RC 21943 (98658) January 30, 2001 **Computer Science**

An Architecture for Acceleration of Large Scale Distributed Web Applications

Seraphin B. Calo, Dinesh Verma
IBM Research Division
T. J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

An Architecture for Acceleration of Large Scale Distributed Web Applications

Seraphin B. Calo, **Dinesh Verma**
IBM T. J. Watson Research Center,
30 Saw Mill River Road
Hawthorne, NY 10532
scalo@us.ibm.com, dverma@us.ibm.com

Abstract

We present a method for maintaining Quality of Service parameters for Internet Web Services by means of dynamic distributed execution and maintenance of networked applications. While the architecture presented provides performance acceleration, it also enables management and administration of the distributed components of the networked applications from a single responsible point of origination. The method deploys a plurality of proxy servers within the network. Clients are directed to one of the proxy servers using wide area load balancing techniques. The proxy servers download programs from main servers and cache them in a local store. These programs, in conjunction with data stored at cached servers, are used to execute applications at the proxy server, eliminating the need for a client to communicate to a main server to execute a networked application. Our architecture enables the content distribution paradigm deployed for static images and streaming video to be extended to the domain of dynamic pages and general web-based applications.

Keywords:
Distributed Systems, Web Services, Application Acceleration, Content Distribution

1. Introduction

The prevalent way to access web-based applications over the Internet is by using a browser that communicates to a web-server across a number of routers. The web-server typically maintains static content in data files, and composes dynamic content by executing programs, typically cgi-bin scripts or Java servlets. During periods of congestion and traffic overload at the servers, the response time experienced by the client is often poor. Typically, as the number of routers lying on the path between the client and the server increase, the chances of encountering congestion increases, and the user is likely to see degraded performance.

Various approaches to alleviate the congestion within the Internet have been suggested in prior work. These include the IntServ/RSVP [RSVP] signaling approach based on the notion of reserving resources within the network and the DiffServ approach [DIFF] based on the notion of mapping traffic aggregates into several different service classes. However, the deployment of any of these approaches requires changes in the basic infrastructure of the Internet, which creates significant operational difficulties, and is unlikely to happen in the medium time range.

An approach that has been partially successful at reducing user response time is that of caching content closer to the user. A client can then access web-based content from a proxy server from which it is likely to obtain a better response time than by going to the original server. The notion of caching has been extended to that of content distribution networks by a number of companies such as Akamai, Digital Island, Cisco, etc. A content distribution network consists of a network of caching proxy servers to which clients are directed transparently using various wide-area load balancing schemes. The caching approach works well for data that is static and unchanging, e.g. images, video clips, etc. However, the techniques of caching that are commonly deployed in the current Internet do not work well with a large portion of web-accessible content. Data that is personalized to a client, or data that is generated by invocation of programs like cgi-bin scripts or servlets cannot be readily cached at the proxies. For a server offering electronic services over the Internet, non-static data forms a significant portion of their overall data content. It would be advantageous to have a scheme whereby such dynamically generated content, and web-centric applications can also benefit from the presence of proxies.

In this paper, we present an architecture that extends the notion of caching to include application components. As in the case of caching of static data, it is highly desirable that the caching of applications be done so that the administrative and operational control of the data/application resides with the original server, rather than with the proxy server. A solution is needed which accelerates applications while still providing the administrative control of the application from the original server, rather than the proxy server.

In Section 2, we describe the overall environment that is being considered. Section 3 contains a discussion of related work. A detailed description of the architecture for application acceleration is presented in Section 4, and its prototype implementation is described in Section 5. The architecture of a personalization application developed using this framework is the topic of Section 6, and Section 7 addresses alternative approaches. Finally, Section 8 delineates areas for further research work.

2. Environment

The application acceleration architecture (AAA) has been developed for the environment shown in Figure 1, which contains machines belonging to three different administrative entities. A web-browser (blue entity) is used by the client to access a web-server (red entity) on the Internet. The web-server owner has a contract with another organization (which we refer to as an ASP or Application Service Provider) which has multiple sites hosted throughout the network. These sites are shown as the green boxes in Figure 1.

Normally, the client browser interacts with the main web-server (the yellow flows). The focus of the AAA environment is on interactions that result in an application invocation at the web-server. In other words, the client browser (blue box) invokes a cgi-bin script at the web-server (red box). This results in specific operations being done at the server location, and in specific messages being generated at the client.

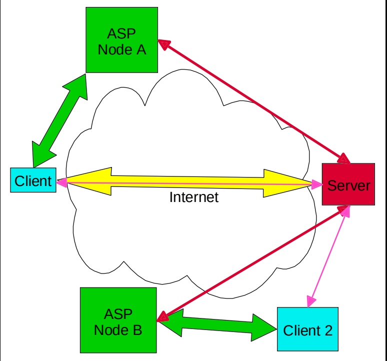

Figure 1: Service Environment

The goal of the AAA environment is to replace the client-server communication by a set of communication flows, one occurring between the proxy and one of the ASP nodes (green flow), and the other occurring between the ASP node and the main server (red flow). There may also be a flow between the browser and the main server (pink flows). However, the magnitude of these flows would be smaller.

The assumption is that there are a large number of applications where the yellow flow can be decomposed into a set of flows for which the green flows predominate, and the amount of red and pink flows are relatively small. For these set of applications, execution of web-based applications at an ASP node would result in a reduced response time to the end user, as well as a reduced load on the network and the main web-server.

In this model, the browser would end up communicating with an ASP node that is close to it, and then invoking a java servlet at the ASP node. This servlet execution may result in invoking some additional servlets/cgi-bin programs at the main web-server. However, the latter invocations would be less frequent and would require less bandwidth.

Note that different clients may access different ASP servers. Figure 1 shows client 1 accessing the ASP node A, while client 2 accesses the ASP node B. We assume that there is a business agreement between the ASP nodes and the web-server owner, that allows for the execution of java programs owned by the web-server at the ASP nodes.

In the original communication, the browser might invoke a cgi-bin script at the main server called the *original-script*. With our application model, the function performed by the *original-script* is split up into two cgi-bin scripts. One of these is invoked at the ASP node; we call this script the *proxylet*. [1]The other script is a *modified-script* which is invoked at the main web-server by the proxylet. This could be a java servlet or any cgi-bin script in any other language.

[1] The term proxylet is found elsewhere in literature, e.g it has been used to describe a component of a proxy within the open extensible proxy services working group within IETF[OEPS], and as a movable program within the active networking literature [ALANREF]. Our notion of proxylet is somewhat different that these other usage, and is used to denote part of a program that is executable at a proxy server.

A proxylet is thus a downloadable program that is fetched on demand from the main server and executed at a proxy within the network. In the first instantiation of the proxylet architecture, we are focusing on Java based implementation of the proxylet concept. Java provides a portable byte-code paradigm that is very well suited for downloadable software. We further focus on http servers and http proxies which form a dominant piece of Internet-based applications. However, the general concepts described here can be applied (with some modifications) to applications written in other languages, as well as those employing other protocols.

One of the key aspects of the proxylet environment is that the configuration, management and control of all applications happens at the main server, and the responsibility of the ASP is simply to provide a suitable execution environment. This provides for a central point of control of application deployment and management, and the enablement of zero-management proxy servers. The ease of management of proxy servers is extremely important to ensure a cost-effective solution to application acceleration.

3. Related Work

The distribution of static data such as images, static documents, and multimedia clips has received a lot of attention within the research community. There is an extensive set of literature of different types of web-caches ([WANG], [BIBLIO]) and Internet standards like [ICP] have been developed in order to address the different aspects of interaction among the web-caches. The replication of static data using content distribution networks is also an established area of activity, with different architectures for content replication being pushed by several companies, e.g. The FreeFlow architecture from Akamai [FREEFLOW] and the Footprint architecture from Sandpiper [FOOTPRINT]. Aspects related to replication of static content data to a network of replicated servers have also been considered [CACHE]. However, due to the inherent nature of caching, the focus of these works has been on caching of static content.

In the area of application distribution, there has been significant research in the field of active networking. In active networks [ACTIVE], network elements, such as routers, are programmable**.** Code can be sent inbound and executed at the router rather than just at the edge nodes. However, given the myriad of security and management issues associated with active networks [ACTIVE2], it is highly doubtful if active networks will ever become a reality. The architecture that we are proposing can be looked upon as a very special case of an active network, one that is likely to be deployed in the context of web-based services and web-applications. There are ideas from the area of active networks that can be borrowed successfully in the context of application distribution, e.g. the concept of a downloadable program that acts like a proxy for a main server, is found in the ALAN active network research project [ALANREF], [ALANPROG]. Our architecture uses some of the concepts from active networks while ensuring that the resulting distributed environment remains manageable.

One of the aspects of determining the right way to distribute applications in a complex network deals with the estimation of the current network statistics and the performance map of the network and the application components. Some of the research related to such estimation is found in [IDMAPS], [GOVINDAN] and [SPAND]. Within the context of content distribution networks, Several research papers have addressed the issue of determining a proxy-server that is closer to a communicating client using various techniques ([Francis] [Sayal] [Schwatz] [OBRACZKA] [CARTER] etc.). All of these server selection schemes can work with the basic architecture provided by the network of proxies created using the Footprint or the Freeflow architecture, or with the architecture presented in this paper. Our work assumes the existence of both a performance monitoring system as well as a system that determines an appropriate proxy server. However, the focus of this paper is in describing the way in which applications are accelerated by using a proxy server.

4. Architecture Description

As indicated in section 2, the AAA environment assumes that a communications network is used to interconnect client devices containing Web browsers to a main web server that provides Web-based applications. Major portions of the application code are moved onto proxy servers that are in closer proximity to the respective end client devices. Communications is maintained between the proxy servers and the main server so that the main server can continue to exercise administrative control over the distributed portions of the applications code. Communications is also maintained between the end client devices and the main server so that the latter can continue to provide applications services that are not readily distributable.

In response to a client service request, a Wide Area Load Balancing module determines the appropriate proxy server to which to direct the request based upon current network performance characteristics and the location of the end client device. The wide area load balancer can be implemented in a variety of manners. One common way to implement it is by means of a modified domain name server. The domain name server, usually abbreviated to DNS server, is the application in the network responsible for mapping machine names to IP addresses. A modified domain name server can return an IP address which corresponds to an appropriate proxy server when a client requests an address for the main server. The appropriate proxy server is determined on the basis of the current network performance characteristics and the location of the client.

An alternative implementation of the wide area load balancer includes a module within the main server that is responsible for redirecting requests to the appropriate proxy server. Such a redirection module might be implemented as a plug-in module among a variety of web-servers such as Apache, NetScape or Microsoft IIS server, which are commonly in use in the industry. The module would look at a table of redirection rules, which specify how requests coming from specific client IP addresses should be dispatched, and use this information to determine the appropriate proxy-server to which the request should be dispatched. The selection of the proxy-server can be based on other criteria included in the rule: e.g., the resource (URL) being requested by the client, or a cookie that is contained within the client's request. The same function can be provided by a stand-alone reverse proxy located near the main server. Some of the algorithms that can be used by a wide-area server are described by [WALS].

When a proxy server receives a request, it determines the set of programs that it needs to run in order to provide an appropriate response to the end client. These programs may be already resident at the proxy server if they had been needed to satisfy previous requests from other users, or if they had been deployed to the proxy server in anticipation of end client requests for basic or popular applications. If they are not locally resident, the proxy server obtains the data and application code appropriate for fulfilling the service request, and caches it locally.

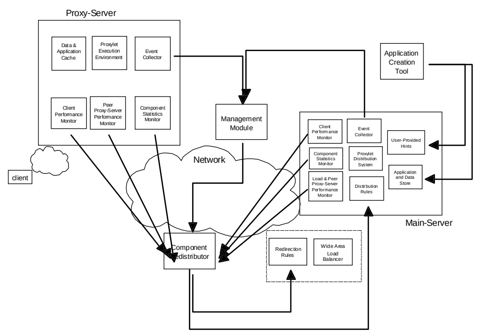
Proxy-Server
Data & Application Cache
Proxylet Execution Environment
Event Collector
Client Performance Monitor
Peer Proxy-Server Performance Monitor
Component Statistics Monitor
client
Network
Management Module
Application Creation Tool
Client Performance Monitor
Event Collector
User-Provided Hints
Component Statistics Monitor
Proxylet Distribution System
Application and Data Store
Load & Peer Proxy-Server Performance Monitor
Distribution Rules
Main-Server
Component Redistributor
Redirection Rules
Wide Area Load Balancer

Figure 2: Application Acceleration Architecture

In order to execute the application, the proxy server needs to determine the execution parameters that may be needed for its invocation. The authoritative source for these parameters is the main server. However, the proxy server caches the parameter information locally. The proxy server then obtains information from the main server regarding which execution parameters to employ in satisfying this specific end client request. The application logic is executed, and any administrative information or error messages resulting from the application execution are logged and sent to the main server.

The mechanism used for caching information about the programs and configuration information within our environment is a proxylet-record. A proxylet-record contains the description of the operations to be performed by the proxy server in response to a URL request from the client. The proxylet record contains information on the programs that need to be executed in order to satisfy the client request, the location from which the programs can be downloaded, and the parameters that ought to be passed to the program upon invocation. The authoritative description of the proxylet records is maintained at the main server, and each proxy server caches this information locally. Standard cache invalidation schemes are used between the proxy servers and the main server to ensure that the information in the proxylet-records stays current.

As an example, let us consider a request from a client which is targeted to the location http://main-server.com/servlet/program1. Let us assume that this request is delivered to the proxy server running at the machine proxy-server.com using a DNS based scheme so that the URL received by the proxy server remains unchanged. The proxylet record for this URL would be keyed by the invocation URL http://main-server.com/servlet/program1, and would include: the information that the code to execute this program can be found at the URL http://main-server.com/proxylets/proxy-program1 ; a list of parameters that need to be passed to the program; an expiration time field after which the record must be revalidated; and, the time when the code being executed at the URL was created.

5. Implementation

We have prototyped the architecture described in Section 4 at the IBM Thomas J. Watson Research Center. As implemented, the application acceleration system consists of a number of web-servers. Each of the web-servers can play one of three roles:

1. A proxy-server: This web-server is within the administrative control of an ASP. It provides for the execution of proxylets downloaded from web-servers playing the role of main servers. Each proxy-server is designed so that it requires minimum configuration and control.
2. A main-server: This web-server is owned by a web-site owner who is the customer of one of the ASPs. The main-server is responsible for controlling the execution of the customer applications.
3. A control-server: There is one control-server within the administrative control of each ASP. The control-server provides configuration information to each of the proxy-servers, and provides information about its customers' main-servers to the proxy-servers.

The proxy-server consists of a standard web-server with a servlet engine. Additionally, the web-server consists of a proxylet class library that provides a set of java functions that assist in the execution of the programs at the proxy server. Each proxy-server is configured with only one parameter, the identity of the control-server to which it should communicate in order to obtain its operating environment. All proxy-servers within the administrative control of an ASP have the same configuration and software information, and can be installed from a single software module.

When started, the proxy-server code generates an identity for itself. The identity consists of the IP address of the proxy-server where it is operational. The proxy-server then contacts the control-server to obtain information about the customer servers that it should be supporting. The control-server uses the identity information to decide which customers the proxy-server ought to be supporting. It then sends the name of the customer class being supported to the proxy-server which is passed as a configuration parameter to the servlet engine at the proxy-server. As currently implemented, each proxy-server machine needs to be dedicated to a single customer. However, it can be easily extended to an environment supporting virtual hosting by expanding the definition of the identity field of the proxy-server.

The support for downloadable code is offered by means of a special library of Java classes. Figure 3 shows the structure of the classes that are implemented within this library using standard UML conventions. The InterceptorServlet is an instance of the servlet which is used to intercept all requests at the proxy-server and process them. This servlet takes the name of the customer and downloads the description of the customer. This description is in the form of a Java class with specific interfaces that describe the behaviour of the customer. The class is downloaded using the CustomerClassLoader, which is a derivative of the standard java URLClassLoader library. The main servlet relies on a proxyletManager class to handle the downloading and maintenance of proxylet classes and information. The proxylet class definitions are downloaded using ProxyletLoader, a subclass of the standard Java URLClassLoader. A ProxyletLogger class provides a means for relaying logging information to the main-server, and a ProxyletException class is used to handle the error situations.

In order to build a distributable application, the application programmer defines a subclass of the

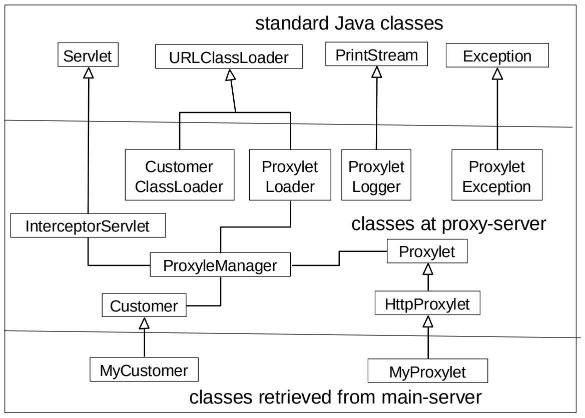

Figure 3

Proxylet. Within the web-environment, a standard implementation of the proxylet interface, the HttpProxylet is provided. This class provides the glue which enables proxylets to run as standard Java servlets. The application developer would typically implement a subclass of HttpProxylet. Similarly, each customer would implement a subclass of Customer to define the rules as to which URLs get processed on the edge, and which run on the main-server only. These are shown as the MyCustomer and MyProxylet classes in the figure.

When a request is received by the proxy-server, it invokes a method within the customer class (which essentially is defined by MyCustomer) to determine if the request should be executed at the proxy-server or simply be relayed to the main-server. When the request is to be executed locally, another method within the customer class is used to retrieve the proxylet-record for the URL being invoked. The proxylet-record is then used to download the appropriate programs and parameters and execute them as defined in the architecture section.

The customer class also contains two methods which return PrintStream objects. The first of these objects is mapped to the standard output of the JVM of the proxy-server, and the other is mapped to the standard error of the JVM. This remapping permits all errors and diagnostics generated at the proxy machine to be easily relayed back to the customer's main-server. The restriction is that each JVM is dedicated to a single customer. The ProxyletLogger is the default implementation of these logging streams.

The implementation of the customer information as a Java class allows a large degree of flexibility in supporting the different types of proxy application environments in an efficient manner. Some customers may rely on simple substitution of URL fields to determine the programs that execute for a given URL, and their class definition can efficiently support this. Other customers may want to have these definitions be controlled by a configuration file at the main-server, and their class implementation can consult the main-server before making these decisions.

The main-server is a standard web-server which is typically augmented by two java servlets, one implementing the receiving end of the standard-output from the proxy-servers and the other being the receiving end of the error-output from the proxy-servers. These messages are simply logged to different files, with each proxy-server having a separate file-log. The main-server must allow the proxy-servers to download the proxylet application programs that are resident at its site. The main-server contains the instance of the MyApplication class as shown in Figure 3.

In our implementation, the main server was augmented with a web-module that provided the optional function of client redirection. The module worked off of a configuration file that contained a routing table indexed by two fields, a client subnet address and a requested URL. The table contained a third field which held the redirection URL. On each request, the IP address of the client was determined, and the routing table was consulted. If an entry matching the IP address of the client and the URL being invoked was found in the routing table, the client was redirected to a proxy server on the basis of the redirection URL. Otherwise, the request was served locally. Thus, the wide area load balancing capability was incorporated within the main server.

Applications that are developed on the main server need to explicitly identify the portions of the program that can run on the proxy servers. By default, we assume that java servelets created for the web-server can execute only on the main-server. However, the programs that are capable for executing on the proxy-servers implement a special java interface, the Proxylet interface, which allows a proxy-server to download the program and to execute it as needed. The proxylet interface allows for functions that can be performed when the code is initially downloaded at the proxy-server, as well as allow special routines to be invoked before and after every invocation in order to track the transaction performance.

The control-server was a simple web-server with a single servlet that responded to the identification requests from the proxy-servers within the domain. The servlet contained a configuration table which mapped IP address of proxy-servers to the customers being supported, and then provided the appropriate customer class definition to the proxy server. The class MyCustomer shown in Figure 3 is stored and retrieved from the control-server.

The execution environment in our implementation consisted of Apache web-servers using Tomcat as the servlet engine.

6. Personalization Application on Proxy Servers

In order to validate the effectiveness of the application architecture described above, we are developing sample applications that can be readily accelerated in the AAA environment. One of these applications is a personalization application which offers clients the ability to customize the web-page according to their interests and needs. The structure of a web-page presented by this instance of the personalization page is as shown in Figure 4. The web-page consists of two advertisement banners which run across the top and bottom of the web-page. The advertisements are generated on the basis of the preferences stored in a profile specific to each user. The goal of this application is to mimic the behaviour of customized portal sites. The main page consists of two panels, one showing the current news events that are of interest to the user, and the other panel shows the current prices of specific stocks that a user is interested in. While the structure of the page is simpler than most commercial portals available on the Internet today, it is adequate to demonstrate the ability of the application architecture to support such applications.

The contents of the pages are designed according to the user profile. In order to create the user

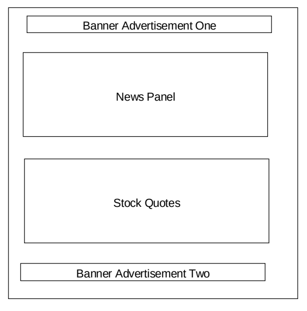

Figure 4

profiles, the application follows the model that each users need to register with the website and create a profile. When the profile is created the user selects the news categories that he/she is interested in, and also selects the set of stock market symbols that he/she is interested in getting a quote for.

The pages are created by means of Java proxylets which are developed and stored at the main server. They creation of the pages requires access to a user's profile. The user's profile is maintained as a master-copy on the main-server and cached as needed at a proxy server. In addition to the cache of user profiles, the proxy server also maintains three other types of caches. The first cache contains of an advertisement cache. The proxy server always selects an advertisement to be demonstrated on the top and bottom banner of the web-page from the advertisement cache. The advertisement cache is refreshed with a random set of advertisements from the main server at periodic intervals. The second cache consists of the cache of current stock prices. This cache is kept up-to-date by means of a simple push system running at the main-server which pushes stock updates out to the various proxy-servers using a publish-subscribe system. The third cache consists of a cache of current news items. News items are cached in various categories. The news in each category are refreshed from the main server on a periodic basis.

The page creation and assembly software runs like a servlet in the proxy environment. It composes the pages on the basis of the cached user profile. In order to access and modify their profiles, users need to log in using a password. The authentication of the users is not done by the proxy servers, but any information provided is simply relayed to the backend server which does the requisite checks to decide if the user should be granted access. Any modification to a user profile is written back to the main server immediately. When a user successfully logs on, the proxy server checks if its current copy of the cached user profile has changed, and if so, loads the new copy successfully.

The personalization application shows how a common web-based application can be developed for efficient deployment in the AAA environment. Experiments conducted within our laboratory setup indicate that application performance can be increased significantly by designing them for distributability. Preliminary performance studies have shown reduction in user response time ranging from 30%-50% from the original response time depending on network environment and application characteristics.

7. Alternative Approaches

The goal of the AAA environment is to enable applications to execute closer to their respective clients. There are alternative approaches that can be used for the same purpose. The most viable alternative would be to create the proxylet and modified script, and push them out to all the ASP nodes by means of content replication software. In conjunction with either of the common wide area load balancing techniques, the content replication model would provide a workable alternative to the proxylet approach.

The advantage of the content replication approach is that all of the content at an ASP node is statically populated. Therefore, it can be used in conjunction with standard data-caching. However, the content replication model would become less attractive as the number of ASP nodes becomes large. The overhead required in synchronizing a large number of ASP nodes with the main server content would become larger as the number of replicated sites grows.

Another disadvantage of the content replication model would be that all of the programs in the server node would need to be replicated at the ASP nodes, regardless of whether they are actually used. If there is a resource constraint at an ASP node, this would not be an acceptable situation. The proxylet model pulls the relevant pieces of software as they are demanded, and therefore results in a better utilization of the resources at the ASP node.

Another advantage of the proxylet model is that the code from different servers can be run at a single ASP site by exploiting the security features available with downloadable java code. These include the security/isolation features associated with Java security managers. In addition, the server administrator has to worry about application development, while the ASP node provider only has to worry about providing the appropriate distributed infrastructure. In a content distribution model, the ASP has to worry about providing adequate pemissions/protections to the various server administrators for them to upload their code to the ASP nodes.

One disadvantage of the proxylet approach is the fact that there is a penalty associated with proxylet usage every time there is a “cache-miss” of a proxylet at an ASP node. Whenever the proxy server does not have the right software or data cached at the ASP node, a flow to the main server would be required. This would increase the user latency on each such miss.

8. Further Research

This work was done as part of a continuing investigation of basic questions in application distribution for web-based services. We envision future computing environments where application components are dynamically distributed to appropriate servers within the networking infrastructure based upon available resources, end user characteristics, and traffic patterns.

System behavior will be modifiable via policies at both the application and networking layers, and services will adapt to changing conditions in order to maintain prescribed quality measures.

Fully dynamic exploitation of distributed computing environments requires an understanding of system state, resource characteristics, and application behavior. The effects of application distribution will depend upon how the components of the application interact. Traffic to the back-end servers should be minimized, but it is a function of end user requests. There are thus various issues in system monitoring, load balancing, and resource allocation that need to be addressed for large scale, Web applications.

There are also basic issues in how applications/services should be structured for distributed environments, and what tools/guidelines can be provided to programmers/composers of web applications to better take advantage of an intelligent network infrastructure. This will require that applications externalize information regarding their structure, and tools need to be provided to gather and maintain appropriate configuration information. There should also be controls that applications externalize to allow intelligent networks to better adapt their behaviors to the characteristics of the shared, distributed system.

A key question is that of determining the best allocation of the component classes of an object-oriented application to computing resources within the network. We plan to investigate the effectiveness of alternative algorithms for doing this assignment.

9. References

[ACTIVE] D. Tennenhouse and D. Wetheral, "*Towards an Active Network Architecture*", Computer Communications Review, vol. 26, no. 2. April 1996.

[ACTIVE2] K. Psounis. *Active networks: Applications*, security, safety, and architectures. IEEE Communications Surveys, 2(1), First Quarter 1999.

[ALANPROG] *The Alan Programing Manual*, available online at "http://dmir.socs.uts.edu.au/projects/alan/prog.html"

[ALANREF] M. Fry and A. Ghosh, "*Application Level Active Networking*", Fourth International Workshop on High performance Protocol Architectures (HIPPARCH 98), June 98.

[BIBLIO] Caching Bibliography available at http://www.web-caching.com/biblio.html

[CACHE] P. Rodriguez and E. W. Biersack, "*Bringing the Web to the Network Edge: Large Caches and Satellite Distribution*", MONET. Special issue on Satellite-based information services, January 2000.

[CARTER] R. L. Carter and M. E. Crovella. *Dynamic server selection using bandwidth probing in wide-area networks*. Technical Report BU-CS-96-007, Computer Science Department, Boston University, March 1996.

[DIFF] S. Blake et. Al. *An Architecture for Differentiated Services*, Internet RFC 2475, Decemember 1998.

[FOOTPRINT] Sandpiper Networks Inc., "*Footprint adaptive content distribution service*, " www.sandpiper.net.

[FREEFLOW] Akamai Technologies Inc., "*FreeFlow content distribution service*," www.akamai.com.

[GOVIDAN] Ramesh Govindan and Hongsuda Tangmunarunkit. Heuristics for Internet Map Discovery. In Proceedings of the 2000 IEEE INFOCOM Conference, Tel Aviv, Israel, March 2000.

[ICP] Wessels, D. and K. Claffy, *Internet Cache Protocol (ICP), Version 2,* RFC 2186, September 1997, [IDMAPS] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y. Jin. *An architecture for a global internet host distance estimation service*. In Proc. of IEEE INFOCOM 99, New York, Mar. 1999.

[LI] Z. Li and P. Yew. *Efficient interprocedural analysis for program restructuring for parallel programs*. In Proceedings of the ACM SIGPLAN Symposium on Parallel Programming: Experience with Applications, Languages, and Systems (PPEALS), New Haven, CT, July 1988.

[OBRACZKA] K. Obraczka and F. Silva. *Looking at network latency for server proximity*. Technical Report 99-714, USC/Information Science Institute, 1999.

[OEPS] G. Tomlison et. al, *Extensible Proxy Services Framework*, Internet Draft draft-tomlinson-epsfw-00.txt, July 2000.

[RSVP] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, *ReSerVation Protocol (RSVP) Version 1 Functional Specification.* RFC2205, Sept. 1997.

[SAYAL] P. S. M. Sayal and P. Vingralek. *Selection algorithms for replicated web servers*. In The 1998 SIGMETRICS/Performance Workshop on Internet Server Performance, June 1998.

[SCHWARTZ] J. Guyton and M. Schwartz. *Locating nearby copies of replicated internet servers*. In Proceedings of ACM SIGCOMM'95, pages 288-298, 1995.

[SPAND] S. Seshan, M. Stemm, and R. Katz, *Spand: Shared passive network performance discovery*, in Proceedings of the USENIX Symposium on Internet Technologies and Systems, December 1997.

[WANG] Jia Wang. *A survey of web caching schemes for the Internet*. ACM Computer Communication Review, 29(5):36-46, October 1999.

[WALS] B. Cain et. Al., *Known CDN Request-Routing Mechanisms*, Internet Draft draft-cain-cdnp-known-request-routing-00.txt, November 2000.