modules will recognize these special requests and perform the necessary actions. The URLs used for this task all have a common structure:

\[
http://servername/_modulename/command/data
\]

The data usually is a path, which identifies a certain document. The underscore as prefix of the module name should protect from collisions with the namespace of the web server. Each module implements a specific set of commands, as we will see later. If no command is given after the slash, the module will return its name, its version number and the set of commands it supports. Performing the communication using this kind of HTTP request has the additional advantage, that the modules may be controlled using a normal web browser. In the prototype version, security policies for the receipt of these commands are not implemented. Ad hoc solutions such as password based authentication between communicating servers or authentication on the basis of IP addresses could be implemented, but they are not manageable in a realistic environment.

Figure 2 gives an overview of the interaction of the two most important modules `mod_cgr` and `mod_replication`. The big arrows show possible HTTP requests while the small arrows represent internal data exchange. In the following chapters the new modules will be described in more detail.
3.2 The module mod_replication

The module mod_replication must be installed on a primary server, in order to initiate the replication of documents. Any proxy server configured with the module mod_cgr is able to inform the primary server with the "subscribe" command, that it intends to replicate a certain document and that it wants to be informed about future changes on this document. For every document on the primary server a list of replication servers is maintained, which is stored on an LDAP server. Since Apache modules are only active during certain phases of the request processing, the module mod_replication starts a subprocess during initialization, which periodically checks for changed and deleted documents in the document space of the server, using the last modified date of the file system. If a document has been changed or deleted, an invalidation message in the form of a HTTP request will be sent to the replication servers and will there be processed by the module mod_cgr.

The module mod_replication currently implements five commands, which can be executed with a HTTP request of the kind

\[
\text{http://servername/\_replication/command/path.} \quad \text{"path" is the path where the addressed document can be found on server "servername".}
\]

subscribe: Using this command, a replication server can inform the module mod_replication on the primary server, that it wants to replicate the specified document. As a result of this command the requesting server will be added to the list of replication server and the HTTP status code "200 OK" will be returned if no error occurred.

unsubscribe: Using this command, a replication server can remove itself from the list of replication servers of the specified document.

collapsed: By this command, the module mod_replication is notified about changed documents. This is done by the module itself in case of documents stored in the file system. The command can also be used to inform about changed documents explicitly, for example in case of documents that are provided by another Apache module (e.g. replicated documents that are taken from the cache and returned to the client by the module mod_cgr).
deleted: Notification about deleted documents (special case of the "changed"
command above).

listrs: This command can be used to retrieve a list of all replication servers for a
given document. The list will be returned as HTML document containing a
hyperlink for each replication server.

For each document which is returned by the server, the module mod_replication checks in its database, if replication servers are registered for this document and adds
to them to the HTTP header of the response using the new field "Replication-Server". This information can be evaluated by clients or proxies. Optionally this list may be
added as hyperlinks to the bottom of a HTML document.

3.3 Changes to the proxy module

Since the module mod_cgr cooperates very closely with the proxy module and uses its cache for storing the replicated pages, some minor changes had to be done to the
proxy module itself. Additionally it has been extended with three new functions. First of all, a proxy log has been added, which logs each proxy request in a file. The
module mod_cgr periodically generates a statistical analysis based on this log file, which can be used as a basis for the decision, which documents should be replicated.

The second new functionality is useful for quantifying the stale rate of the cache. If enabled, it checks for each proxy request resulting in a cache hit, if the delivered
document was up to date by sending a conditional request to the originating server. The result is added to the log file entry of the request. The stale rate of a cache is the
percentage of documents, where newer versions did exist on the originating server at the time the cached document was delivered. Besides the hit rate the stale rate is the
most important value when rating/benchmarking caching techniques for the world wide web. Unfortunately it cannot be derived from the log files without such additional
measures.

It is a common problem in the WWW, that a requested document cannot be retrieved, because the server, on which the document is located, is currently not reachable. In
this case our enhanced proxy module asks a possibly installed module mod_wlis for a list of replicas of this document. It then tries to get one of these replicas instead of
just returning an error message to the user.

3.4 The module mod_cgr

The new module mod_cgr extends the proxy module of the Apache with the ability, not only to cache documents, but also to actively replicate them. The selection which
documents should be replicated may either be performed manually or automatically using an appropriate heuristic. The decision of the heuristic may be based on statistical
data which is periodically generated from the new proxy log. The replicated documents are stored in the cache of the proxy module. The advantage of this approach is that
parts of the infrastructure of the cache may be used by the new module too and that the proxy module will automatically find replicated documents when processing proxy
requests.

A replicated document may belong to one of currently three different replication types which differ in their effect when receiving an invalidation message:

*immediate*: If a replicated document belongs to this type, it will be updated immediately after
the receipt of an invalidation message.

*nextaccess*: Upon receipt of an invalidation message, the concerning document will be
marked as stale in the cache, if it belongs to this replication type. The document
will be updated when it is accessed the next time.

*deferred*: Documents of this type will be, like in the previous case, marked as stale and
updated on the next access. Additionally, these documents will be updated at
configurable times. This allows to update outdated documents at times of low
traffic, e.g. by night.

The module mod_cgr also implements a set of commands, which can be sent to the module as HTTP requests of the structure

http://servername/_cgr/command/url, where url is the URL of the requested document without the prefix "http://".

*replicate*: This command can be used to start the replication of the specified URL at the
addressed proxy server. The module will send the "subscribe" command to the
originating server and download the document with a common HTTP request.
If the document is already present in the cache, a conditional request will be
sent.

*stopreplicate*: Using this command, the replication of the specified document can be stopped.
The "unsubscribe" command will then be sent to the originating server and the
document will remain as a normal entry in the cache.

*changed*: With this command the module can be informed, that a document has changed
on the originating server. Based on its replication type, the document will be
updated immediately or will just be marked as stale. A possibly installed
module mod_replication will also be informed about the change, because the
document may be replicated again by other servers. Due to limitations of the
Apache API we mentioned earlier, this is done with a HTTP request
addressed to the own server.

*deleted*: Similar to the previous command, the module can be informed about no longer
existing documents.
This command is used internally for the scheduled start of the garbage collection, the statistical analysis, and the deferred update of replicated documents. Because Apache modules are only activated during the processing of a request, the module mod_cgr starts a subprocess during the initialization of the server. This process generates a HTTP request at specified times in order to execute some or all of the above actions.

When initiating the replication of a document, downloading it from the originating server may be unnecessary because the document is already present in the cache and has not changed. Therefore, it is better to postpone the clean up of the cache (garbage collection) until the candidate documents for replication have been chosen. Because of this, the control over the execution of the garbage collection has been moved from the proxy module to the module mod_cgr.

Usually a document can be accessed in the world wide web under a unique name, its URL. Because a URL always contains a server name, it is not possible to route a HTTP request directly to a replication server without knowing its name in advance. Solutions based on the necessary DNS lookup, where the responsible DNS server returns different IP addressees are well known. But this approach was not suitable for us, because it only allows the replication of whole servers.

In this project, only the very popular documents from different servers of the WWW should be replicated. One aim was to make replicated documents available under a consistent name on all replication servers. The URL of a replicated document always has the following structure: \texttt{http://servername/replicas/url}, where \texttt{url} is the URL of the document on the originating server without the prefix "http://". So if for example the document
\begin{verbatim}
http://www.server1.com/path/doc1.html
\end{verbatim}
is replicated on \texttt{www.server2.de}, it can be accessed there under the name
\begin{verbatim}
\end{verbatim}
If \texttt{www.server2.de} is part of a hierarchical network of replication servers, it may also act as a primary server. This means, that the already replicated document may be replicated again by another server \texttt{www.server3.de}. Here, the document would be accessible under the following addresses:

\begin{verbatim}
\end{verbatim}

In practice the first address will normally be used, because it is shorter and consistent on all servers.
We already mentioned earlier, that the decision, which documents should be replicated, can be done manually or automatically by an appropriate heuristic. In order to provide a basis for the heuristic, the module mod_cgp periodically analyzes the proxy log and generates a statistic for each accessed document. This statistical data is stored on an LDAP server and contains information like documents size, number of accesses, number of changes, last modified date, date of last request, average retrieval time, cache hits and misses, byte hits and misses, number of errors and so on. All this data is available for the last period and also as cumulated values. That way a heuristic may use short term as well as long term data to derive its decisions. Currently, in a simple heuristic, documents with the highest byte miss rate are selected as candidates for replication.

3.5 The module mod_wlis

The mirroring and replication of FTP servers has been used in the past in order to move the data closer to possible users and to balance the load between several servers. But in these cases the user must gather the information about available mirrors and must explicitly decide which mirror to use. And if the primary server cannot be reached for some reason, it can be difficult to get a list of its mirrors. This problem is addressed by the Web Location and Information Service (WLIS). The main purpose of this name service is to maintain a distributed database which contains information about replication servers of documents. This information is added to the HTTP header by the module mod_replication and is then automatically collected by the module mod_wlis. Because of the consistent paths of the replicated documents, only the name of the replication server has to be transmitted and saved. The HTTP request http://servername/_wlis/listrs/url returns list of hyperlinks to replication servers for the specified URL.

4. Further Work

The modules mod_replication and mod_wlis add hyperlinks to a list of replication servers at the end of a document. By these means, they inform interested clients about document replicas without any changes in the browser code. Users may switch to another replication server (or the primary server) by selecting one of these hyperlinks, e.g. in case of network congestion. But in practice, this functionality has some problems: Often the layout of a page is destroyed and with pages using frames this approach also doesn't produce the intended result. And for non HTML documents like media files links cannot be shown at all. A reasonable solution for this problem would be a browser plugin, which extracts the list of replication servers from the HTTP header and presents them as a dropdown menu. Additionally, this plugin may switch automatically to a different server if the transfer rate drops below a configurable threshold.

Currently, a primary server does not know about the exact number of document accesses because many requests will be served by a proxy cache or replication server. As a consequence, the access statistics of a primary server shows fewer hits than the actual number of document accesses. This is especially a problem for commercial web sites, which make money with banner advertisements, because the price for a banner ad is closely related to the number of page hits of the site. Because of this, many
Implementing Advanced Caching and Replication Techniques in the Apache Web Server

webmasters of commercial sites supply an expiry date in the past with their documents, which prevents any caching of these documents by browser and proxy caches. A natural solution for this problem would be, that the replication servers periodically report the number of document access of the replicated documents to the primary servers.

The continuously growing percentage of dynamically generated documents in the world wide web cannot be cached or replicated without major efforts. Because of server side includes and embedded scripts it is sometimes difficult to identify dynamic pages. So, before a server allows the replication of one of its document, it should check if the document contains dynamic parts.

Under certain circumstances it could be useful if a primary server is able to initiate the replication of certain documents. If for example an update of a very popular software is released, the update and the information about it could be replicated on several replication servers immediately after its release. This feature could be easily added to the existing modules.

If a replication server is not available any more, there's currently missing a possibility to remove it from the database of the WLIS modules. Since the WLIS information is automatically propagated over the entire internet, piggy-backed with the conventional HTTP response, and is stored locally, a reasonable solution for this problems seems to be difficult to find.

Replication can be used to balance the load between several servers, but a client usually accesses the primary server first, which results in a still very high load of the primary server. So under heavy load it could be a good idea to redirect HTTP requests to a different server using the HTTP status code 302. Ideally one would chose a server which is close to the requesting client. Unfortunately no reliable methods exist for determining the distance between two hosts in the network topology.

5. Conclusions

In this project replication based on invalidation messages has been added to the Apache web server by developing three new Apache modules: mod_replication, mod_cgr and mod_wlis. For each document, the module mod_replication keeps track about existing replication servers. It checks autonomously for changes in documents and reports them to the replication servers by sending invalidation messages coded into a HTTP request. Additionally it adds the information about replication servers to the HTTP header of each delivered document. The module mod_cgr can be installed on a proxy server which should be upgraded to a replication server. It controls the selection of the replicated documents on the base of statistical data derived from the proxy log. The module registers itself as registration server the originating server of each replicated document and controls the updating of invalidated documents. Currently three different replication types are available, which result in an immediate or delayed update of an invalidated document. Last but not least the module mod_wlis extracts the information about replication servers from the HTTP header of each document, which passes the proxy server, and collects them in a database. This information can be used to address an alternative server if the desired server cannot be reached or is too slow.

In comparison to writing a new server from scratch, implementing the new functionality as modules using the Apache API has saved us a lot of time and work. Additional advantages are that the new replication features are part of a fully featured HTTP server and that they can be easily added to already running web servers without much effort. We have found pragmatic solutions to overcome the current limitations of the Apache API, which is currently mainly designed for the processing of HTTP requests. For future releases of the Apache server, a better support for concurrent processes and inter module communication (possibly between different servers) would be desirable.

For performing basic tests with the new modules, we have set up a small network of clients, primary and replication servers and generated artificial traffic in this network. But performing tests under real world conditions has been proven to be difficult, because no important server has our replication module installed yet. Webmasters interested in installing these modules can download the source code from http://www.jam-software.com/jm/paper/.

References


Steffen Rothkugel is research assistant at the System Software and Distributed Systems group at the University of Trier since 1997. Diploma thesis in Computer Science at the University of Kaiserslautern. His research interests currently focus on support for user mobility and object-oriented middleware.

Peter Sturm: Diploma Thesis (1987) and Ph.D. (1993) for Computer Science at the University of Kaiserslautern, Germany. Since 1997, Professor for System Software and Distributed Systems at the University of Trier, Germany. Current research interests: Caching and Replication in the WWW, User Mobility, and highly dynamic distributed applications.