Proxy-based services in Distributed Systems

Gaurav Pancholi
Computer Science and Engineering,
The University of Texas at Arlington,
pancholi@cse.uta.edu

April 4, 2002.

Abstract

In today's vast and heterogeneous environment, new technologies and services are emerging at an extremely rapid pace. Many of these cutting-edge technologies cannot be easily incorporated into the present network infrastructure. Their deployment requires significant changes in the current infrastructure, which may not be feasible, because of the scale of infrastructure in place. Eventually, the existing network protocols will evolve to support most of the emerging technologies; yet it will be almost impossible to keep up with the continually changing applications and hardware interfaces. The paper discusses a very efficient, cost-effective and flexible alternative to solving the above issues, with the use of proxy-based services. These services can be used to improve the functionality and performance, regardless of the underlying network. They can be applied to the existing network infrastructure, to improve the quality and reliability of the existing services.

Keywords: proxy-based services, performance enhancing proxies, tacc architecture, proxy services, adaptive proxies.

1. Introduction

Today's Internet infrastructure comprises of uncountable clients and servers. They vary along many axes like screen size, resolution, processing power, etc. These devices are continuously evolving, along with that new smart applications are being introduced. These conditions make it extremely difficult for the server to provide a service that will be relevant to the all types of clients. Traditionally, a web client sends a HTTP [11] request to a server, the server responds back to the client. The client and server assume that the data will reach the other end unchanged. There is no effort on the part of the server to take into consideration the type of client, or to tailor the data according to the client specifications [8].

If a proxy-based service exists in the above-discussed scenario, the client-server communication will be tunneled through the proxy residing between the client and the server. The existence of such a proxy will be transparent to the end client and the server. The proxy in the network would funnel the client request and distill it based on what work it is intended to do. For example, a caching proxy would look for the page within the cache before sending the request further; a load balancing proxy would send the request to a mirror site if the target server is over-loaded. In both the above cases, the request may not reach the target server, but the client gets what it had requested [13]. Proxy based services should be able to perform adaptation based on network variation, hardware variation and software variation.

The paper discusses performance enhancing proxies and the architecture used to deploy the proxies. Section 2 discusses the proxy-based services in general, Section 3 deals with the TACC Architecture, a widely accepted architecture used to develop proxy services, Section 4 explains the SNS model to deploy proxy-based services, Section 5 explores the areas of application and Section 6 has discussion and conclusion.

2. Proxy-based services

Proxy-based services are emerging as an alternative for adaptation in a heterogeneous environment. The proxy-based approach has a clear advantage over the *client-based adaptation approach*, where the clients are brought upto the minimum level of functionality to cater to the ever-changing environment, and the *server-based approach*, which deals with updating the server with all the functionality required for adaptation [6].

2.1 Advantages of proxy-based services

The advantages of proxy-based services over the other two are:

- The existing Internet has a vast infrastructure in place. The solution for solving client and server heterogeneity should not discard the existing servers and clients. Moreover, it should support incremental development and upgradation to digest the needs of the future. Proxy based approach succeeds in achieving both the above objectives. It sits between the client and the server, thus requiring none or minimum changes to the client or the server. The architecture for the proxy services is chosen such that it can be easily expanded.

- Proxy-based approach enables new features and behaviors for the client, many of which maybe impossible without proxy support. By transferring the complex subsystems of the application on the proxy, those applications can be supported on simple clients. For example, the Top Gun Wingman web browser [4] for 3Com Palm-Pilot [1].

- Incremental deployment of the services is much more cost-effective and easy in a proxy-based environment than in a traditional large-scale network.

2.2 Design Issues

Critical issues concerning the design of proxy-based services are:

- **Scalability**: The first necessary requirement is scalability. As load increases, the system should be able to scale easily by addition of hardware or software. The service should also be able to handle sudden upward surges in load [15].

- **Fault Tolerance**: The service must be available to the end-users all the time, inspite of system crashes, overload or any hardware change. Any fault within the service should not break down the whole system. The system should be able to initiate recovery procedures in case of any fault [15].

- **Cost Effective**: The design should be economical to administer and expand.

- **Adaptation**: The service should be able to adapt to different types of clients. The most common approach used for meeting the client adaptation is to use data specific compression [3]. Data specific compression makes intelligent decision about the information to present to the client and information that can be discarded, based on the type of client and the type of data. For example, compressing an image would deal with reducing the color information or pixel resolution. Also the client can be shown a thumbnail of the image; the full-scale image is shown only on client request. Text compression deals with reducing some formatting information, removing bold and italics if the client cannot handle it. In all cases, there is an attempt to present the information in its best semantic value. This process is referred as *distillation* [3].

2.3 Example Distiller

GifMunch is an image distiller developed at UC Berkeley [10]. It performs compression on GIF [9] images. The distiller reduces the resolution of the image according to the client. The level of distillation applied on the image reduces the end-to-end latency of image retrieval. The image viewed by the client would be of low clarity. The client can zoom on a portion of the image, and that portion can be viewed in original resolution. Also the software can reduce the color encoding information or transform the image into a gray scale, which would be appropriate for small-scale devices like PDA's. Some devices like the 3COM PalmPilot [1] use their own proprietary image encoding formats. The software has ability to convert the GIF image into such proprietary formats, hence reducing space and latency.

3. TACC – A programming model for proxy-based services [2]

TACC is a programming model for development of proxy-based services. TACC stands for *Transformation*, *Aggregation*, *Caching* and *Customization.* *Transformation* stands for changing the content of a data object; for example encrypting user data, or encoding\decoding data. *Aggregation* means collecting data from different sources and using it in a way specified by the user. *Caching* is having frequently accesses\used data stored locally. Caching in proxy-based systems also deals with storing the data after transformation locally. Customization is the core of proxy-based services. The system maintains a database of user preference that allows it to deliver the output based on user preferences.

In the TACC architecture, applications consist of building blocks or '*workers*'. Each building block is an expert in a particular task; the building blocks are inter-connected by a set of API's. Each worker specializes in a particular *task*. In the Internet scenario discussed in section 1, there can be a worker for image distillation, a separate worker for caching pages etc. The applications are formed by aggregation of workers, the workers can inter-communicate [3]. A worker can call a routine of another worker. The resulting application model is very general and simple for proxy-based application development. An important feature of the TACC architecture is that the workers get the user-profile information along with the input data from the client. The advantage of this is that the same worker can be re-used several times. For example, a worker dealing with image-compression can reduce the image resolution given a set of parameters, and the same worker can be used to reduce the size by reducing the color-level given a separate set of parameters.

Workers can be of two types [2]:

- Passive workers receive their task from some other component of the system, they compute the desired task and send back the result.

- Active workers are the ones that interact with the outside world. This category of workers get request from the outside world. They decide on the steps to be taken, whether to respond back directly to the sender, or to direct the request to some other worker.

In a web-based service, the active workers would listen for any HTTP [11] requests from the client, and forward the requests to other workers in the system. It sends the response from the internal worker back to the client. The back end passive workers are the ones that do the actual computation, and the passive workers rely on them for getting the work done [2].

Workers have certain characteristics, which help in simplifying the design of the underlying architecture.

- There are different classes of workers. The workers performing similar tasks are clubbed together in same class. All the workers belonging to same class are considered similar and interchangeable.

- Workers accept tasks from a simple serial interface. The originating worker sends a task to the consuming worker by specifying the class of the worker, and the input data.

- The tasks are considered to be atomic and restartable. This is achieved my making the workers stateless. Hence the system remains consistent even if one task is executed multiple times or a worker faults during execution.

Many different services can be developed using the TACC programming model. The next section discusses a widely accepted architecture for the TACC programming model.

4. Cluster Based TACC Architecture [2]

A cluster-based architecture is most widely accepted for proxy-based services. Clustered architecture has significant advantages and fits well for proxy application development.

4.1 Advantages of Clustered Architecture.

Clustered architecture provides three significant advantages over single large servers [2].

1. **Scalability**: Clusters are well suited for proxy application, where multiple workers perform computation in parallel. Typical computation time consists of a few CPU seconds. Clusters provide better performance in such environment. In addition, clusters have an ability to grow incrementally with time. This characteristic is particularly important in proxy-based application where planning the future capacity is impractical and highly prone to errors.

2. **Availability**: Availability time in clusters is much higher due to independence of the nodes. Each node has its own set of resources and hence the system can easily recover from faults. Also, a subset of nodes within a cluster can be disabled in place and upgraded. This will impair the system performance temporarily but is very essential when round the clock uptime is expected and system should overcome hardware and software failures inspite of the faults.

3. **Commodity blocks** [15]: The cluster architecture comprises of small building blocks. These blocks provide a cost/performance advantage; the memory, data storage and other components of the blocks follow the leading edge in technology. Also it is easy to obtain, configure and maintain the independent nodes, than in the case of a large centralized server.

The above advantages of cluster architecture go extremely well with the design issues concerning the proxy-based application discussed in previous section.

4.2 SNS Architecture

Designing and managing a system that is distributed across multiple clusters poses numerous challenges. This section discusses the actual architecture for building such cluster-based systems, called "*Scalable Network Services*" or **SNS.** Application developers build their application on top of SNS. The SNS architecture relieves the application developer from the details of the underlying service management, thus allowing him to concentrate on the semantics of the application [5].

In the simplest form a worker can be independent, which means that it can be executed on any node of the system. On the other hand, some workers can have an affinity with a particular resource in the system and hence can be executed only on some special nodes.

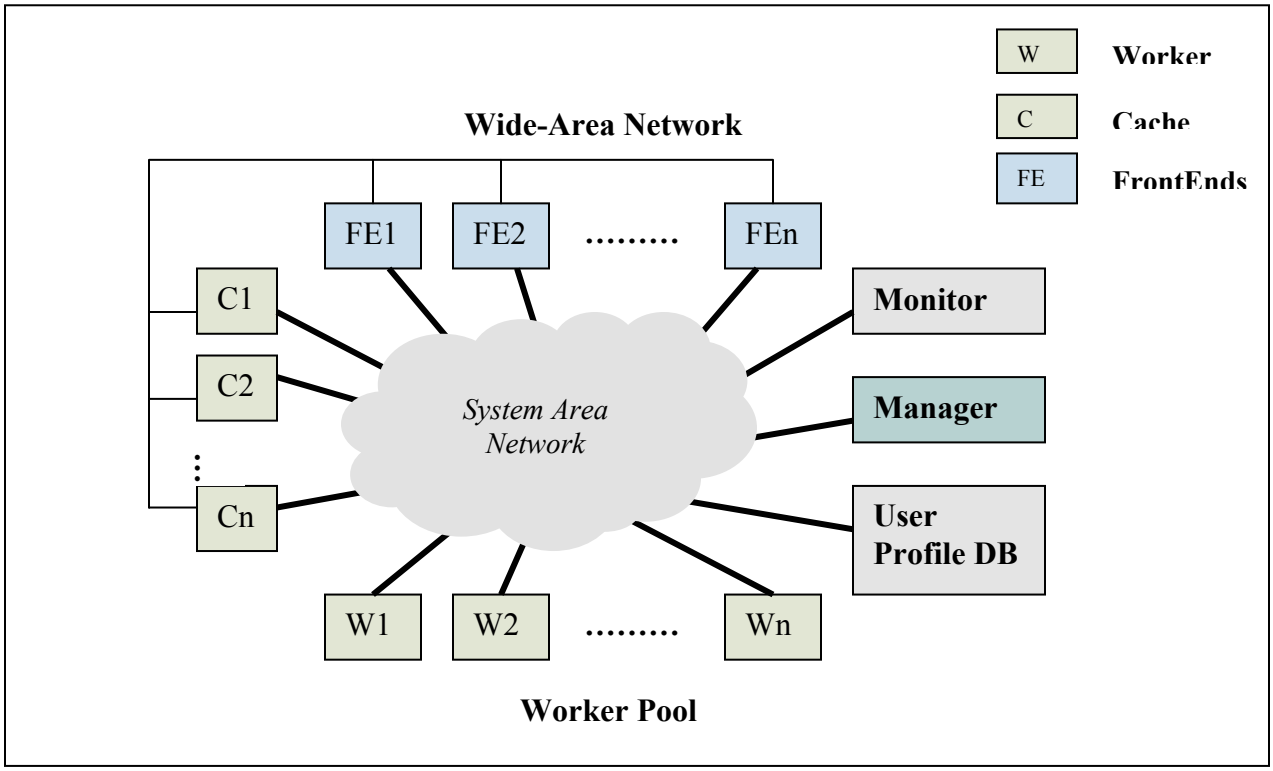

Figure 1: Block Diagram of SNS Architecture

The block diagram on a generic SNS is shown above. The workstations in the diagram can support one or more software component. The workstation can also be a network of workstations (NOW) [14][16]. The components of the block diagram are:

- **FrontEnds**: FrontEnds are the interface of the SNS with the outside world. The FrontEnd components listen to the request from the end-users. They match the incoming request with the appropriate user profile from the User Profile DB and decide on which passive worker should serve the request. They channel the request to the particular passive worker within the system.

They can be replicated for scalability and availability. FrontEnds maintain state for many simultaneous request [14].

- **Worker Pool**: The worker pool comprises of the actual service components and caches. The number of service components in the pool depends on the load on the system.

- **User Profile DB**: This is the database that stores the profiles of different users. This profile is used while servicing the request and for adaptation.

- **Manager**: The manager is responsible for all the groundwork. It does the load balancing across the system, instantiates new workers based on the offered load. In case of sudden bursts in load, the manager activates the *overflow pool.* The overflow pool is a set of back-up machines, which are used in cases of sudden increase in workload [5].

- **Graphical Monitor**: Graphical Monitor is used to visualize the system behavior and notification in case of errors\faults in the system.

- **System-Area Network**: It is a very low-latency and high bandwidth network. The important consideration is that the interconnect network should not become a bottleneck as the system scales.

4.3 SNS Manager

The SNS Manager is a central server. If we can guarantee that the Manager is fault tolerant and is not a performance bottleneck, we can design various load balancing and resource location policies. The main duties of the manager consist of resource location, load balancing and handling sudden bursts in network traffic.

4.3.1 Resource Location

The SNS Manager acts as a name service for the rest of the system. The components of the system contact the Manager to locate the other resources. The SNS Manager has a table of all available resources\components within the system. When a worker contacts the Manager for locating a particular resource, the manager performs a look-up in the table and returns the location of the resource back to the worker. Workers cache this information and avoid contacting the manager again for locating the same resource again.

4.3.2 Load Balancing

The Manager handles the load across the system. Load gathering mechanism is centralized at the Manager. Each worker periodically sends its local load report to the Manager. Based on these local load reports, the Manager generates a load-balancing plan, which is sent to all the workers. Workers use this plan to make local decision regarding which group of workers to communicate for a task.

4.3.3 Handling Sudden Bursts

The SNS Manager is responsible for instantiating new nodes and starting new workers as the load on the system grows. In addition, the Manager has to take care that the system performs well with burst traffic. Sudden bursts in traffic load can bring down the system. The Manager should deal with such traffic by replicating workers at idle nodes. The Manager can also bring in required nodes from the "*Overflow*

Pool". Overflow Pool is a group of nodes which are set aside and are used only in case of emergency, like bursty traffic. The nodes in Overflow Pool are normally used for some other purposes. When the Manager runs out of dedicated nodes, it uses the nodes from the Overflow Pool.

4.4 Resource Location

The resource location in an SNS architecture deals with three basic issues:[5]

- Locating the service,
- Locating the Manager,
- How the workers locate each other.

4.4.1 Locating the Service

For the end-users to use the service, they should be able to locate the service through some well-defined naming mechanisms. The application should have some location component, which is used to communicate with the rest of the world. Web-based services can use the Uniform Resource Locators as a service location mechanism. Some of the other techniques that can be used are DNS round-robin [16] and IP Multicast. [16]

4.4.2 Locating the Manager

Manager in SNS architecture is centralized. Multicasting is used to locate the SNS Manager within the system. The manager sends a message to all other components in the system at regular intervals indicating its existence. All the other components listen to the messages from the manager. If a manager crashes and comes up at a different location, it indicates the new location to the other components through the multicast. The components get the new location and redirect their communication.

4.4.3 Locating Workers

Workers are grouped into classes based on their semantics and functionality. Each class of workers is given a unique name. Names are assigned such that they indicate the service provided by the class. Each worker registers itself with the SNS Manager, along with its class name. The SNS Manager uses this information to build the table of active components in the system.

4.5 Launching of Workers

When a task needs to locate some other worker in the system, it sends a *Find* message to the Manager. The message also contains the name of the class the worker belongs to. If the manager cannot locate the worker in its table of active components, it needs to start a new worker. The manager also maintains a table of all workers in the system, called the *Launch Table*. The manager looks up the launch table and finds a worker on the lightly loaded node and makes an attempt to start the new worker at the selected node. There may be instances when the server cannot spawn a new worker, in that case it responds to the requesting worker with a *NotFound* message.

4.6 Load Balancing

A good system design should be able to distribute load uniformly across the system. Load balancing in SNS architecture deals with three basic issues: 1) Load Measurement, 2) Communication, 3) Resource Allocation. SNS Manager is responsible for load balancing.

4.6.1 Load Measurement

Load can be measured in terms of several system parameters, CPU usage being the most common. The TACC model allows the application developer to choose from several load measurement parameters. When a task arrives at the worker, the worker performs a brief analysis on the task and makes a rough estimation of the load that the task would generate. Each worker has a measurement of its current load. On task arrival or completion of a task the load measurement is updated. The worker sends its load measurement to the SNS Manager at regular intervals. The time interval is configurable.

4.6.2 Balancing the load

The SNS Manager has the load information of the system. As discussed before, the SNS Manager multicasts a message to all the components indicating its existence. The Manager also piggybacks the load information with this message. The workers can use this centralized load information to make local load balancing decisions.

Another important aspect of load balancing is the ability of the Manager to launch new workers as the offered load increases. The SNS Manager uses the load reports from the workers; if load on a particular worker is above the acceptable limit, the manager launches a worker of the same class and transfers some tasks to the new worker. After launching of new workers, it may take some time for the load to balance. In some cases, there maybe false signals of overload from the previously overload worker. Hence the manager disables the load balancing mechanism for that particular class of workers [5].

4.7 Fault Tolerance

The issues critical to fault tolerance are detecting the occurrence of fault and recovering from the fault.

4.7.1 StarFish Tolerance [5]

Most conventional systems have a primary/secondary fault tolerance mechanism. Each component is the system has its secondary copy, which is used in event of the failure of the primary. Hence, each component has its replicated copy. In cluster-based model with multiple independent nodes, replication is a very expensive approach. The SNS architecture uses a StarFish tolerance approach. The mechanism is named after *star fish*, which can regenerate all its arms as long as it has one intact. In SNS architecture each component monitors the other components and ensure that they are available. The SNS Manager is the centralized actor, and it monitors the whole system. In event of any component crashing, the manager restarts it. The workers are responsible for restarting the manager.

4.7.2 Worker Failure

The SNS Manager uses the periodic load reports from the worker to detect worker failure. If the manager does not hear from a worker for certain interval of time, it assumes that the worker has a fault, and notifies all the other components of the system about the fault. If there were some tasks under execution at the worker, they are lost. Since the tasks are inherently restartable, the task migrates to a new worker. Fault of a passive worker does not need immediate attention by the manager; a new worker will be generated only when required. For active workers, immediate restarting mechanism is necessary. Such workers when they initially register with the manager set a “restart-on-death” flag. The manager checks the flag and knows that the worker should be restarted immediately.

4.7.3 SNS Manager Failure

The centralized manager can be a single point of failure; hence the system must have mechanism to overcome a manager failure. All the workers in the system are responsible for restarting the manager. When the workers do not get an existence message from the manager, a distributed restarting algorithm starts [2]. Each worker starts a random timer. The worker whose timer expires first restarts the manager. Before restarting the manager, the worker sends "amResending" message to all the other workers, the other workers on receiving the message remove their local timers. Even then, it can happen that two workers restart two different managers. In cases like that, one of the manager commits suicide. Each manager sends a random number in the existence message it multicasts to the other components. When a manager gets an existence message from another manager, it compares the number in the beacon with its own and if its number is smaller, it has to die.

During the time when the manager was inactive, rest of the components in the system can still perform. The components can communicate with each other using their local cached information. Load balancing is hampered, as in absence of the manager new workers cannot be spawned. Also the system uses *soft-state* updates, which ensure that recovery from the crash is easy. When the manager restarts, the components gets new existence multi-cast message, and hence they re-register with the manager. The manager gets load information from the workers and re-builds its load database.

4.7.4 Timeouts

Timeouts are used for fault management throughout the system. When a worker sends a task to another worker, it waits for a fixed amount of time to get the response. If it does not listen from the other worker, it sends the task to another worker. Similarly, if the SNS manager does not listen from a worker for a fixed amount of time, it assumes the worker to be dead.

5. Areas of application

Proxy-based systems have been implemented in different areas. This section lists some of them:

1. **OREO transducer** [8] – This is an application for WWW browsers and servers. The application has a browser side wafer, a server side wafer and a proxy filling in between the wafers. Some of the uses of OREO's are for checking malformed URL's and report error back to the client, measuring date traffic, text indexing and retrieval etc.

2. **Caching proxies** – These proxies perform extensive caching on behalf of the end users

3. **Load balancing proxies** – Proxy-based services can be applied for load balancing at the server. When a request is send to a server, the proxy checks the load and directs to a replicated server is the main server is overloaded.

4. **TranSend** [7]: TranSend is a web distillation proxy developed at UC Berkeley. It performs on the fly transformation of the web documents as per the clients.

5. **Wingman** [4]: Wingman is a proxy-based web browser for 3Com PalmPilot [1]. Most of the functionality is moved to the proxy and the client is very minimal. The proxy does web caching; transforming HTML into a proprietary format, image converters etc.

6. **VSAT networks** [12] – VSAT networks are implemented with geosynchronous satellites. There is a large hub earth station and all the terminals communicate through the hub. Proxies are used to improve the throughput and reduce response time.

6. Discussion and Conclusion:

Currently, a lot of research in proxy-based services is focused on improving the performance of existing network protocols, particularly for TCP\IP. It has been proven that the performance of TCP is exceptional in networks that provide fixed bandwidths and minimal delay. Researchers are trying to study the behavior of TCP in networks that do not provide favorable parameters to TCP, and are trying to develop proxy-services that can augment TCP and improve its performance. One of the researches suggests deploying proxies at the edge router. The proxies estimate the bandwidth available in the link. If the proxies can estimate that the request by the sender will reach the destination, they send a premature ACK to the sender, which is similar to the one that the destination would have send. In addition, proxies to improve congestion-control mechanism in TCP are under research. It has been observed that the end-points do not detect congestion in a TCP network fast. Researchers are trying to solve this problem with the use of active proxies. A variety of applications have been suggested for different layers of TCP. Another area of research is improving the security mechanism in IP. An approach suggested is using multi-layer IP security. The drawback of this approach is significant amount of complexity in implementation.

Use of proxies in Wireless WAN's is also an area of current research. Researchers are trying to develop proxies that will improve the response time and throughput in a Wireless WAN. The Mowgli system [17] is an approach that addresses the low bandwidth issues in Wireless WAN's. An application layer proxy is inserted between the client and the server. The proxy makes effort to improve the response time in Web browsing. A lot of work is also directed towards the Wireless Application Protocol (WAP). The WAP architecture has a proxy between the WAP client and WAP server. The client sends a request to the proxy. The proxy does the work of converting the client request into a HTTP request and submits the request to the server. The server sends the response to the proxy that again goes the encoding and directs it to the initiating client. One particular research suggests the use of proxies in the existing JAVA RMI specification, to adapt it for wireless networks. The existing specification is not suited for work in wireless network architecture.

A critical issue in most of the proxy-based services research is where the proxies should be deployed so that they can have optimal impact on the architecture. Some of the factors to consider in making the decision involve the information the proxy require, the latency in obtaining the information and the extra traffic the proxy would generate.

There are a group of researchers, who argue that the use of proxies is not desirable, the main reason being that they break the end-to-end TCP\IP reliability and security. The argument is that the existing protocols will eventually evolve to handle the client-server adaptation, and the proxies will become obsolete.

Proxy based services have a significant influence on the performance and efficiency of heterogeneous and ever evolving network environment. These services will be deployed in numerous different areas. Currently, a lot of research is directed in the field of proxy-based services, main focus of attention being: 1) improving the architecture on which such services will be deployed, and 2) exploring different areas in which such services will lead to significant improvement in terms of quality of service and the end-user experience as a whole. Proxy based services are not limited to the scope discussed in this paper. They can be applied to any area where they would be beneficial and cost-effective.

On the contrary, proxy-based services should only be used in specific environments and circumstances where end-to-end mechanisms for performance improvements are not available or are not feasible. Efforts to improve end-to-end performance should be the primary goal. Also environments where proxy services are deployed, the end-user should be aware of the presence of such services, and the choice of employing such services should be under the control of the end-user if possible. This is important, especially if employing any middleware service would interfere with the security mechanism in place at the client end or would have any other undesirable implications.

7. References

[1] 3Com Corporation, 3 Com PalmPilot – http://www.3com.com/palm/index.htm

[2] A. Fox, "*The Case for TACC: Scalable Servers for Transformation, Aggregation, Caching and Customization*", UC Berkeley Computer Science Division, April 1997

[3] A. Fox, S Gribble, Y. Chawathe, E. Brewer, E Amir, "*Adapting to network and client variability via On-demand Dynamic Distillation*", In Proceeding of ASPLOS-VII, Oct. 1996

[4] A. Fox, I. Goldberg, S. Gribble, A. Polito, D. Lee, E. Brewer, "*Experience with Top Gun Wingman, A Proxy-Based Graphical Web Browser for the 3COM PalmPilot*", March. 1998.

[5] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, P. Gauthier, "*Cluster-based scalable network services*", In Proceeding of SOSP '97.

[6] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, P. Gauthier, "*Adapting to Network and Client Variations using Active Proxies: Lessons and Perspectives*", Feb 1998

[7] Armando Fox et al., "*TransSend web accelerator proxy*". UC Berkeley.

[8] Charles Brooks, Murray S. Mazer, Scott Meeks, Jim Miller, "*Application specific proxy servers as http stream transducers*", In Proceedings of the Fourth International World Wide Web Conference, Dec 1995.

[9] Graphics Interchange Format (GIF) . CompuServe Incorporated, Columbus, Ohio.

[10] GifMunch – A GIF Image Distiller, UC Berkeley.

[11] Internet Engineering Task Force, Hyper Text Transfer Protocol – HTTP 1.1, Mar 1997. RFC 2068

[12] J. Border et. al., "*Performance Enhancing Proxies*", RFC 3135, http://www.ietf.org/rfc/rfc3135.txt, July 2000.

[13] Rob Barret, Paul P. Maglio, Daniel C. Kellem, "*How to personalize the Web*", 1996. WBI, developed at IBM; http://www.raleigh.ibm.com/wbi/wbisoft.htm

[14] T. E. Anderson, D. E. Culler, D. A. Patterson, "*A case for Networks of Workstations: NOW*", August 1994.

[15] Y. Chawathe, E. Brewer, "*System Support for Scalable and Fault Tolerant Internet Services*", In Proceedings of Middleware '98, Sept. 1998

[16] Y Chawathe, S. Fink, S. McCanne, E. Brewer, "*A Proxy Architecture for Reliable Multicast in Heterogeneous Environments*", Feb 1998.

[17] Mika Liljeberg, Heikki Helin, Markku Kojo, Kimmo Raatikainen, "*Enhanced Services for World-Wide Web in Mobile WAN Environment"*, University of Helsinki, Finland.