SEcS: Scalable Edge-computing Services

Raffaella Grieco
Dipartimento di Informatica ed Applicazioni "R.M. Capocelli"
Università di Salerno
84081, Baronissi (Salerno), Italy
rafgri@dia.unisa.it

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

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

ABSTRACT

We present the architecture of a scalable and dynamic intermediary infrastructure for developing and deploying advanced Edge computing services, by using a cluster of heterogeneous machines. Our main goal is to address the challenges of the next-generation Internet services: scalability, high availability, fault-tolerance and robustness. Moreover, SEcS offers an easy, "on-the-fly" and per-user configuration of services. The architecture is based on IBM's Web Based Intermediaries (WBI) [8, 9].

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/server, distributed applications*

Keywords

World Wide Web, Edge Services, Intermediary Systems, Proxy Servers, Personalized and Mobile Services.

1. INTRODUCTION

With the exponential growth of the Web and its explosive success, more and more dynamic, multimedia and interactive services are increasingly offered to the end users. In particular, access to the Web comes from a wide range of heterogeneous devices, that connect to the Internet using a variety of existing and emerging IP wireless technologies ranging from Bluetooth to GPRS and 3G.

These trends have involved a growing demands for value-added proxy services infrastructures, that is, for network service infrastructures that must be able to efficiently provide intelligent services at the *Edge* of the network, as close as possible to the end users, that are, indeed, the ultimate judges of the quality of the services.

The trend in the research about advanced Internet services [1, 8, 15, 16, 17, 18, 19] is how to develop and deploy intermediary infrastructure, strategically distributed through the network, in order to allow content adaptation and other complex functionalities on the HTTP data flow exchanged between clients and servers. In this avenue, we can cite WBI [8, 9] of IBM as well as BARWAN [24] and Ninja Projects [20] by UC/Berkeley as architectures that place emphasis and provide tools to develop significant services on the Edge of the network. In this context, an important step toward standardization is being conducted by the IETF Working Group for "Open Pluggable Edge Services" (OPES) [6]. Among their results, it is defined [31] an architecture of composable Edge Services that simplify the efficient delivery of complex content and services.

Systems that provide Edge-computing Services (EcSs) must exhibit some important characteristics, such as, e.g., programmability, adaptability, as well as a compositional framework to dynamically build a pipeline of services according to a higher-level programming model. The services must also be easily personalizable and reconfigurable by each service user, by changing service parameters as well as assembling them in new services.

Many are the natural applications of EcSs: from the *geographical personalization* of the navigation of pages, with insertion or emphasis of content that can be related to user geographical location, to *translation services*; from support for *group navigation and awareness* for social navigation to advanced services for *bandwidth optimization* such as adaptive compression and format transcoding.

A platform for Scalable EcSs

Our proposal, named Scalable Edge-computing Services (SEcS), is based on IBM Web-Based Intermediaries (WBI) [8, 9] by adding *(1)* support for remote interaction of WBI components, *(2)* a communication infrastructure among remote WBI components and distributed dispatchers to balance dynamically the load on a cluster of workstations and *(3)* support for personalization and configuration of the services on a per-user basis.

2. SEcS ARCHITECTURE

An overview of WBI

SEcS is based on a framework, Web Based Intermediaries (WBI), developed at IBM Almaden Research Center in order to simplify the development of Web Intermediaries, i.e. applications that deal with HTTP information flows. In order to introduce SEcS, we, first, describe concisely WBI architecture and the terminology that is used to describe its components.

WBI is a programmable proxy written in Java, that has been used in various contexts [2, 10, 11, 21, 33] and its technology lies at the heart of parts of commercial products of IBM such as WebSphere Transcoding Publisher. WBI acts as an HTTP request processor, receiving a request and sending a response as customary for HTTP proxies. A typical WBI transaction flows through a combination of the four types of basic stages, called MEGs, that intercept and deal with HTTP requests and responses.

The WBI developer writes the application by writing several MEGs and assembling them in Plugins. WBI dynamically builds a

SAC'05 March 13-17, 2005, Santa Fe, New Mexico, USA

data path through the various MEGs for each transaction by using rules and priorities established by the programmer. When an HTTP request is received by WBI, it is sent through the chain of MEGs triggered by such rules, and finally, the response is returned back to the client through the same path.

An overview of SEcS
Our service architecture, shown in Fig. 1, consists of a local execution environment in which local services are applied on the HTTP request/response flow, and a remote execution environment that allows content requests and responses to be transformed or adapted by remote components (for services not locally available).

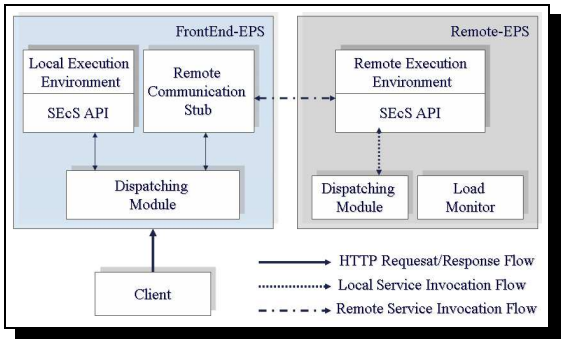

Figure 1: SEcS architecture overview.

SEcS architecture includes the following basic components: the Remote Edge Proxy-Server (Remote-EPS), the Front End Edge Proxy-Server (FrontEnd-EPS), and the SEcS Manager (as shown in Fig. 2).

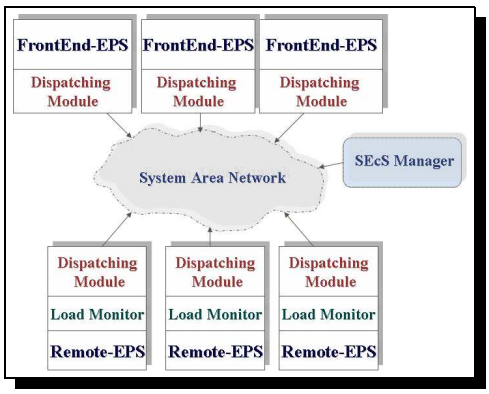

Figure 2: SEcS components.

FrontEnd-EPSs provide an interface that is accessible by the outside world, interacting directly with the browser for the execution of all the HTTP requests. The SEcS Manager allows to keep the internal status in terms of addresses (i.e. hosts) of available Edge Services, provided by the Remote-EPSs. Load information (to balance the workload) are sent with the responses to the remote invocations to each FrontEnd-EPS. In addition, it is necessary to spread evenly the requests among the FrontEnd-EPSs. Solutions to this problem had to rely exclusively on standard mechanisms (i.e. no modification of clients was allowed). Our solution is using a Javascript Client Autoconfiguration file that implements an hash function to evaluate the target FrontEnd-EPS to be used.

Because of efficiency, we used sockets as communication mechanism among Remote-EPSs and FrontEnd-EPSs but also provided the programmer with object marshalling and un-marshalling methods, that makes SEcS extremely easy to use for a WBIprogrammer. For efficiency, we have used the Java API `SocketChannel` since it provides non-Blocking I/O and direct access to memory buffer.

2.1 SEcS Components

The Client Autoconfiguration File
The Client Autoconfiguration File (CAF) is a JavaScript file that can be used by the most popular browsers to provide a proxy to be used for the HTTP connections [13, 29, 32]. Its main goal is to avoid a bottleneck on a single FrontEnd-EPS, by allowing a (transparent for end users) load balancing among the FrontEnd-EPSs of the system.

The Front End Edge Proxy Server
The Front End Edge Proxy Server (FrontEnd-EPS) is the component of the architecture that interacts directly with the browser for the execution of all the HTTP requests. At start-up, each FrontEnd-EPS registers itself with the SEcS Manager. When the client issues an HTTP request, the FrontEnd-EPS identifies the user (or provides to initiate a challenge-response authentication), and then loads the configuration of the services that the user selected during the configuration (stored in a User Profile Database).

Once identified the client, the services selected by the users are applied to the HTTP flow of requests/responses. For each service, the Dispatching module of the FrontEnd-EPS chooses, among the Remote-EPSs that offer the first MEG of the requested service, the one with the lowest load, and then it establishes the remote communication.

Finally, the FrontEnd-EPS takes care of the configuration and personalization of the services for each user, by intercepting a particular URL. The user can choose the services and configure the parameters (internal to each service) by a personalized homepage.

The Remote Edge Proxy Server
The Remote Edge Proxy Server (Remote-EPS) is in charge of providing the Edge services. It is a plugin of a WBI session that instantiates both remotely used MEGs as well as locally used (i.e. standard WBI) MEGs. An EcS can consists of a single MEG as well as many of them. When the Remote-EPS starts-up, it registers itself with the SEcS Manager by communicating details on the services it is offering to FrontEnd-EPS. The SEcS Manager broadcast a table of all the active services to all the FrontEnd-EPSs.

The Dispatching module in Remote-EPS is needed to choose the best Remote-EPS for the next MEG of the service while the Load Monitor collects the system state information.

The services offered by Remote-EPSs can be replicated on many nodes of the cluster in order to guarantee the required scalability, availability and fault-tolerance. The number of FrontEnd-EPS and Remote-EPS that are deployed on a cluster is left as a choice to the system manager, according to the resources and the requirements of availability, fault tolerance, scalability and computational load of the services to offer. Of course, one of the most natural choices (as in the experiments described later) is to deploy on each node of the cluster both a FronEnd-EPS and a Remote-EPS.

SEcS Manager
The SEcS Manager is a Java application that may be located on any machine in the cluster. Its main goal is to act as a name server for the rest of the system. It is invoked by FrontEnd-EPSs for their registration as dispatchers as well by Remote-EPSs for the registration of new Edge services.

It should be noticed that, while the Manager represents the only centralized point in the architecture, it is not subjected to substantial

workload since it is contacted by EPSs only at startup.

2.2 Load Balancing

In this section we describe the different load balancing algorithms that we have implemented in SEcS.

Round robin algorithm

The round robin algorithm does not consider any system state information. If S_i is the last server being invoked, the new request is assigned to $S_{(i+1) \bmod n}$ where n represents the total number of proxy servers.

Least Loaded algorithm

In order to keep the workload evenly distributed among the machines, it is necessary to provide a (programmable) mechanism to balance the load.

The load of each machine involved in providing Edge-computing Services is monitored by a thread (Load Monitor Module). The values are added into each request that passes through the MEGs running on that machine (piggy-backing). In this way, the FrontEnd-EPS takes the information from the requests as they are sent back to it, allowing scheduling decision based on the most recent results. As consequence no overhead is introduced in the network, in fact load information are not periodically exchanged.

Also we offer to the developer a tool to change the way the load is measured and how the best proxy is selected. The workload of each Remote-EPS is represented along different axes: the parameters are CPU load, free memory percentage, bandwidth, number of active network connections and number of wait connections. In this way we provide a complete view of the responsiveness of the Remote-EPSs. The programmability of the mechanism can be obtained by using provided methods that allow the programmer to define the relative weight of each workload parameter.

K2 Least Loaded algorithm

When balancing the load in a distributed system, it is well-known [14] that the strategy of sending each request the the least loaded machine can behave badly if load information is old. In such systems occur an "herd effect", that is, machines that appear to be underutilized quickly become overloaded because everyone sends their requests to those machines until new load information is propagated. To solve this problem, some system adopts randomized strategies that ignore load information or simply take into account only a subset of load information.

Solutions for load balancing with stale information are present in literature and we follow Mitzenmacher's work [26, 27]. In his settings, if there are n servers, instead of sending the request to the least loaded server, a client randomly select a subset of size k of the servers, and sends its request to the least loaded server from this subset. When $k = 1$ this is the same as sending the request to a randomly selected server, without taking into account any load information, while the case $k = n$ is like sending the request to the least loaded server. Since, in Mitzenmacher analysis [26, 27] it is shown the $k = 2$ version of the algorithm is a good choice in situations with small update intervals (like in our case), we have used this value for our implementation.

2.3 Structure of a Service in SEcS

SEcS's architecture supports the dynamic composition of services into a data path, as well as the adaptation along such data path. The basic building blocks of a SEcS's application is a WBI MEG, that is specialized for a particular task (e.g. transcoding, compression, etc.). Complete applications are built by composing MEGs in what we call an *Edge computing Services or EcS*. The chaining operation is very simple: the output from one MEG becomes the input to the next processing MEG.

Porting an existing WBI application in SEcS is trivial, unless the application heavily relies on additional local services that have to be made scalable and deployed on the cluster as well (for example, if a DB is used to store large amounts of data for each HTTP transaction).

Details about the service (such as a short description, location, cost/byte, version, etc.) are provided (by the programmer) into the Service itself and are made available to the SEcS Manager at startup time. Then, the FrontEnd-EPS can show the details of all the available Services to the user to allow the personal configuration. The HTML page that allows the configuration change is built on-the-fly and shown by a FrontEnd-EPS when the URL (`http://_chooser`) is chosen by the user. In this way, users can easily change his/her configuration by choosing among the available services.

3. EXPERIMENTAL RESULTS

We briefly report, here, on an extensive benchmarking of our architecture on a cluster of workstation.

Performances can be defined as the capability of a system to do what needs to be done, as quickly and efficiently as possible. Our goal is to evaluate the performances and the scalability of SEcS's architecture under realistic workload conditions. We, first, define our workload model and, then, describe the experiments.

Workload Model

Our workload model includes the most recent results on the Web Proxy Workload characterization. In particular we took into account the following workload parameters: (*a*) file size distribution is modeled through a hybrid distribution where the body follows the lognormal distribution and the tail the Pareto distribution [4, 30], (*b*) resource popularity is modeled through the Zipf-like distribution [4], (*c*) one-time referencing and temporal locality are modeled through the Lognormal distribution [3, 5, 7]. A summary of our workload model and the chosen parameters for each distribution are shown in the Table 1.

We have considered the response time as performance metric of the evaluated system and the 90-percentile and cumulative distribution functions as statistical measurement.

In our work ProWGen (Proxy Workload Generator) [12] is used to analytically synthesize Web proxy workloads and the httperf benchmark tool [28] for generate the synthetic Web traffic (a stream of HTTP requests that adheres to the various workload parameters and that must be applied on the tested system) from the selected workload model.

Category	Distribution	Parameters	Formulas
Resource size (tail)	Pareto	α=0.1 k=10000	$\alpha k^{\alpha} x^{-\alpha-1}$
Resource size (body)	Lognormal	$\mu = 7000$ $\sigma = 11000$	$\frac{1}{x\sqrt{2\pi\sigma^2}} e^{\frac{-(lnx-\mu)^2}{2\sigma^2}}$
Temporal Locality	LRU Stack	100	
Resource Popularity	Zipf-like	c=0.15	$P(r) = kr^{-c}$
One-timers	Lognormal	70%	$\frac{1}{x\sqrt{2\pi\sigma^2}} e^{\frac{-(lnx-\mu)^2}{2\sigma^2}}$
Correlation popularity-resource size		0	

Table 1: Workload Model.

Testbed Architecture

The Flatland cluster is composed by 10 nodes, 8 of them run the SEcS'intermediary entities (one Remote-EPS and one FrontEnd-EPS per node), the ninth acts as client running httperf [28], and

finally the tenth, Wonderland, acts as Web server.

All nodes of the cluster have the following system properties: each node is a dual Intel XEON 2.66Ghz stepping 05, 2GB of memory, 36 Gb HD with a controller Adaptec AIC7902 Ultra320 SCSI, and finally, Linux Red Hat 9 operating system (kernel 2.4.10). The nodes of the cluster are interconnected through an Intel 82545EM Gigabit Ethernet network interface.

The Wonderland node, that acts as Web server running Apache HTTP Web server 2.0, is connected on the Gigabit Ethernet switch; it is a dual Intel(R) Xeon(TM) CPU 2.40GHz with a 1GB of memory, 72 Gb HD with a controller aic7892 Ultra160, and finally, Linux Red Hat 9 operating system (kernel 2.4.10).

Performances

In each test we compare SEcS performances (with K2 algorithm) with the standard solution of replicating n independent sessions of WBI, each per node, named nWBI. We have also conducted experiments on the performances of the different load balancing algorithms but we cannot fully include them here because of lack of space. We can only say that, in general, K2 is better than the other two algorithms that, anyway, in very few cases perform better than K2. Before we describe the experimental results, two important advantages that we give to nWBI are to be explicited:

1. SEcS architecture handles the mechanism of user's authentication which, on the contrary, is not managed by the standard WBI architecture;

2. we assume that nWBI can rely on an external mechanism to equally balance all the incoming requests to each independent WBI.

We begin by showing the performance comparison according to different services and report the result (because of space limitation) only on two types of services: the Compression-EcS, that performs the compression on-the-fly of the Web resources requested by the clients, and the Transcoding-EcS that realizes the transcoding (i.e. the conversion of images by reducing size, resolution, or color depth) of the Web resources requested by the clients. The Transcoding-EcS uses the freely available ImageMagick library version 5.5.7 [22] to adapt the resources to client capabilities. In particular as interface to ImageMagick we have used JMagick [23], that is implemented in the form of Java Native Interface (JNI) into the ImageMagick API.

Let us start with the first test, where on the subsumed request stream the Compression-EcS has been applied. From the plot shown in Fig. 3 we can see that SEcS is able to achieve the 90-percentile of the response time of 45 ms, while the analogous percentile is equal to 55 ms in nWBI.

The difference between the curves of nWBI and SEcS is slight, but it is due to the fact that the Compression service has been applied on Web resources whose average size is lower than 15KB (because of the workload).

Different results have been achieved when on the request stream has been applied the Transcoding-EcS. In fact, from the plot in the Fig. 4 we can see that SEcS is able to achieve a 90-percentile of the response time of 40 ms. The analogous percentile is equal to 145 ms for nWBI.

The consistent improvement is due to the fact that the Transcoding service is more computational expensive than the compression service and then, distributing the requests among the nodes of the cluster, taking into account load information, is able to guarantee better response times.

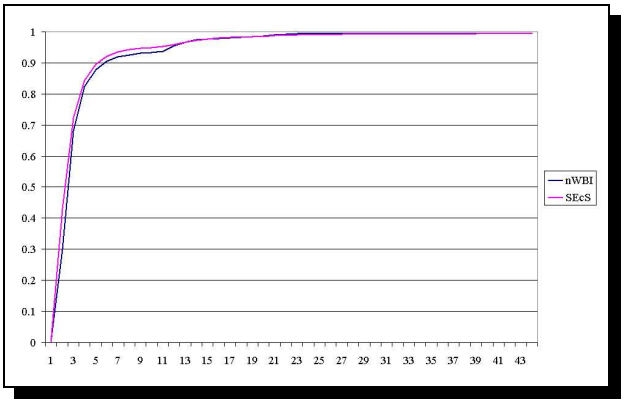

Figure 3: Compression Edge Service (ms*10).

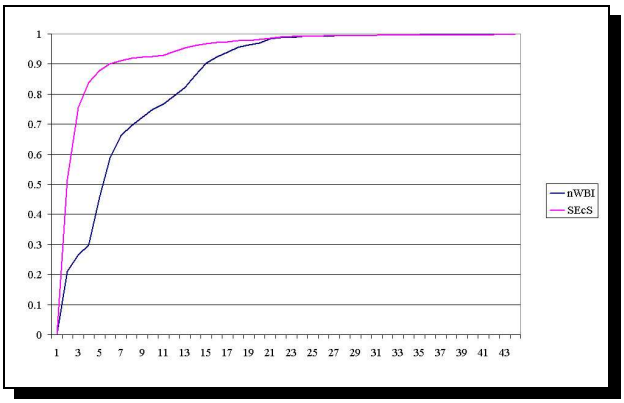

Figure 4: Transcoding Edge Service. (ms*10).

Finally, in the last test reported here, we have stressed the 50% of the *Edge* proxy servers with the 90% of the requests while the remaining *Edge* proxy servers received the 10% of the HTTP requests, by handling only a small fraction of the overall traffic. Our goal, with this test, it to prove that SEcS K2 outperforms or SEcS LL outperform SEcS RR (whereas SEcS RR employs the round robin algorithm for load balancing, SEcS LL the Least Loaded algorithm and SEcS K2 the Mitzenmacher's algorithm).

As we expected, from the plot in Fig. 5 we can see that SEcS K2 is able to achieve the 90-percentile of the response time of 60 ms (75 ms for SEcS LL, 90 ms for SEcS RR) while the analogous percentile is equal to 345 ms for nWBI

Our conclusions is that our software infrastructure is able to handle a high number of requests (end users) for complex services and that our system scales much better than a replicated solution (nWBI) even when authentication and even distribution of requests on cluster nodes are given for free.

4. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a framework to build and deploy Edge-computing Services on clusters of heterogeneous machines.

Our goal with SEcS is to establish an overlay network of intermediary entities that provide mechanisms for the implementation and the composition of value-added services, that exhibit scalability, fault tolerance, high availability and robustness, and that allow programmable functionalities. In addition SEcS allows an easy, "on-the-fly" and per-user configuration of services.

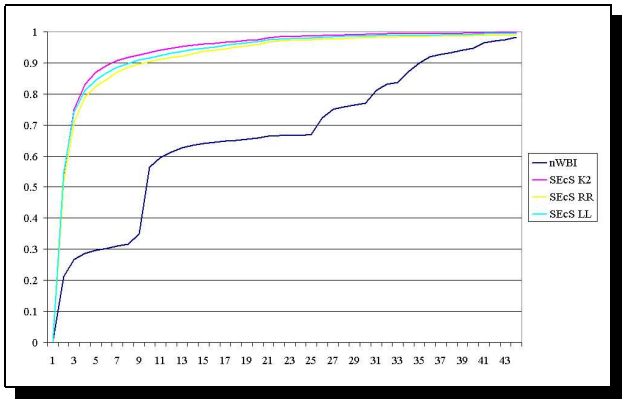

Figure 5: Transcoding Edge Service. *HeavyLight* Distr. (ms*10).

Placing services on the edge of the network permits to offer "orthogonal" services (i.e. general-use services, applied to existing Web resources) such as translation, transcoding and accessibility services, as well as services based on assembling and composing resources on the Web based on the nature of the client and the nature of the accessed resources such as groupware, localization and personalization.

By using a workload model that includes the most recent results in Web Proxy workload characterization, we conducted experiments (only partially reported here) that assessed the efficiency and the scalability of our architecture that are complemented by the easyness of programming of the model, that is based on a well-established environment such as WBI.

As a final remark, we would like to emphasize that our project defines a programming framework that fits naturally into the architecture [6] that is coming out from the most recent activities of OPES Working Group. Therefore, an important future work will be to lead our project into the OPES standard once it becomes an accepted standard, in order to develop an Edge services overlay network of intermediary entities distributed on wide area networks.

5. REFERENCES

[1] Akamai, Inc. `http://www.akamai.com`

[2] C. Anderson and P. Domingos and D. Weld. "Personalizing Web Sites for Mobile Users". In Proceedings of the 10^{th} International WWW Conference, Hong Kong, 2001.

[3] M. Arlitt. "A performance study of Web servers". Master's Thesis. University of Saskatchevan, 1996.

[4] Arlitt M., Friedrich R., T. Jin. "Workload Characterization of a Web Proxy in a cache cable model environment". ACM Performance Evaluation. August 1999.

[5] M. Arlitt, C. Williamson. "Internet Web Servers: Workload Characterization and Performance Implications". IEEE/ACM Transaction on Networking, VOL 5, NO. 5. pp. 631-645. October 1997.

[6] A. Barbir, R. Chen, M. Hofmann, H.Orman, R. Penno and G. Tomlinson. "An Architecture for Open Pluggable Edge Services (OPES)". `http://www.faqs.org/ftp/internet-drafts/draft-ietf-opes-architecture-04.txt`, December 11th, 2002. Web site `http://standards.nortelnetworks.com/opes`.

[7] P. Barford, and M. E. Crovella. "Generating Representative Web Workloads for Network and Server Performance Evaluation". In Proceedings of ACM Performance 1998 / SIGMETRICS 1998, pp. 151-160. Madison, WI. July 1998.

[8] R. Barrett and P. P. Maglio, "Intermediaries: New Places for Producing and Manipulating Web Content". In Proceedings of the 7^{th} International WWW Conference, 1998.

[9] R. Barrett and P. P. Maglio. "Intermediaries: An approach to manipulating information streams. IBM Systems Journal, 1999, 38(4): 629-641, 1999.

[10] R. Barrett and P. P. Maglio. "Adaptive Communities and Web Places". In Proceedings of the 2^{th} Workshop on Adaptive Hypertext and Hypermedia, ACM HYPERTEXT 98, Pittsburgh (USA), 1998.

[11] R. Barrett and P. P. Maglio. "WebPlaces: Adding people to the Web. In Proceedings of the 8^{th} International WWW Conference, Toronto (Canada), 1999.

[12] M. Busari. Prowgen. Tehnical Report, University of Saskatchewan, 2000.

[13] I. Cooper and I. Melve and G. Tomlinson. "Internet Web Replication and Caching". RFC 3040, Taxonomy, January 2001.

[14] M. Dahlin. "Interpreting Stale Load Information. In Proceedings od International Conference on Distributed Computing Systems (ICDCS99). Austin, Texas. May 1999.

[15] M. Dikaiakos. "Intermediary Infrastructures fot the World Wide Web". Computer Networks and ISDN Systems, 45(4):421-447, July 2004.

[16] A. Fox and E. A. Brewer. "Reducing WWW latency and bandwidth requirements by real-time distillation." In Proc. of the 5th International World-Wide Web Conference, May 1996.

[17] A. Fox, Y. Chawathe, and E. A. Brewer. "Adapting to Network and Client variation using active proxies: Lessons and perspectives". IEEE Personal Communications, 5(4):1019, 1998.

[18] A. Fox, S. Gribble, E. A. Brewer, and E. Amir. "Adapting to Network and Client Variabilty via On-demand Dynamic Distillation". In the 7th International Conference On Arch. Support for Prog. Lang. And Operating Systems. (ASPLOS-VII), October 1996.

[19] A. Fox, S. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. "Cluster-based scalable network services. In Proceedings of the sixteenth ACM Symposium on Operating systems principles, pages 7891. ACM Press, 1997.

[20] S. D. Gribble et al. "The Ninja Architecture for Robust Internet-Scale Systems and Services". Special Issue of Computer Networks on Pervasive Computing. `http://ninja.cs.berkeley.edu/`

[21] M. Hori, G. Kondoh, K. Ono, S. Hirose, S. Singhal. "Annotation-Based Web Content Transcoding". In Proceedings of the 9^{th} International WWW Conference, Amsterdam (The Netherland), 2000.

[22] ImageMagick 5.5.7, 2003. `http://www.imagemagick.org`

[23] JMagick 5.5.6-0. http://www.yeo.id.au/jmagick/

[24] R. Katz, E. A. Brewer. "Bay Area Research Wireless Access Network (BARWAN)". `http://www.cs.berkeley.edu/$\sim$randy/Daedalus/BARWAN/BARWAN_index.html`

[25] A. Mahanti, C. Williamson, D. Eager. "Workload Characterization of a Web caching hieratchy". Mobile Networks and Applications. 2000.

[26] M. Mitzenmacher. "How Useful is Old Information". In Proceedings of Sixteenth Symposium on the Principles of Distributed Computing (PODC 97). pp. 83-91, 1997.

[27] M. Mitzenmacher. "How Useful is Old Information". In Proceedings of IEEE Transaction on Parallel and Distributed Systems, 11(1), January 2000.

[28] D. Mosberger and T. Jin. "Httperf: A Tool for Measuring Web Server Performance."

[29] Netscape, "Navigator Proxy Auto-Confi g File Format". `http://www.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html`, March 1996.

[30] J. E. Pitkow, M. E. Crovella. "Summary of WWW Characterization". In Proceedings of International WWW Conference, 1999.

[31] Stardust.com. "Content Networking and Edge Services: Leveraging the Internet for Profi t". White Paper, September 2001. `http://www.stardust.com/cdn/`

[32] SuperProxy, SuperProxy Script, `http://naragw.sharp.co.jp/sps`.

[33] H. Weinreich and W. "Concepts for improved visualization of Web link attributes Lamersdorf". In Proceedings of the 9^{th} International WWW Conference, Amsterdam (The Netherland), 2000.