A Platform for a Programmable Proxy Farm

Delfina Malandrino Vittorio Scarano
Dipartimento di Informatica ed Applicazioni "R.M. Capocelli"
Università di Salerno – 84081 Baronissi (Salerno) – Italy
{delmal ,vitsca}@dia.unisa.it

Abstract: **In this paper we describe a project whose purpose is to provide a framework for developing remote intermediary applications on the Web. This framework ensures high availability and tolerance to faults like a distributed Web Server Farm that has redundant, load-balanced servers that provide the highest availability and will easily scale in forecast of future needs. The platform is based on Java distributed objects so that it inherits the benefits offered by the language, such as being type-safe, code mobility, reusability and independence from hardware/software infrastructure. Our project is based on Web Based Intermediaries (WBI) [1], developed at IBM Almaden Research Center.**

1. INTRODUCTION

In this paper we describe a project whose purpose is to provide a framework for developing remote intermediary applications on the Web. The proposed architecture is similar to the well-known Web Server Farms where many servers are housed together in a single location and provide the same "service", completely transparent to users.

A Web Server Farm manages several requests by distributing the workload between different computers in the farm therefore obtaining a scalable platform where service is provided with high availability and fault-tolerance. As matter of fact, a Server Farm uses the power of multiple servers and, in the meantime, provides a network infrastructure that ensures scalability, reliability and low latency access to the Internet content.

A Web Server Farm, usually, consists of hundreds of Web Servers. In general, data are replicated through many servers that all provides the same content. Moreover, servers can be specialized for particular tasks. This means that different servers can perform different task on the base of different factors such as characteristic of the computer, particular system properties, necessity of dedicated hardware/software only available on a particular server. Moreover, a Web Server Farm also encompasses a switch that monitor performances and route requests to the server that is, at the moment, the best server in terms of availability and loads. Further, many load balancing methods can be applied: Round robin, Weighted round robin, Least connection and so on.

Our architecture allows the deployment of proxy applications on several machines that provide the same service "as one", i.e. transparently to the user. In a sense, our architecture is aimed to bring to proxy applications the same benefits that Web Server Farms bring to server-side applications. For this reason, we call this environment a Proxy Farm.

Intermediaries

The information are everywhere, on computer networks, on telephone lines, on any device the data are constantly transmitted from one place to another.

Much of the time this data are transmitted directly from producer to consumer. Sometimes, other entities, the intermediaries [13], can be in whatever point along such data flow and can be programmed to adapt, personalize, forward and produce improvements on the data. The intermediaries supply a powerful and flexible mechanism for programming Web applications.

Generally, the intermediary application can be classified in four classes of functionality: (1) Filtering of HTTP request/response; (2) Customization of whole messages; (3) Transcoding of the content, (4) Additional elaborations according to the contents.

IBM Almaden Research Center has implemented a framework, Web Based Intermediaries [1] (WBI) that allow to personalize the Web by realizing an architecture that simplify the develop of such intermediary applications.

Our project

The goal of our project is to enhance the structure of WBI with a distributed architecture that allows the proxy and its WBI plugins (components) to provide scalable and fault-tolerant services on several hosts while preserving the quick design and deployment of applications offered by WBI.

The system, called RemWBI (Remote WBI), is based on Remote Method Invocation [18] (RMI) offered by Java, and it is composed by two types of components: a Local Proxy that interacts with the client for the management of all HTTP request, and Remote Proxies that constitutes our environment of Web Proxy Farm.

In this paper we describe the status of the project and its future developments.

2 WEB BASED INTERMEDIARIES

Here we briefly describe WBI and introduce the terminology used in the rest of the paper.

To simplify the development of Web Intermediaries, IBM Almaden Research Center has developed WBI (acronym of Web Based Intermediaries, pronounce "webby"). WBI is a programmable proxy server designed for easy development and deployment of intermediaries applications. WBI can be used to implement all form of intermediary, from simple server functions to complex distributed applications.

WBI is written in Java and has reached the version 4.5; it has been used in various contexts [2,3,4,5,14,20] and its technology is the base of commercial products of IBM (IBM WebSphere suite) such as WebSphere Transcoding Publisher [19].

Processing Model

WBI acts as an HTTP request processor receiving a request and sending a response. A transaction flows through a combination of four type of basic stages called MEGs:

Request Editor (RE): it receives a request and can modify it before passing it along a path.

Generator (G): it receives a request and produces a corresponding response.

Editor (E): it receives a response and can modify it before passing it along a path.

Monitor (M): it receives a copy of request and response but it cannot modify the data flow.

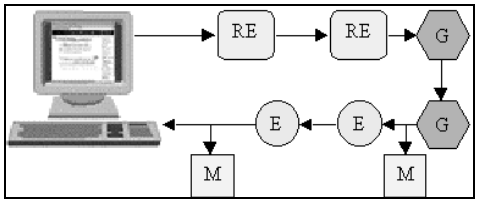

Figure 1. An example of the flow of a request through a WBI proxy

WBI dynamically builds a data path through the various MEGs for each transaction (i.e. a customary HTTP request from a client to the server that passes through the proxy). The system, in order to configure a path for a particular request, uses rules based on a priority associated with each MEG.

A WBI applications is composed by a number of MEGs that operate together for producing new functions. Such a group of MEGs is called a WBI Plugin. WBI instantiates plugins at start-up that register their MEGs with WBI.

When a request is received by WBI it is sent through the MEGs on the base of the rules and the priorities established by the programmer. The final result is returned back to the client.

Some Examples

WBI can operate like a conventional Web server, as a programmable Web server, a transparent Web proxy or can perform any of a range of intermediary functions, such as:

Web Personalization: To change the web's behaviour depending on the user and user's history of browsing [5]. WBI can also annotate hyperlinks [4] with network speed information (any user notes), to save pages viewed for later access, to provide shortcuts links and so on.

Caching: A caching Web Proxy is a simple example of an HTTP intermediary; this intermediary cache checks its store documents to see if the request can be satisfied locally or if it must be sent to another intermediary or to the server [6].

Transcoding: An important intermediary application is the transformation of information from one form to another, such process is called transcoding.

Applications developed with WBI can be directly used with the IBM WebSphere Transcoding Publisher that dynamically adapts Web contents and optimizes it for delivery to pervasive devices.

3. REMWBI

In RemWBI WBI Plugins (or more precisely its components, i.e., MEGs) are made remote by realizing a new architecture that, while maintaining unchanged the original properties, adds new characteristics that allow to use MEGs by remote calls.

Our modification to the original IBM architecture allows MEGs to be used both locally and remotely, i.e. by having several WBI on several Java Virtual Machines on different computers.

Motivations

Load balancing: First of all, this architecture allows to balance requests load over different (Java Virtual) machines. As an example, consider accessing large databases with heavy queries by a WBI proxy. By distributing the queries among different proxies (with replicated databases) the overall load can be easily balanced.

Another example involves an heterogeneous cluster of machines devoted to providing proxy services. Load can be easily balanced among diverse machines and HTTP queries that require particular devices or data (available only on a specific machine of the cluster) can be dealt with by the remote MEG that is able to perform the task more efficiently.

Fault tolerance: RemWBI provides the designer to replicate MEGs on different machines thereby offering tolerance to malfunctioning machines. Requests can be,

therefore, served through the available MEGs on the available remote machines.

4 REMWBI ARCHITECTURE

Our architecture (shown in figure 2) allows to place MEGs that provides services on a Proxy Farm: these remote MEGs can be used by any (local) Proxy that is directly accessed by the client (via the configuration of the browser). In this way, each Local Proxy (where the HTTP request is directed) can either process the data via local MEGs or can invoke remote MEGs from the Proxy Farm.

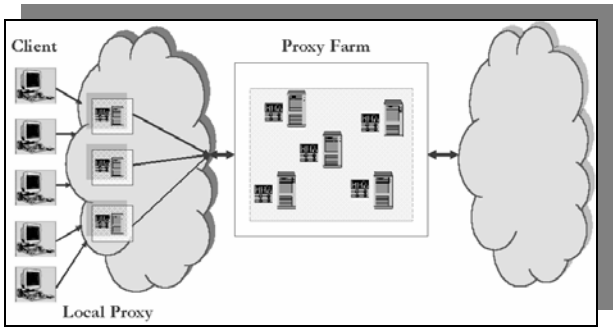

Figure 2. RemWBI architecture's components.

On the other hand, the remote MEGs can be invoked remotely, from any Local Proxy, or locally within the same remote WBI session (within the same JVM). The Remote Proxy, then, offers MEGs that works according to the WBI architecture, i.e. MEGs invoked on the base of specifics conditions and priorities established by the programmer, and at the same time, they can be invoked from remote through RMI.

Since communications between local MEGs (Local Proxy) and Remote MEGs (Remote Proxy of the Web Proxy Farm) occur through RMI, a registry is necessary to allow dealing with remote references. The architecture allows a local MEG to:

- invoke a remote MEG passing the input.
- get the output produced by the remote MEG.
- get the control of the execution.

Unlike a totally local WBI transaction, a remote transaction (i.e. one that involves also remote MEGs) is characterized by a sequence of control transfer between local and remote MEGs.

The logical architecture of RemWBI compared to WBI is described in the figures below: in Figure 3, the standard WBI architecture is described with the request flowing among MEG of different Plugins. In Figure 4 we describe RemWBI architecture that makes the requests flow (possibly) among different machines.

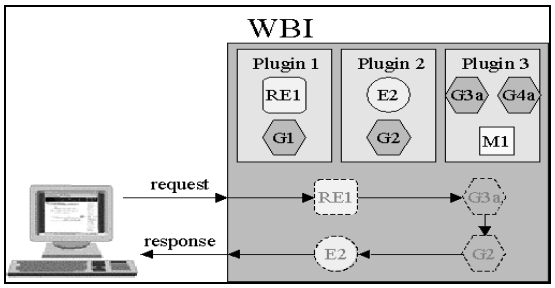

Figure 3. Standard WBI architecture

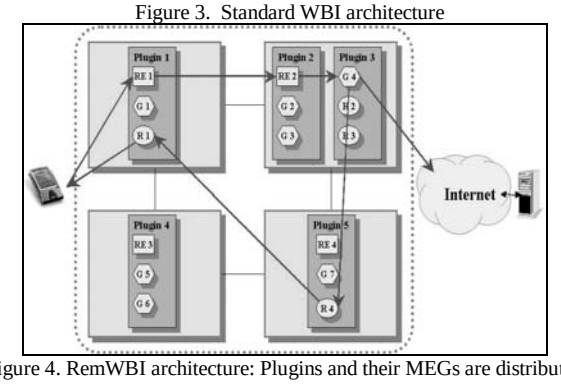

Figure 4. RemWBI architecture: Plugins and their MEGs are distributed and HTTP requests flow through the remote machines.

Local MEG's Operations

The communication between a local and a remote MEG occur through a stub on the local machine.

The steps are described in Figure 5. The local MEG, first, instantiates the proper stub (1) that gets the remote reference of the Remote MEG from the RMIregistry (2).

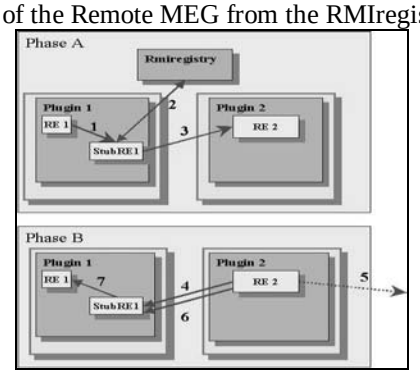

Figure 5. The role of the stub.

The stub is then used by the Remote Meg to exchange data: the stub establishes a connection with the remote MEG sending its own reference (4). In this way, the remote MEG can use stub's methods read() and write() (steps 4 and 6) to take the input flow and return the output flow after performing its own computation (possibly with other MEGS (5)).

There is a different stub implementation for each WBI category (Editors, Monitors, Request Editors and Generators).

A remote MEG presents the following peculiar characteristics with respect to a local one:

- Method initialize() has been slightly modified to allow registering the MEG into the RMI registry.
- Implementation of a remote MEG extends one among RemoteHttpGenerator, RemoteHttpMonitor RemoteHttpRequestEditor, RemoteHttpEditor, instead of the standard WBI HttpGenerator, HttpMonitor, HttpRequestEditor, HttpEditor.
- Every remote MEG must implement the method RemotehandleRequest(MegStub ref) that is used by the stub to send its own reference for callbacks.

5. Related Work

With the explosive growth on the World Wide Web, load on popular servers can increase rapidly and therefore high availability of these servers under pressure becomes increasingly important. Scalability and load balancing have been, historically, primary goals for web servers designers and several solution have been proposed.
Classical solutions to the scalability for Web servers are based on the (balanced) distribution of requests to a pool of federated web servers. Distribution can be directed and guided either server side or client side.
Different techniques can be used: HTTP redirection based methods (CISCO LocalDirector [7], ClusterCATS [8]) balance the load at the HTTP level: requests are, usually, initially dispatched to a redirection server using round-robin DNS, then they are dispatched by the redirection server using the HTTP redirection mechanism. A different client-side technique uses Server Selection algorithms [9] that allow Web clients to select one of the replicated servers which is close to them and thereby minimize the response time of the Web service.
Several servers offer load balancing mechanisms. The first is the NCSA scalable HTTP server [10] that consists of several copies of the (slightly modified) NCSA HTTP server running on different nodes of a cluster. SQUID [11] is a high-performance proxy caching server for web clients, supporting FTP, gopher, and HTTP data objects. Unlike traditional caching software, Squid handles all requests in a single, non-blocking, I/O-driven process. Squid acts as an agent, accepting requests from clients (such as browsers) and passes them to the appropriate Internet server. It stores a copy of the returned data in an on-disk cache.
iProxy [15] by AT&T is an example of programmable proxy that allows accessing, caching, and processing of Web data. iProxy is designed for application that access Web Services. iProxy runs either on the host where the user's browser is running or it can be shared among organizations as a system proxy server. A set of iProxy may also collaborate to share caches and network access.
Finally, an important result is the project of the IETF Working Group that has developed OPES: "Open Pluggable Edge Services" [16]. Their idea is to extend the functionality of traditional HTTP proxy from simple caching to complex intermediary application. Examples of additional services that can be provided by the intermediary (and then by OPES) includes advertisement insertion (add user-specific regional information to web pages), HTML content adaptation, limited bandwidth adaptation, virus scanning, language translation.

6. Preliminary analysis and further work

To analyze the performances of our architecture we have tested a simple (but heavy) proxy application on a cluster of PC's: each HTTP request provides, first, to fetch and compress the requested page that is, then, returned to the client. The proxy application, therefore, provides a simple service of compression of Web pages hat can be particularly useful over limited bandwidth links toward clients (e.g. wireless connections for PDAs).
The cluster consists of six PCs: a server node executes (only) a dispatcher MEG (on the Local Proxy) while the others nodes runs five different (remote) sessions of WBI with the same Plugin replicated on each node.
The dispatcher's task is to receive the HTTP requests from the client and remotely invoke the remote MEGs assuring a correct load balancing among the remote sessions of WBI.

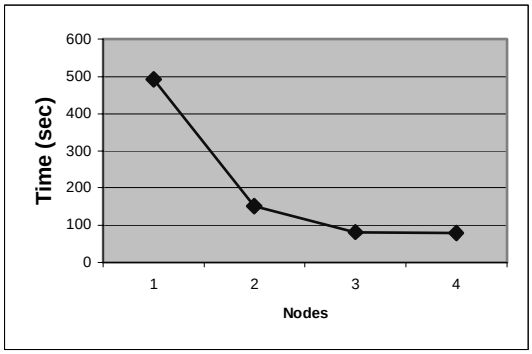

Figure 6. Test on a 100 Mbs Ethernet, 4 nodes, and file size ~1M

The cluster is connected through two network (Ethernet 100 Mb/s and Myrinet 1.2 Gb/s) and therefore we realized different tests using (separately) both networks and different file sizes. To analyze the performances of the architecture we stressed the Local Proxy with an high number of HTTP requests. As client application we used W3C's Libwww [12] by generating 100 requests at once through command line. The dispatcher uses a simple

round-robin technique to equally distribute the MEG invocation calls among the five different Remote MEGs.
In our test, we compute the overall response time, that is the time between the first request and the last response. In Figure 6 and 7 we show the plot of the results obtained.
Let us first comment on the first diagram: the "superlinear" speedup achieved when 2 proxies are managing the requests is explained by the overload of the internal WBI "scheduler" that is overwhelmed by the requests. In fact, performances of the internal WBI mechanism to distribute requests among MEGs was one of the motivating factors in our project. On the other diagram, when 2 nodes are working, the speedup is not as high since the dispatcher becomes loaded to deal with the responses (given the high bandwidth provided by Myrinet).

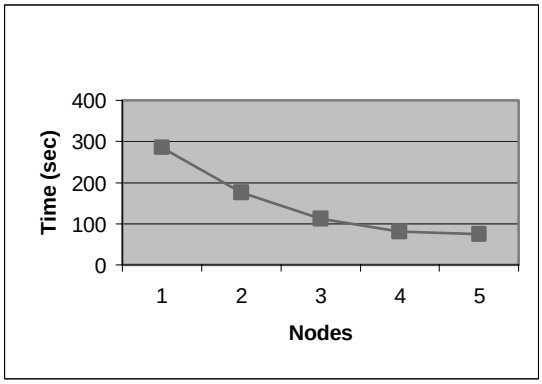

Figure 7. Tests on 1.2Gbs Myrinet, 5 nodes and file size ~1M

When further nodes are added to the cluster, the dispatcher receives a high number of responses by the remote MEGs that (especially with a fast interconnection network such as the Myrinet) are efficiently answering RMI invocation calls. As a matter of fact, the remote computation on MEGs is so fast (even with as few as 3 nodes) that the dispatcher receives responses while still distributing the (last) initial requests.
In conclusion, our architecture preserves WBI easiness of programming but also offers the possibility to deploy remote MEGs that can be efficiently accessed: the single Local Proxy becomes a bottleneck (because of the WBI internal scheduler) before the remote MEGs are unable to offer the service.
In the scenario that we assumed (i.e. deployment of services by Proxy Farms) this bottleneck is not a problem (since several Local Proxies are present) while in a context of a cluster that provides both Local Proxy and Remote Proxies it can become problematic to ensure a high throughput. We are now studying the solution to the second scenario (a cluster with Local and Remote Proxies) by providing a distributed dispatcher within the cluster.

REFERENCES

[1] Barrett R., Maglio P.: "Intermediaries: New Places for Producing and Manipulating Web Content". In Proc. of 7th International WWW Conference. 1998.
[2] Barrett R., Maglio P.: "Adaptive Communities and Web Places". In the Proc. of 2nd Workshop on Adaptive Hypertext and Hypermedia, HYPERTEXT 98. Pittsburgh, USA 1998.
[3] Barrett R., Maglio P.: "WebPlaces: Adding people to the Web". In the Proc. of 8th International WWW Conference. Canada 1999
[4] Weinreich H., Lamersdorf W.: "Concepts for improved visualization of Web link attributes". In Proc of 9th International WWW Conference 2000.
[5] Anderson C., Domingos P., Weld D.: "Personalizing Web Sites for Mobile Users". In Proc of 10th International WWW Conference 2001.
[6] Hadjiefthymiedes S., Merakos L.: "Using Proxy Cache Relocation to Accelerate Web Browsing in Wireless/Mobile Communications". In Proc of 10th International WWW Conference 2001.
[7] Cisco System. Cisco LocalDirector Version 4.1 Software.
[8] ClusterCATS. Enterprice load balancing. http://www.allaire.com/products/brighttiger/index.cfm
[9] Mehmet Sayal, Yri Breitbart, Peter Scheuermannm RadekVingralek. "Selection Algorithms for Replicated Web Servers." Proc. of Workshop on Internet Server Performance '98.
[10] Eric Dean Katz, Michelle Butler, Robert McGrath. "A Scalable HTTP Server: The NCSA Prototype" Computer Networks and ISDN Systems.
[11] SQUID.Adrian Chadd, Robert Collins, Henrik Nordstrom, Alex Rousskov, Duane Wessels. http://www.squid-cache.org
[12] Libwww - the W3C Protocol Library. http://www.w3.org/Library/
[13] Barrett R., Maglio P.: Intermediares: An approach to manipulating information streams. *IBM System Journal*, 38(4): 629-641, 1999.
[14] M. Hori, G. Kondoh, K. Ono, S. Hirose, and S. Singhal. "Annotation-based web content Transcoding". In Proc of 9th International WWW Conf., Amsterdam (The Netherland), 2000.
[15] AT&T Labs-Research. *i*Proxy: a Programmable Proxy. http://www.research.att.com/sw/tools/iproxy.
[16] Tomlinson G. at. Al., "A model for Open Pluggable Edge Services", IETF Internet Draft, draft-http://www.ietf.org/internet-drafts/tomlinson-opes-model-00.txt, November 2001.
[17] Beck A. and Hofmann M., "IRML: A Rule Specification Language for Intermediary Services", IETF Internet Draft, http://www.ietf-opes.org/documents/draft-beck-opes-irml-02.txt, November 2001.
[18] "The java remote method invocation (RMI) specification". Technical report, Javasoft, Sun Microsystems Incorporated., 1998. http://java.sun.com/j2se/1.4/docs/guide/rmi/ .
[19] WebSphere Transcoding Publisher. http://www.ibm.com/software/webservers/transcoding/ .
[20] M. Barra, Paul P. Maglio, A. Negro, V.Scarano: "GAS: Group Adaptive System". Proc. of Second International Conference, AH 2002, Malaga, Spain, May 29-31, 2002 47-57. Lecture Notes in Computer Science 2347 Springer 2002.