Design and Applicability of a Transparent Network Redirection Layer

Pedro José Marrón and Georg Lausen

Universität Freiburg
Institut für Informatik
Georges-Koehler-Allee, Geb. 51
79085 Freiburg, Germany
{pjmarron, lausen}@informatik.uni-freiburg.de

Abstract. In this paper we present the design, applicability and preliminary experimental results of the network redirection layer developed at the University of Freiburg as part of PLEXOR, a load-balancing and fault-tolerance middleware framework. The non-invasive nature of our layer allows all dynamically-linked network applications to benefit from our framework without the need to recompile them, which makes it a very elegant solution to solving some of the problems introduced **by design** in Internet protocols like HTTP, the Internet Caching Protocol (ICP), and FTP, among others. In the former two protocols, the inability to specify more than one proxy server may lead to decreased performance, or even the report of phantom network failures which could be easily solved by providing an alternate server. In the case of FTP, the inability to specify a set of mirror sites selected at download time based on the current load also leads to a decrease in performance. Rather than trying to find a solution for each particular protocol in isolation, we have opted to design a general middleware framework that, not only addresses these shortcomings, but also provides additional services, like load-balancing, group membership, etc. without incurring in any noticeable overhead.

1 Introduction

Two of the most widely used keywords, as well as the most desired features, in the Internet today are *interoperability* and *performance*. In fact, interoperability was the driving force that shaped most of the protocols in the former days of the Internet. However, some of them have shortcomings that have started to become apparent as we gain more experience, which make them less than optimal for the kind of work (and performance) expected by most users nowadays.

Traditionally, interoperability is achieved in the following two ways:

- Definition of standard protocols that everybody should adopt.
- Design and implementation *a posteriori* of wrappers and abstract layers that hide the differences of two incompatible protocols, in case there is no clear standard.

Performance, on the other hand, can be achieved in the following ways:

- More bandwidth: Hardware additions like more routers or more data lines (out of the reach of most Internet researchers).
- Improvement of current routing protocols, like IS-IS [Cal90] or BGP4 [RL95].
- Caching/Mirroring: Taking advantage of the collaborative nature of the Internet to bring the most commonly requested information as close to the clients as possible, and, therefore, leveraging topological locality factors.

Because more bandwidth is not attainable by software and requires the investment of large amounts of money, and the adoption of new protocols usually takes a long time, mirroring and caching information has become a popular way to attempt to improve performance on the Internet [BDH+94,BCC+95].

However, current protocols either do not have support for caching at all, like FTP [PR85], or when they do (HTTP 1.1 [FGM+97] or ICP [WC97]), they do not support a community of caches or mirror sites that perform any kind of load balancing to try to improve on their average response time.

Section 2 of this paper describes the generic architecture of PLEXOR, the middleware framework we are currently developing at the University of Freiburg, and that tries to overcome some of the shortcomings detailed above. Section 3 concentrates more specifically on the redirection layer, and the techniques used to provide interoperability and transparency even on non-native applications, that is, applications that were not originally designed for our framework. Section 4 provides experimental data on the overhead incurred by our middleware on non-native applications. An overview of some of the systems we are building on top of PLEXOR, as well as their ability to solve the afore mentioned protocol problems is discussed in section 5. Finally, related work is compared in section 6, and section 7 concludes this paper.

2 Plexor Overview

Originally designed to provide a better level or service and smaller computational requirements on a node of computers running the FLORID [FHK+97,MM99] database system, at the University of Freiburg, PLEXOR has evolved into a generic set of middleware services that enhance the level of availability, reliability and scalability, not only of distributed database systems, but also of any other distributed application, independently of the distributed paradigm used (CORBA, RPC or raw TCP/IP). Furthermore, PLEXOR provides for an elegant and efficient solution to overcome some of the shortcomings of caching and mirroring protocols on the Internet, as we will see in more detail in section 5.

Our goals in designing the system aim at achieving the following characteristics:

Fairness: Through the use of a load balancing algorithm, no one particular node in the system should be overloaded while others sit idle.
Adaptability: Through the definition of generic computation nodes, the system can adapt on-the-fly to changes in network topology and network type.
Scalability: A well defined hierarchical structure for complex computations allows for scalability and adaptability where needed.
Dynamism: Nodes and services can be added and removed at will with zero down-time.
Transparency: Computation redirection, task migration, and on-the-fly redirection occur transparently at the middleware layer, without requiring intervention from the application layer.
Lightweight: All modules and protocols impose minimum overhead on the network.
Reliability: Down times as short as possible.
Fault-tolerance: Where applicable and doable, no single point of failure should compromise the stability and reliability of the system.
Real time: Through the use of an admission policy, soft real time guarantees can be made.

Therefore, we have designed our system as an integrated set of independent modules to allow for maximum flexibility. The following section gives a brief description of the each one of the modules that compose our system. For more detailed information, as well as for a thorough explanation on how the modules work together to provide the level of services and breadth of characteristics described above, see [Mar00].

2.1 Module description

Our framework is made up of the following modules:

Membership module: takes care of managing groups of nodes, as well as determining whether or not members are needed for particular services.

Configuration module: keeps information about the particular resources of a node. This information is used to determine what node should be selected by the membership module.
Redirection module: migrates a connection from one node to another on demand.
Task migration module: provides the infrastructure needed to migrate a task or service from one node to another.
Profiling module: collects data on the performance of a particular set/type of nodes.
Policy module: makes the system adjust its parameters for load balancing, task migration, admission policy, etc.
Communication module: takes care of managing the details of different distributed paradigms so that our system can transparently talk to CORBA objects, RPC services, Java Beans, etc.

The redirection layer can be thought of as the collection of modules that provide the following services:

- Offer a completely transparent interface to allow non-native applications to anchor into our system, and consequently, benefit from it.
- Perform the monitoring and, if needed, the migration of connections and tasks from one node to another.

In this paper, we concentrate mainly on the first of these services, its feasibility, and its applicability to solve some of the problems mentioned earlier.

3 Redirection Layer

The redirection layer, composed of the profiling module, task migration and redirection module implements the core functionality that allows non-native applications to anchor into our system and benefit from it.

The redirection module sits on top of the communication layer and is responsible for breaking up and establishing a new connection between a pair of nodes.

There are two kinds of redirections that occur in our system:

- Continuous stream redirection (TCP based)
- Datagram redirection (UDP based)

In the first case, the connection between the two nodes must be broken and reestablished using the new node. If there is saved state in one of the nodes, it has to be transferred before the connection can be established again. Therefore, the Redirection module works closely with the Task migration module to achieve its goals.

In the second case (UDP based), there is no permanent connection established between two nodes, and therefore, no transient connection state to be saved. The next request can be redirected to the new node without affecting the result of the computation.

The redirection module then contacts the membership layer and uses its membership information to determine which node is a possible candidate for redirection. Once it has selected a candidate, and based on the information from the profiling module, the redirection protocol starts performing its function.

3.1 Non-native Applications

In the case of non-native applications, that is, of applications that have not been designed to work with our system, the redirection layer needs to effectively "high-jack" an existing connection, and re-establish it using another (hopefully) less overloaded node. In order to do that, and taking into account that this layer has no help from the underlying application, we have implemented a wrapper around the socket library functions similarly to what Baratloo et al. have done to implement their `libsafe` library [BTS00].

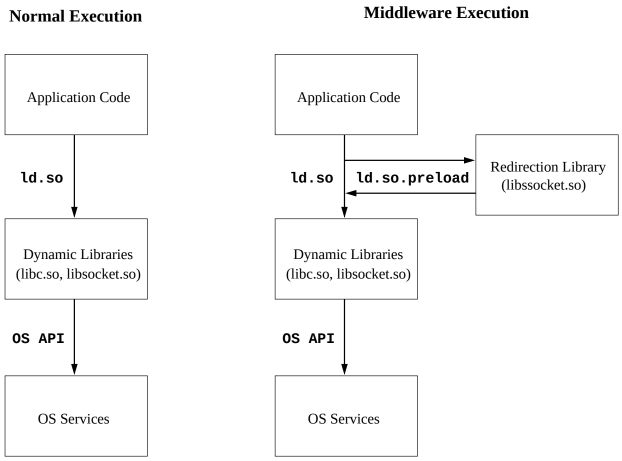

Fig. 1. Replication Layer

Figure 1 depicts the anchor mechanism of our framework. On the left we can see the normal execution path of a dynamically linked application, and on the right the path it would take if our framework is installed. In the first case, the application makes use of the dynamic loader (`ld.so` in most Unices) to resolve symbols defined in libraries like `libc.so` or `libsocket.so`. In the second case, the process is slightly different because our framework makes use of the dynamic preloader (`ld.so.preload` in most Unices) to provide a rewriting of all socket functions in the `libssocket.so` library. Then execution continues as in the normal case.

The advantages of using this method are obvious:

- No need to recompile network applications to work with our framework.
- Integration is completely transparent to the application.
- No operating system modifications are necessary. Therefore, systems like Solaris, where the source code is not available can also benefit from our framework.
- Easy installation and deinstallation.
- Allows for selective use of our framework by means of environment variables understood by the preloader.
- Overhead is negligible (see section 4).

Nevertheless, this technique also has slight disadvantages:

- Only usable by dynamic applications.
- Modifications at the operating system level generally yield better performance, since the TCP/IP stack is implemented in the kernel.

As for the first disadvantage, almost all applications developed for Unix are dynamically linked, and only in extreme cases, where the purpose is either to create a minimal bootable system, or an extremely secure system, are static applications in the higher numbers. Even secure programs, like `ssh` are dynamically linked, and, therefore, are also suitable for our framework.

As for the second disadvantage, it is only a problem if the difference in performance between running our framework inside the kernel and outside it, is big. There is a tradeoff between portability and performance, and we believe that the former clearly outweighs the latter, as supported by the experimental results detailed in the next section.

4 Experimental Results

The feasibility of our solution is not only backed-up by the fact that most applications are dynamically compiled, but because our framework imposes minimal overhead on the execution time of non-native applications running on top of it.

The redirection layer processing can be divided in two phases: In the first one, the profiling module gathers the data needed to determine whether or not the connection should be redirected to another node in the system, and in the second phase, the actual redirection is performed.

As far as overall performance is concerned, the data gathering phase is the one that could potentially introduce a non-trivial overhead in applications that happen not to need any redirection in their execution. Therefore, we have tried to lower as much as possible the data gathering overhead, while at the same time supplying the rest of the framework with enough information to perform an informed decision on whether or not redirection is needed.

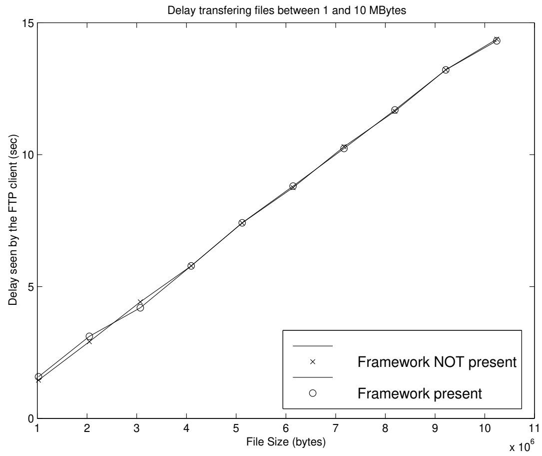

Fig. 2. Big File Graph over LAN

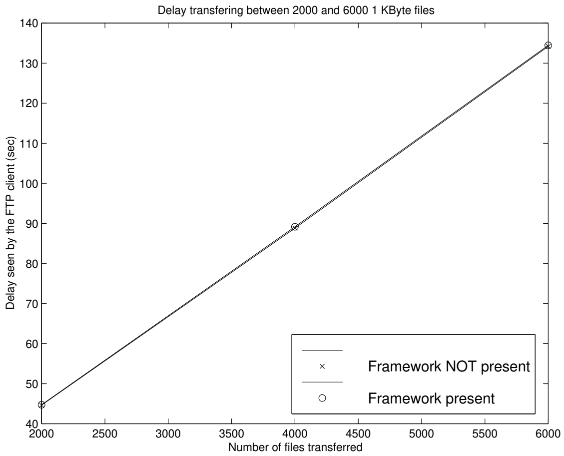

Fig. 3. Small Files Graph over LAN

4.1 Experimental Setup

In order to determine whether or not the overhead is noticeable, we executed the command-line FTP client installed by default on Linux Red Hat 6.1, and measured what the transfer time inside and outside our framework was.

PLEXOR was instructed to record information about the following events:

- Connection establishment,
- size and frequency of send and receive events,
- socket options changes,
- connection tear-down.

In the process of running the experiments, over 600 MBytes worth of logging information were generated. Most of it, though, was debugging information that should not be present when running our framework in "production server mode", because it unnecessarily slows down the data gathering process.

In the first set of experiments, we transferred files between 1 and 10 MBytes between two Linux computers inside our LAN, and recorded the delay reported by the FTP client. Figure 2 shows the resulting graph. In this, and all subsequent experiments, the transfers were repeated 100 times, and the averages plotted in the graph for both, the FTP client running alone, and the FTP client running as part of our framework.

The second set of experiments aimed at discovering whether or not transferring a large number of small files, as is typical in most Web applications, would produce a dramatically different result. Figure 3 shows the resulting graph of transferring between 2000 and 6000 1 KByte files. As with the other graph, both, the FTP client alone, and in the presence of our framework are shown.

Finally, we repeated both experiments over the Internet, yielding almost identical results to those depicted on figures 2 and 3.

4.2 Experimental Conclusion

As we can see in the graphs above, even though our framework was collecting quite a bit of information at the socket level, as well as logging over 600 MBytes worth of debugging information, the overhead imposed is negligible.

Only when transferring files of size 1 and 2 MBytes (in figure 2) is the delay experienced by the FTP client in the presence of our framework consistently higher. For the 1 MByte file, this difference is 0.1274 seconds, and for the 2 MBytes file, 0.2019 seconds. In any other cases, the difference is masked by other factors, like disk activity, and is therefore no longer noticeable.

In the case of using our framework in a Wide Area Network, like the Internet, factors like network delay, packet loss, etc. outweigh even more strongly the overhead imposed by our middleware, making it an ideal candidate to solve problems and deficiencies associated with protocols designed for this environment.

5 Applications of our Framework

Using PLEXOR [Mar00] as a basis, and trying to solve the problems in caching and mirroring inherent to protocols like HTTP, FTP and ICP, we are currently building two additional systems: HLCACHES and MICOM.

5.1 HLCaches

HLCACHES is a system to manage a set of "Hierarchical LDAP-based Smart Web Proxy Cache Communities". Its purpose is to improve on current Internet caching technology by providing caching proxy servers with the following characteristics:

Hierarchical: In order to benefit from the logical topology of the network and improve on the breadth and depth of the caching strategies employed at the proxy.

LDAP-based: To improve on the storing techniques of documents with a clear hierarchical internal data organization.

Smart: Exchanging enough information among their peers at a particular level of the hierarchy to determine whether or not it is necessary to access the next hierarchy level.

Community-aware: Taking advantage of the functionality provided by PLEXOR to allow changes in the caching topology, selection of the most best candidate at the next level, and being able to address more than one proxy server at the next level.

A browser making use of our system no longer has the limitation of using just one proxy server, as imposed by HTTP or ICP, since the system takes care of selecting the best proxy at run-time based on the profile of the current session.

This selection could happen at the following two locations:

- At the local computer where the browser is running, if our framework also runs on the same computer, and the user selects `localhost` as the proxy server, or
- at the first proxy server selected by the user in the browser preferences.

In either case, the proxy server limit on the HTTP and ICP, as described in [DC00] no longer exists.

5.2 MiCom

Another system we are developing on top of PLEXOR, sharing many similarities with the previous system is MICOM. MICOM allows the user to specify and manage a **Mi**rror **Com**munity, that is, a group of FTP servers that offer the same content.

One of the main differences between MICOM and HLCACHES is the level at which the interaction between the client and the framework takes place. In the previous system, most browsers can be configured in the way explained above to interact with our system, whereas in this case the redirection layer (section 3), and its anchor mechanism for non-native applications plays a decisive role.

By "patching" FTP clients at the socket level, we are able to overcome the protocol restriction without the need to modify each client individually.

6 Related Work

Research related to load-balancing middleware services and caching protocols in Internet has been quite prolific recently, leading to very interesting protocol designs and implementations, some of which we mention briefly in the following paragraphs:

6.1 Load-balancing fault-tolerant middleware

Most load-balancing middleware specifications, like OMG's Fault-tolerant CORBA Specification [OMG99] define a set of services and protocols that encompass their own protocol only. In other words, interoperability with other distributed paradigms is not a concern, as it is in our middleware. On the other hand, in order to benefit from most of the features defined in the new standard, modifications to already existing CORBA applications are required. Again, this is something we are working hard to try to avoid, because there are too many legacy applications that should also be supported.

Other middleware services, like ARMADA, developed at the University of Michigan, provide real time communication services [ABD+98,ASJS96], temporal consistency primary-backup services [ABD+98,ZJ98], and a data dissemination system [MJS97], but the main purpose of this middleware is the creation of a system that supports real-time constraints using commercial-off-the-shelf (COTS) components. ARMADA defines a new API to be used by real-time applications, and is, therefore, not concerned with either interoperability or with supporting legacy applications.

6.2 Caching in Internet

The Internet is full with different flavors of caching protocols, like ICP [WC97], HTCP [VW00], Squid [HRW98], etc. But none of them allows the specification of a logical hierarchical topology of caches, like HLCaches does, so as to take advantage of the natural topology of the Internet. For a good overview of the kind of caching protocols existing in the Internet, see [CMT00]

Only the Harvest system [BDH+94], developed at the University of Colorado - Boulder addresses the hierarchy issue in its caches, but uses it as a means to improve on the integration of heterogeneous information sources. One of the problems is that for this system to work, wrappers for each type of information server must be defined, not allowing for the integration and caching of information not understood by their "agents".

7 Conclusion

In this paper we have presented the design, implementation details and applicability of the redirection layer in Plexor, a load-balancing middleware framework developed at the University of Freiburg. Its services are being used to implement HLCaches, and MiCom, two systems that aim to overcome the shortcomings of current caching and mirroring protocols. Although quite some work has been done in the middleware and caching areas in Internet, our framework is novel for the following reasons:

- It does not require modification or recompilations of existing network applications to benefit from it.
- It can be deployed without having to wait for new standards or protocols to be accepted.

- It addresses some of the problems introduced **by design** in some caching and mirroring protocols, providing a smooth transition towards the new standards, whenever they are adopted.
- It causes minimal overhead regarding profiling data gathering, even on non-native applications.

To the best of our knowledge, no other load-balancing middleware system addresses all the issues our system is able to cope with.

References

[ABD+98] T. Abdelzaher, M. Bjorklund, S. Dawson, W.-C. Feng, F. Jahanian, S. Johnson, P.J. Marrón, A. Mehra, T. Mitton, A. Shaikh, K. Shin, Z. Wang, and H. Zou. Armada middleware and communication services. *Journal of Real-Time Systems*, 1998.

[ASJS96] Tarek Abelzaher, Anees Shaikh, Farnam Jahanian, and Kang Shin. Rtcast: Lightweight multicast for real-time process groups. In *Proc. IEEE Real-Time Technology and Applications Symposium (RTAS '96)*, pages 250–259, Boston, MA, June 1996.

[BCC+95] Azer Bestavros, Robert L. Carter, Mark E. Crovella, Carlos R. Cunha, Abdelsalam Heddaya, and Sulaiman A. Mirdad. Application-level document caching in the internet. In *Proceedings of the Second International Workshop on Services in Distributed and Networked Environments*, 1995.

[BDH+94] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, Michael F. Schwartz, and Duane P. Wessels. Harvest: A scalable, customizable discovery and access system. Technical report, University of Colorado at Boulder, March 1994.

[BTS00] Arash Baratloo, Timothy Tsai, and Navjot Singh. Transparent run-time defense against stack smashing attacks. In *Proceedings of the USENIX Annual Technical Conference*, June 2000.

[Cal90] R. Callon. Use of osi is-is for routing in tcp/ip and dual environments. RFC 1195, December 1990.

[CMT00] I. Cooper, I. Melve, and G. Tolimson. Internet web replication and caching taxonomy. Internet Draft. Network Working Group, June 2000.

[DC00] J. Dilley and I. Cooper. Known http proxy/caching problems. Internet Draft. Network Working Group, March 2000.

[FGM+97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2068, January 1997.

[FHK+97] Jürgen Frohn, Rainer Himmeröder, Paul-Th. Kandzia, Georg Lausen, and Christian Schlepphorst. FLORID: A prototype for F–Logic. In *Intl. Conference on Data Engineering (ICDE)*, 1997.

[HRW98] M. Hamilton, A. Rousskov, and D. Wessels. Cache digest specification - version 5. http://squid.nlanr.net/CacheDigest/cache-digest-v5.txt, December 1998.

[Mar00] Pedro José Marrón. Design of a middleware service for scalable wide area network applications. In *Proceedings of the International Symposium on Distributed Objects and Applications*, Antwerp, Belgium, September 2000. IEEE Press. To appear in September 2000.

[MJS97] G. Robert Malan, Farnam Jahanian, and Sushila Subramanian. Salamander: A push-based distribution substrate for Internet applications. In USENIX, editor, *USENIX Symposium on Internet Technologies and Systems Proceedings, Monterey, California, December 8–11, 1997*, pages 171–181, Berkeley, CA, USA, 1997. USENIX.

[MM99] Wolfgang May and Pedro José Marrón. FLORID, Version 2.2, User Manual, December 1999. Available from http://www.informatik.uni-freiburg.de/d̃bis/florid/manual.ps.gz.

[OMG99] OMG. Fault tolerant corba. OMG TC Document orbos/99-12-08, December 1999.

[PR85] J. Postel and J. Reynolds. File transfer protocol (ftp). RFC 959, October 1985.

[RL95] Y. Rekhter and T. Li. A border gateway protocol 4 (bgp-4). RFC 1771, March 1995.

[VW00] P. Vixie and D. Wessels. Hypertext caching protocol (htcp/0.0). RFC 2756, January 2000.

[WC97] D. Wessels and K. Claffy. Internet cache protocol (icp), version 2. RFC 2186, January 1997.

[ZJ98] Hengming Zou and Farnam Jahanian. Real-time primary backup (rtpb) replication with temporal consistency guarantees. In *Proceedings Intl. Conf. on Distributed Computing Systems*, pages 48–56, Amsterdam, Netherlands, May 1998.