World Wide Web (2006) 9: 317–341
DOI 10.1007/s11280-006-8559-x

# A Scalable Cluster-based Infrastructure for Edge-computing Services

**Raffaella Grieco · Delfina Malandrino · Vittorio Scarano**

Received: 14 February 2005 / Revised: 15 December 2005 /
Accepted: 5 January 2006 / Published online: 8 June 2006


**Abstract** In this paper we present a scalable and dynamic intermediary infrastructure, SEcS (acronym of "Scalable Edge computing Services"), for developing and deploying advanced *Edge* computing services, by using a cluster of heterogeneous machines. Our goal is to address the challenges of the next-generation Internet services: scalability, high availability, fault-tolerance and robustness, as well as programmability and quick prototyping. The system is written in Java and is based on IBM's Web Based Intermediaries (WBI) [71] developed at IBM Almaden Research Center.

**Keywords** Edge Services · intermediary systems · World Wide Web · proxy servers · personalized and mobile services

## 1. Introduction

The World Wide Web has been, in the last decade, the engine that has pushed the Internet from being an obscure, academic and "technical" (i.e., not user-friendly) network to the widespread diffusion and popularity in every industrialized country. Such popularity has leveraged on a well-known, widely deployed and open framework of protocols that make universally accessible a whole universe of information.

However, with the exponential growth of the Web and its explosive success, more and more dynamic, multimedia and interactive services are increasingly offered to

R. Grieco · D. Malandrino · V. Scarano (✉)
Dipartimento di Informatica ed Applicazioni "R.M. Capocelli", Università di Salerno
via Ponte don Melillo, 84084 Fisciano, SA, Italy
e-mail: vitsca@dia.unisa.it

R. Grieco
e-mail: rafgri@dia.unisa.it

D. Malandrino
e-mail: delmal@dia.unisa.it

end users. In particular, access to the Web comes from a wide range of heterogeneous devices that connect to Internet through different wireless technologies, such as, Bluetooth, GPRS, UMTS, etc.

These trends have involved a growing demands for added-value proxy services infrastructures, that is, for network infrastructures that are able to efficiently provide *intelligent* services, such as personalization, localization, adaptation services, etc., at the *edge* of the network, as close as possible to end users, that are, indeed, the ultimate judges of the quality of the services.

The recent trend in the area of Internet services [1, 19, 36, 41–44] is how to develop and deploy intermediary infrastructures strategically distributed through the network, in order to allow complex intermediary functionalities on the HTTP request/response flow exchanged between clients and servers. In this field, we can cite WBI [19, 20] of IBM as well as BARWAN [55] and Ninja Projects [47] 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 IETF Working Group for "Open Pluggable Edge Services (OPES)" [64]. Among their results, it is defined an architecture [12, 13] of composable edge services [68] that realize an efficient provisioning of complex and added-value services to end users. A thorough and updated classification of intermediary systems is presented in [36].

In literature, *edge* services all share some important non-functional requirements such as programmability and versatility. In fact, heterogeneity and quick obsolescence of devices requires a number of specific services to be easily assembled and provided to clients. In our vision, *Edge computing services* (EcSs) must also have the following characteristics. First, they have to be dynamically reconfigured and must easily interact with other EcSs, by cascading them with other services. Therefore, a crucial capability is allowing the service user[1] to (re)configure the EcS that is actually used, by changing some parameters and/or placing it in a pipeline with other EcSs. Full configurability of EcSs is greatly enhanced by another important characteristic: the personalization, i.e., services can be provided to each user, once he/she is recognized by the system through some authentication mechanism. As defined in [45], this can be realized by using the Proxy Authorization HTTP header Proxy-Authorization. In this way each user has the full control and can manage the configuration of EcSs as she prefers.

Finally, given the growing number of users and the complexity of the services, crucial are the requirements of scalability, high availability and fault tolerance. In this context our research is focused on designing an architecture on a cluster of heterogeneous machines that becomes a "natural" platform to deliver these efficiency requirements.

Many are the application fields of Edge-computing Services that our vision encompasses: from the *geographical personalization* of the navigation of pages, with insertion/emphasis of content that can be related to user geographical location, to *translation services* [71]; from support for *group navigation and awareness* [18, 28], for social navigation [15, 16] to advanced services for *bandwidth optimization* such as adaptive compression and format transcoding [6, 48, 50, 66].

[1] Notice that the service user is, in general, placed on the client-side. EcS users can also be on server side (i.e., providers) but configuration is necessary as well.

A particularly useful application field for EcSs is the accessibility of Web sites. These applications are usually heavy-weight and therefore, assuring their scalability is particularly critical. A typical example is an ubiquitous service that provides text-to-speech [17] translation of the pages navigated by the user, regardless of the location and configuration of the local machine (i.e., without installing particular software).

Of course "traditional" proxy services like caching can be further enhanced by combination with advanced EcSs: an example can be offering the user a "search engine" that operates only on cached/personalized/translated pages.

### 1.1. Related results

The BARWAN project [55] by UC/Berkeley has the goal to provide an intermediary system that is able to support ubiquitous access to Internet services from mobile and thin clients. It also realizes Web content adaptation in order to meet the different network conditions and the widespread client heterogeneity. An important system component of this project is the proxy architecture, TACC, that acts as intermediary between servers and mobile clients. The TACC programming model provides important functionalities: transformation, aggregation, caching and customization of the Web content [25, 44]. The main components of the TACC compositional framework are the workers, software entities that represent the building blocks of TACC applications. They encapsulate the core functionalities of the services offered to the end users, and, in addition, they may be combined in order to provide more complex service applications. Finally, TACC runs on a cluster of workstations. The evolution of the TACC model is then provided in the Ninja Project [47] by UC/Berkeley. The Ninja project seeks to develop a scalable and robust architecture that provides Internet Services. Important components of the Ninja Architecture are the Active Proxies that provide intermediary functionalities between Ninja services and terminal devices, such as, dynamic adaptation, distillation, caching, etc. Services can be combined and dynamically invoked along data paths. Finally, the Ninja architecture runs on a cluster of workstations.

iMobile architecture [67], by AT&T Research, aims to hide the complexity of multiple devices and content sources from mobile users. It provides a framework for developing and composing intermediary components in complex distributed applications. Its main component is iProxy [10], a programmable proxy server that provides an environment for hosting personalized services, which are implemented as reusable building blocks in Java. iProxy is a classical example of a programmable proxy that allows accessing, caching, and processing of Web data.

The *extensible Retrieval Annotation and Caching Engine* (eRACE) [37, 38] is a modular and programmable intermediary infrastructure whose main goal is to provide personalized services for a wide range of client devices (Desktop PC, mobile and thin clients), such as, personalization, customization, filtering, aggregation, transformation both on wireline and wireless Internet. eRACE includes agent-proxies like WebRACE (for retrieving and caching contents from the Web), mailRACE (for POP3 mail accounts), newsRACe (for USENET NNTP-news) and dbRACE (for managing databases). The most important is WebRACE, a proxy accessible through the HTTP protocol. Is developed in Java and consists of a distributed crawler and an Object cache.

Another interesting work is MARCH, a distributed architecture [5, 6] that provides multimedia content adaptation services, in order to match the access network capabilities of different client devices and user's preferences. With respect to the infrastructure services described above, MARCH adopts a server-centric approach.

ALAN, Application Level Active Network [46], is a network architecture that provides mechanisms to dynamically load code, or proxylet, to facilitates user based content delivery [57]. Communications is enhanced by Dynamic Proxy Server that are located at strategic points along the path between client and server. Proxylet can be downloaded, as JAR archive, from HTTP servers into a Dynamic Proxy Server infrastructure, making their deployment very easy (URL to reference proxylets). The main contributions of this approach are: (a) the fast deployment of new communication services on demand without the drawbacks of the Active Networks (deployment of new elements at network routers) and (b) a platform for flexible, added-value services provision (WWW streaming audio, WWW compression, etc.).

### 1.2. A platform for scalable EcSs

Having stated our case for Edge-computing Services, and having presented related results, here we come to the objective of the project presented in this paper: a framework for easy development and deployment of efficient Edge-computing Services on a cluster of machines.

Our proposal, named Scalable Edge-computing Services (SEcS), is based on IBM Web-Based Intermediaries (WBI) [19, 20] 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.

SEcS allows to build scalable services that enhance the simplicity and the ubiquity of the WWW: additional, complex and advanced services can be placed on the edge of the network in order to intercept and deal with HTTP information flows. Services built with SEcS are placed on top of the three foundational standards of the Web (HTTP, URL, HTML) and have minimum impact on the users because of their independence from user's HW/SW platform.

In conclusion, we believe that SEcS makes possible to envision the "edge" of the network as a real computing platform where services can be assembled and personalized.

#### *1.2.1. Organization of the paper*

In the next section we first describe SEcS architecture, by presenting Web Based Intermediaries, and then some SEcS service's assumption and examples. In Section 3 we describe the components of the SEcS architecture and describe the algorithms implemented to balance the workload among the cluster nodes.

Then, we analyze the performances of our framework in Section 4 where, first we describe the workload model that we have used to test our prototype and, then, we describe the experimental results that we obtained. Final comments in Section 5 will conclude the paper.


**Figure 1** SEcS architecture overview (*left*) and the dataflow between FrontEnd-EPS and Remote-EPS (*right*).

## 2. SEcS architecture

In this section we introduce the architecture of our framework. First of all, we briefly need to summarize the relevant aspects of Web Based Intermediaries (WBI) in order to delineate the starting point. Then we describe SEcS architecture and provide a high-level introduction to services definition and personalization.

### 2.1. 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 written in Java and has been used in various contexts [3, 18, 21, 48, 73]. Its technology lies at the heart of parts of commercial products of IBM such as WebSphere Transcoding Publisher [50]. WBI acts as an HTTP request processor, receiving a request and sending a response as customary for HTTP proxies defined in [40, 56]. A typical WBI transaction flows through a combination of the following four type of basic stages, called MEGs:

- *Request Editor (RE)* that receives a request and can modify it before passing it along a path towards its destination.
- *Generator (G)* that receives a request and produces a corresponding response.
- *Editor (E)* that receives a response and can modify it before passing it along a path.
- *Monitor (M)* that receives a copy of request and response but it cannot modify the data flow.

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

WBI can be programmed to operate like a conventional Web server, a transparent Web proxy or can perform any of a range of intermediary functions, such as Web Personalization, Caching and Transcoding.[2]

### 2.2. An overview of SEcS

Our service architecture, whose overview is shown in Figure 1 on the left, 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 to be transformed or adapted by remote components (for services not locally available or more efficiently available on remote machines).

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.

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 keeps the internal status in terms of addresses (i.e., hosts) of available edge services, provided by 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 [69] that implements an hash function to evaluate the target FrontEnd-EPS to be used.

During the early phases of the project, we experimented with different prototypes of the SEcS framework, based on different communication mechanism among Remote-EPSs and FrontEnd-EPSs. The first prototype (R-SEcS) was based on Sun's Java Remote Method Invocation (RMI) [52] as communication protocol. A communication between a FrontEnd-EPS and a Remote-EPS happens through a *Stub* on the local machine that is used by the Remote MEGs to exchange data. Then, we moved to a prototype that used KaRMI, efficient RMI for Java, [54] as communication protocol. In the final implementation, S-SEcS, RMI communication protocol is replaced by a more efficient communication mechanism via sockets. In this context, we provide the programmer with object marshalling and un-marshalling methods, that makes S-SEcS as easy to use, for a WBI programmer, as the previous, RMI-based, versions. Because of efficiency, we have used the Java API SocketChannel since it provides non-Blocking I/O and direct access to memory buffer.

In the rest of the paper we will always refer to this last implementation of SEcS.

### 2.3. SEcS services

In this section we discuss some important characteristics of SEcS services, their definition, composition and configuration, and some of the more interesting services that we have implemented on a cluster of workstations.

[2] For other examples of WBI applications see the WBI Web site at [71].

### 2.3.1. Services definition and composition

SEcS 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 MEG, that is specialized for a particular task (e.g., transcoding, compression, etc.). Complex applications are built by composing MEGs in what we call an *Edge computing Services or EcS*. The chained operation is quite 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. In Figure 2 we show an example of the way a user can change his/her configuration by choosing the available services and the information related to the service. The HTML page that allows the configuration change is built and shown by a FrontEnd-EPS when the URL (http://_chooser) is chosen by the user.

### 2.3.2. Example services

Many example services we built and experimented on the top of SEcS. We would like to remind the reader that, from a WBI programmer point of view, *edge* services

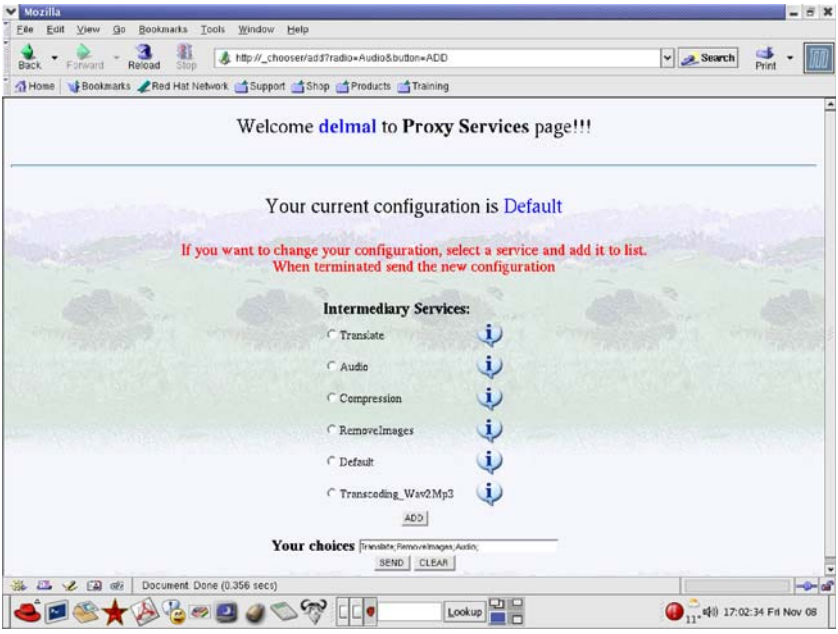


**Figure 2** User personal configuration.

built with SEcS are the same as those built with WBI, i.e., there is no difference in the programming model as well as a full compatibility with WBI plugins. Therefore, porting services from WBI to SEcS is an absolutely trivial task.

In Figure 3 we show shows the typical HTTP transaction data flow between clients and SEcS and how the Remote-EPS offers different services that are composed according to user's preferences.

One of the most important services that we realized is TextToSpeech [17] that has been designed to make Web content accessible to people with visual disabilities. The nature of this service makes it an ideal testbed for our definition of Edge-computing Services since it is a heavy-load service that should be ubiquitous, i.e., regardless of the software installed on the local client, whereas, on the contrary, other solutions require the local installation of a particular software to navigate (such as IBM Home Page Reader [49] or CAST eReader [30]). To implement this service we have followed the W3C Guidelines [72] that discuss accessibility issues and provide accessible design solutions.

Moreover, the TextToSpeech service can help the comprehension of documents that are written in foreign languages and provide a support for kids that want to learn a language by providing them a written/spoken presentation of material of their interest during a "natural" experience as navigating the Web.

Another main field of applications of EcS is the cooperation. In fact, several additional functionalities can be obtained by leveraging on intermediary systems. In particular, the Cooperative Navigation-EcS offers a collaborative environment to store and retrieve navigation paths and supports the paradigm of social navigation since it offers full awareness of what other users in the group are navigating. Making the Web a social place means: (*a*) enable users to take advantage of other people's activity by reducing the time spent for searching information on the Web, (*b*) to become aware of interests and knowledge of the own research's group, (*c*) get recommendation on useful information based on other people's experiences [15, 16].

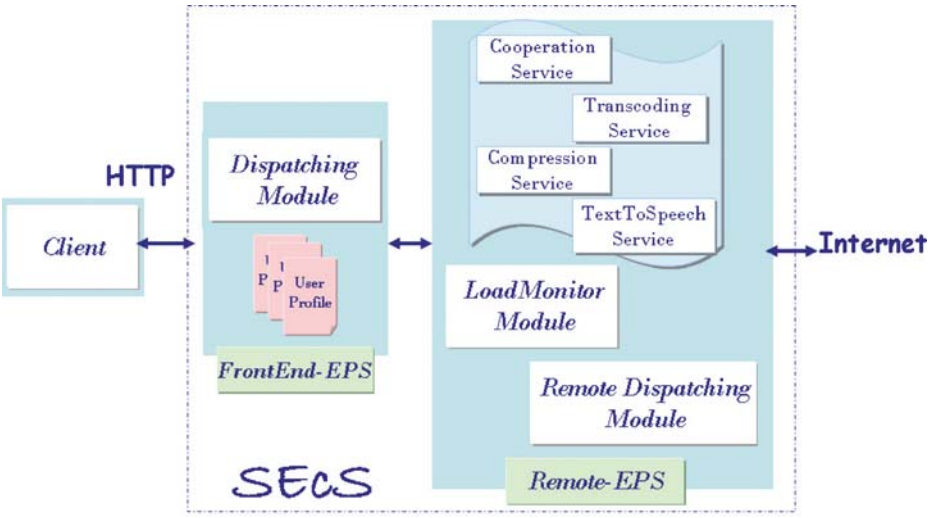


**Figure 3** The roles of SEcS components and their position in the architecture, between the user that requests the service and the server.

In collaborative environments users can exchange information about their navigation through various mechanisms, such as, path recording, annotation and editing. Each user, during his/her navigation, can create a permanent and modifiable collection of URLs that can be shared with other users. As an example application, a typical activity of a researcher is to navigate to find relevant references to his/her current research interests; once the user found and annotated a set of URLs, he/she can share them with her research team, therefore increasing, as a result, the group's knowledge.

## 3. SEcS components

In this section, we refine our description of SEcS, by presenting, first, the components and how they interact with each other, according to the scheme shown in Figure 1. Then, we deal with the load balancing issue and present the load balancing algorithms that have been implemented and tested, as described in the next section.

### 3.1. 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 [32, 63, 69]. The file, with .pac extension, can be located everywhere (on a local or remote WWW server or locally on the client machine). Its main goal is to avoid a bottleneck on a single FrontEnd-EPS, by offering a transparent mechanism for balancing the requests among the FrontEnd-EPSs of the system.

In particular, the CAF defines a function FindProxyForURL that returns, for each accessed URL, which proxy could be used. The parameters that can be used, in general, by this function are the hostname of the client and the URL that has been requested. In particular, our choice was to perform the hashing on the hostname, and not on the URL; in fact, if different FrontEnd-EPSs are used during the same "session" (from the user point of view) a new challenge-response authentication is

```
function FindProxyForURL(url, host){
  res=hash(host);
  if ((res % 3)= 1)
   return "PROXY agarthi.dia.unisa.it:8103;
           PROXY wonderland.dia.unisa.it:8203;
           PROXY atlantide.dia.unisa.it:8303;
           DIRECT";
  if ((res % 3)= 2)
   return "PROXY wonderland.dia.unisa.it:8203;
           PROXY atlantide.dia.unisa.it:8303;
           PROXY agarthi.dia.unisa.it:8103;
           DIRECT";
  else
   return "PROXY atlantide.dia.unisa.it:8303;
           PROXY agarthi.dia.unisa.it:8103;
           PROXY wonderland.dia.unisa.it:8203;
           DIRECT";
}
```

**Figure 4** An example of a CAF. The function hash () is also in the script and is omitted.

required for each reached FrontEnd-EPS by deeply annoying the users during their navigation.

We also used the CAF as a way to guarantee automatic and transparent fault-tolerance of the Edge-computing Services. In fact, if the proxy chosen by the script is no longer available, then the browser automatically chooses the next proxy (designated by the script) and, as last option, it chooses the DIRECT option i.e., direct connection with no use of services at all (Figure 4).

Autoconfiguration files also support proxy failover, so if a proxy server is unavailable, the browser will transparently switch to another proxy server. Other mechanisms include Round Robin DNS [26] or commercial routers such as Cisco LocalDirector [31].

### 3.2. 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.

The goal of the challenge-response authentication mechanism is to verify that the user making a request is authorized to do so, or that user is charged for a specified operation. To this end, we have realized a specific Proxy-Authorization-EcS, that is useful both for restricting access to a proxy server as well as to distinguish between users. Indeed, if we are interested in services that deal with users based on per-user settings, it is important to provide a mechanism that is able to distinguish between them. By realizing a service that provide such challenge-response mechanism, each user has the full control and can manage the configuration of EcSs he/she needs.

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 with it.

FrontEnd-EPSs also take from the transaction data[3] the information about the load of the remote machines where the services were executed. More details on the monitoring and load balancing information are presented in Section 3. They can be replicated for both scalability and availability.

Finally, the FrontEnd-EPS takes care of the configuration and the 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 (see Figure 2).

### 3.3. 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) MEGs. An EcS can consist of a single MEG as well

[3] Transaction data, in WBI, are sent within each request as it flows through the MEGs.

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-EPSs. The SEcS Manager broadcast a table of active services to all the FrontEnd-EPSs.

The *Remote Dispatching* Module in Remote-EPS (see Figure 3) is needed to choose the best Remote-EPS for the next MEG of the service (when the execution of a service requires more chained MEGs to be invoked on the HTTP client request) while the *Load Monitor* module collects the system state information (every 10 s).

Several services can be offered by Remote-EPSs, such as: service personalization, aggregation from multiple source, language translation, virus scanning, content adaptation, filtering etc. Such services are unambiguously identified with the combination of two information, (*a*) the service URL and (*b*) the specific-task name (typical mechanism used by Internet services for service location). The services offered by Remote-EPSs can be replicated on many nodes of the cluster in order to guarantee scalability, availability and fault-tolerance. The number of FrontEnd-EPSs and Remote-EPSs 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 FrontEnd-EPS and a Remote-EPS.

### 3.4. SEcS manager

The SEcS Manager is a Java application that may be located on any machine in the network. Its main goal is to act as a name server for the rest of the system. It is invoked by the FrontEnd-EPS for the registration of new FrontEnd-EPSs, and by the Remote-EPSs for the registration of new *edge* services.

If a Remote-EPS crashes before de-registering itself, the SEcS Manager detects the broken connection and communicates the new configuration to all the FrontEnd-EPSs, in order to allow them to update the information about the active services in the system.

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.

The information provided by the SEcS Manager can be used as backup if the Manager itself crashes for any reason. In this way the system will continue to operate normally until the Manager will be restarted. In fact, it is a very easy task to allow a Remote-EPS, that enters the system, to check if the SEcS Manager exists and correctly works, and eventually restart it if a crash occurred. The restarted Manager checks on any node of the cluster (e.g., by checking all the nodes in its subnetwork) to contact any Remote-EPS and get updated information.

### 3.5. SEcS load balancing

Here, we focus on the implementation of the SEcS load balancing algorithms. To this end, we take into account both static and dynamic algorithms:

- *Static dispatching algorithms* They do not consider any state information while making scheduling decisions. Examples are the Random algorithm and the Round Robin algorithm (see [70]).

- *Dynamic dispatching algorithms* They can take into account state information while making scheduling decisions. Examples are the Least Connections algorithm (adopted by Cisco's LocalDirector [31]) and the Least Loaded algorithm (described in the sequel).

Dynamic algorithms, usually, outperform static algorithms as they take into account several information during the process of dispatching decision. On the other hand, they are more complicated to implement because all mechanisms of gathering and analyzing of information are in charge to the developer.

### *3.5.1. 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 number of proxy servers in the cluster.

### *3.5.2. 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, see Figure 1). The values are added (by piggy-backing) into each request that passes through the MEGs running on that machine. 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 a 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.

It should be noticed that the framework allows different policies for load balancing depending on the requested service. The standard default implementation (shown in Figure 5) can be, in fact, easily customized taking into account the nature and complexity of the service.

Finally, the workload of each Remote-EPS is normalized by the local *Load Monitor* thread since it takes into account a locally stored file that contains a table of multiplicative factors that are used to send out "standardized" values about CPU load, free memory percentage, bandwidth and number of active and wait connections.

### *3.5.3. K2 least loaded algorithm*

When balancing the load in a distributed system, it is well-known [35] that the strategy of sending each request to the least loaded machine can involve an instability in the system if information become too old. To solve this problem, also

```
//get the list of the Remote-EPSs
// for the service servreq
proxyList=proxyForService(servs,servreq);

// default policy is instantiated
Policy prop = new Policy();

prop.setDefaultPolicy();
// CPUrate=30; ConnAct=19; ConnWait=13;MemFree=8; BandW=30;

// else customization of the policy goes here
// prop.setCPUrate(20);
// prop.setConnActiverate(10);
// prop.setConnWaitrate(10);
// prop.setMemFreerate(30);
// prop.setBandW(30);

//the Balancer chooses the bestProxy
bproxy=balancer.computeBestProxy(
           proxyList, // the proxy list
           nodes,     // the load of the nodes
           prop);

//get info to perform remote calls
info=(infoService)active.get(servreq+bproxy);


String MegName=info.getmegName();
host=info.gethost();

//Remote Service Invocation....
```

**Figure 5** A (simplified) version of the code for establishing and customizing the load balancing policy. Codes for unexpected events (such as Remote-EPS crash etc.) has been removed.

know as "herd effect" [35], some systems adopt randomized strategies that ignore load information or simply take into account only a subset of them.

Load balancing with stale information is becoming an increasingly important problem for distributed systems. Our solution to this problem follows Mitzenmacher's work [60, 61]. Under this algorithm, if there are $n$ servers, instead of sending the request to the least loaded server, a client randomly selects a subset of size $k$ of the servers, and sends its request to the least loaded server from this subset. Since, in Mitzenmacher analysis [60, 61] 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.

## 4. Performance evaluation

We report, here, on an extensive benchmarking of our architecture on a cluster of workstations. Evaluating the performances of a system means to evaluate the capability of such system to do what needs to be done, as quickly and efficiently as possible. In particular, our goal is to evaluate the performances and the scalability of SEcS 's architecture under realistic workload conditions. To this purpose, we have defined our workload model, that includes the most recent results on the Web Proxy Workload characterization, and then, by using such model, we have realized a set of specifics experiments.

### 4.1. Workload model

By definition, a *workload* consists of a set of all inputs that a system receives over a period of time. Hence, a workload model consists of a collection of parameters that affects resource allocation and system performance [11]. Some examples of Web workload parameters are illustrated in Table 1.

Workload characterization involves modeling each parameter of the workload with a specific or more suitable probability distribution function [11].

Studying the size of Web resources is crucial as it affects the storage requirements on Web server or proxy systems, the network bandwidth and the latency in delivering Web contents to the clients. Several studies [8, 33] of Web traffic show that the distribution of resource sizes is heavy-tailed, that is, very large items in the tail of the distribution are relatively few in number, but they are large enough to contribute significantly to the overall traffic observed [34].

This high variability of Web resources can be modeled through the Pareto distribution:

$$F(x) = \alpha k^{\alpha} x^{-\alpha-1}$$

where $\alpha, k > 0$ and $x \geq k$.

In particular the tails of the distributions, for files larger than 10,000 bytes, fit well to a Pareto distribution with $\alpha \in [0.93, 1.33]$ [8] (The value for $\alpha$ is 1.0 in our workload model).

Since some Web resources are more popular then others, this means that studying their popularity [8, 24, 59] is crucial in reducing server overhead for responding requests to the clients (server-side caching is most effective when a small number of resources is repeatedly requested).

Such studies have also proved that the Zipf's Law distribution [74] can be applied to model this behavior:

$$P(r) = kr^{-c}$$

for some constant $c$.

**Table 1** Web workload parameters.

| Category | Parameters |
|---|---|
| Protocol | Request method |
| | Response code |
| Resource | Content type |
| | Resource size |
| | Response size |
| | Popularity |
| | Modification frequency |
| | Temporal Locality |
| | Number of embedded resources |
| Users | Session interarrival times |
| | Number of click per session |
| | Request interarrival times |

The value of $c$ varies across different Web proxies and servers [2, 24]. Smaller values of $c$ correspond to smaller differences in popularity access through the set of resources. Recent studies have proved that the best value for $c$ is in the range between 0.75 and 0.90, indicating a less variability in popularity among resources. For proxy systems, instead, smaller values of $c$ (0.15 in our tests) are proposed in the literature, suggesting even less skew in popularity. In fact, proxies handle requests for diverse collections of Web sites, which typically result in a larger number of resources with comparable popularity [11].

The temporal locality, that represents the time between successive requests for the same resource, may be a significant impact on the Web traffic. One way to measure temporal locality is using the notion of *stack distance* [2]. Given a sequence of requests, its corresponding stack distance sequence can be computed as follows. Assuming that the most recently referenced item is on the top of the stack and the least recently item is on the bottom, for each new request the stack is searched until the requested resource is found or the bottom of the stack is reached. If the resource is found it is removed from its current position and placed on the top of the stack. On the contrary, if the requested resource is not found in the stack, it is simply added to the top of the stack moving down all the other items. The corresponding depth represents the request's stack distance [2]. Several studies prove that such distance follows a Lognormal distribution [8, 14].

Finally, an interesting observation in analysis of Web systems performance is that, typically, the 15–30% of the Web resources are accessed only once [7]. Several possible explanations of this behavior, known as "One-time referencing," are the vastness of the WWW, the habits of the Web users to visit a Web page only once, caching and the presence of Web resource pre-fetching.

By summarizing, our workload model takes into account the following workload parameters: (*a*) the file size distribution, modeled through a hybrid distribution where the body follows the Lognormal distribution and the tail the Pareto distribution [9, 65], (*b*) the resource popularity, modeled through the Zipf-like distribution [9, 74], (*c*) the one-time referencing and temporal locality, modeled through the Lognormal distribution [7, 8, 14]. A description of our workload model and the chosen parameters for each distribution are showed in the Tables 2 and 3.

Once defined the workload model the next step is to define the performance metrics of the evaluated system. We choose to analyze the response time, that represents the time that elapsed between sending the request and obtaining the corresponding response.

For measuring the performances of SEcS's architecture, we have used the cumulative distribution functions (CDFs) as statistical measurements, that give the probability that a performance index takes a value less than or equal to $x$, that is:

**Table 2** Workload model.

| Category | Distribution | Formulas |
|---|---|---|
| Resource size (tail) | Pareto | $\alpha k^{\alpha} x^{-\alpha-1}$ |
| Resource size (body) | Lognormal | $\frac{1}{x\sqrt{2\pi\sigma^2}} e^{\frac{-(\ln x-\mu)^2}{2\sigma^2}}$ |
| Temporal Locality | LRU Stack | $\frac{1}{x\sqrt{2\pi\sigma^2}} e^{\frac{-(\ln x-\mu)^2}{2\sigma^2}}$ |
| Resource Popularity | Zipf-like | $P(r) = kr^{-c}$ |
| One-timers | Lognormal | $\frac{1}{x\sqrt{2\pi\sigma^2}} e^{\frac{-(\ln x-\mu)^2}{2\sigma^2}}$ |

**Table 3** Workload parameter values.

| Parameter | Value |
|---|---|
| Pareto tail index | 1.0 |
| Beginning of the tail (k) in bytes | 10,000 |
| Zipf Slope | 0.15 |
| LRU Stack Size | 100 |
| Resource Popularity | 0.15 |
| Unique documents (% of total requests) | 30% |
| One-timers (% of unique documents) | 70% |
| Correlation (file size and popularity) | 0 |
| Mean of the lognormal distribution ($\mu$) | 7,000 |
| StdDev of lognormal distribution ($\sigma$) | 11,000 |

$F(x) = P(X \leq x)$. In particular we have used the cumulative distribution function of the response time since in a system with high variability of values it is more significant than average values [29].

Moreover, we have used the ProWGen tool (Proxy Workload Generator) [27] to analytically synthesize Web proxy workloads and the httperf benchmarking tool [62] to 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.

The ProWGen tool is able to capture the salient characteristics of the Web Proxy Workload Characterization: one-time referencing, Zipf-like document popularity, heavy-tailed file size distribution and temporal locality. Moreover, this program outputs a very detailed report structured as a two column text file; for each resource there is an entry with two values: file id and file size and each file id represents a distinct URL in the workload.

The httperf tool, that allows both hybrid and trace-based approaches [4] to generate the request stream, collects several metric measurements, i.e., connection time, latency time, request and reply rate, throughput, etc. Unfortunately, the response time of a single Web resource is not collected, in fact it collects only the mean response time of the overall requests that compose the request stream. For this reason, we have modified httperf in order to provide also this important metric.

Finally, in our tests, we have applied two types of requests' distributions:

- *Constant Load Distribution* All nodes of the cluster receive the same number of client's requests from the request's stream whereas the request's stream does not take into account information about the size and the popularity of the Web resources. Each node receives the same set of requests, that are divided as follows: the 70% of the requests are for image files and the remaining 30% for HTML files. The average file size is 7–15 KB. This percentages are chosen according to the early studies on Web Workload Characterization [9, 58, 59].
- *Real Load Distribution* The request stream, generated by ProWGen, reflects the Web proxy Workload characteristics.

These two types of requests distributions have been employed in order to show how SEcS works in "real" life conditions (*Real Load*) and how it works when *stressed* with "heavy" services (*Constant Load*).

### 4.2. Experimental results

In this section we, first, describe the three different implementations of SEcS (according to the different algorithm implemented for the load balancing) that we have tested and then we give a detailed description of how tests are performed and which are the experimental results.

The three different implementations of \SEcS are the following:

- *SEcS RR* SEcS with the Round Robin algorithm for load balancing. A simple module function is applied by the FrontEnd-EPS in order to chose the target Remote-EPS that will handle the incoming client's request.
- *SEcS LL* SEcS with the Least Loaded algorithm for load balancing. The Remote-EPS with the lowest load will be chosen by the FrontEnd-EPS in order to handle the client's request.
- *SEcS K2* SEcS with the modified Least Loaded algorithm for load balancing. Under this algorithm, if there are $n$ servers, instead of sending the request to the least loaded server, the FrontEnd-EPS, randomly, select a subset of size $k$ of the Remote-EPSs, and sends the request to the least loaded Remote-EPS from this subset.

In each test we compare SEcS performances with the standard solution of replicating $n$ independent sessions of WBI, each per node, named $n$WBI.

Before to describe the experimental results, two important advantages that we give to $n$WBI have to be emphasized:

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 $n$WBI could rely on an external mechanism to equally balance all the incoming requests to each independent WBI.

#### *4.2.1. Testbed architecture*

The Flatland cluster is composed by ten nodes, eight of them running SEcS' intermediary entities (one Remote-EPS and one FrontEnd-EPS), the ninth acting as client with httperf [62], and the tenth, Wonderland, acting as Web server. All nodes of the cluster have the following system properties: each node is a dual Intel XEON 2.66 GHz stepping 05, 2 GB 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 2.40 GHz with a 1 GB of memory, 72 Gb HD with a controller aic7892 Ultra160, and finally, Linux Red Hat 9 operating system (kernel 2.4.10).

In our testbed architecture several changes was required to the Linux kernel configuration to overcome the limitations on both the total and the per-process number of file descriptors that can be opened. In addition we also increased the socket buffer sizes (both send and receive). Finally, all non-essential processes was disabled to minimize the consumption of the resources.

#### 4.2.2. Performances

In our experiments we have considered two main different classes of services' complexity:

- Low Complexity (*Compression Working Test*) We provided the implementation of a Compression-EcS, that performs the compression of the Web resources requested by the clients. This service can become an heavy weight one and, then, expensive in terms of computation power if applied on Web resources with large sizes.
- High Complexity (*Transcoding Working Test*) This class includes examples of heavy services (in terms of consumption of resources). Our example for this class is the Transcoding-EcS that realizes the transcoding (conversion of images by reducing size, resolution, or color depth) of the Web resources requested by the clients. The Transcoding-EcS will be extremely expensive in terms of consumption of resources if applied on very large Web resource sizes.

The Transcoding-EcS uses the freely available ImageMagick library version 5.5.7 [51] to adapt the resources to client capabilities. In particular as interface to ImageMagick we have used JMagick [53], that is implemented in the form of Java Native Interface (JNI) into the ImageMagick API.

#### 4.2.3. Compression working test

In this Working Test we have applied the Compression-EcS and in both the experiments (*Constant Load* and *Real Load*) SEcS (in particular SEcS K2) outperforms *n*WBI.

Before describing in detail the results of this class of complexity, we must anticipate that the Cumulative Distribution Functions (CDFs) of the testbed architectures show a similar behavior. The main motivation of this behavior is that on the request's stream is applied a service that could seem *heavy*, but that, on the contrary, applied on Web resources with smaller sizes does not to achieve very good results.

From the plot in Figure 6 we can see that SEcS LL and SEcS K2 are able to achieve a 90-percentile of the response time less then of 50 ms, while the similar percentile is achieved by SEcS RR and *n*WBI in approximately 55 ms.

Similar results are achieved with the Real Load distribution, in fact also in this test the CDFs of the response time show very similar behaviors.

From the plot in the Figure 7 we can see that SEcS K2, SEcS RR and *n*WBI are able to achieve a 90-percentile of the response time less than of 55 ms. The analogous percentile is equal to 105 ms for SEcS LL. Finally, SEcS LL performs worst since with more heavy services loads on cluster's node grow and then load information begin to become stale, consequently SEcS K2 performs better than SEcS LL.

The reader should remember the advantages that are given (in terms of performances) to the *n*WBI version: no authentication and assumption of an external, free load balancer of the requests.

#### 4.2.4. Transcoding working test

In this set of experiments SEcS is able to achieve definitely better performances with respect to *n*WBI and other SEcS's implementations.

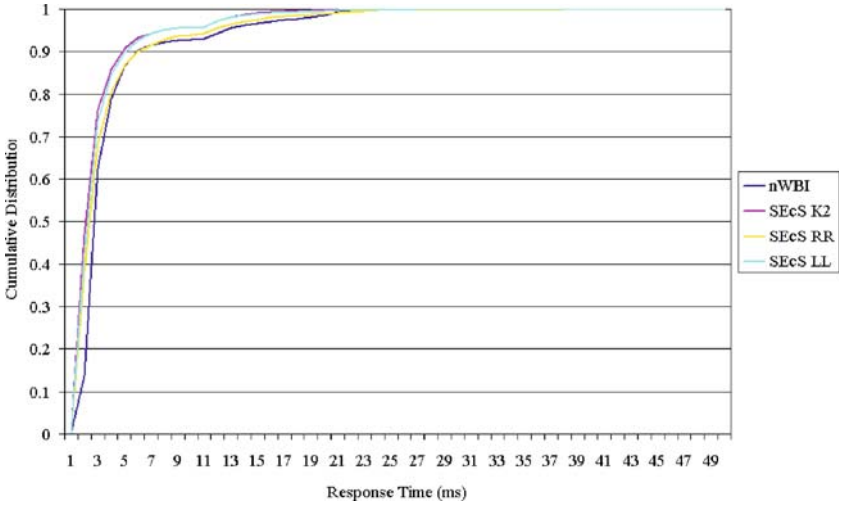


**Figure 6** Compression edge service. Constant load distribution (ms*10).

From the plot in the Figure 8 we can see that SEcS K2 is able to achieve a 90-percentile of the response time of 130 ms. The analogous percentile is less than of 180 ms for SEcS LL and SEcS RR and it is equal to 365 ms for *n*WBI.

As we expected SEcS RR outperforms SEcS LL and SEcS K2, since the applied distribution is the Constant Load one, that is, the requests are equally distributed on the nodes of the cluster. From the plot in the Figure 9 we can see that SEcS LL is able to achieve a 90-percentile of the response time of 50 ms. The analogous percentile is equal to 55 ms for SEcS K2, 70 ms for SEcS RR and it is equal to 145 ms for *n*WBI.

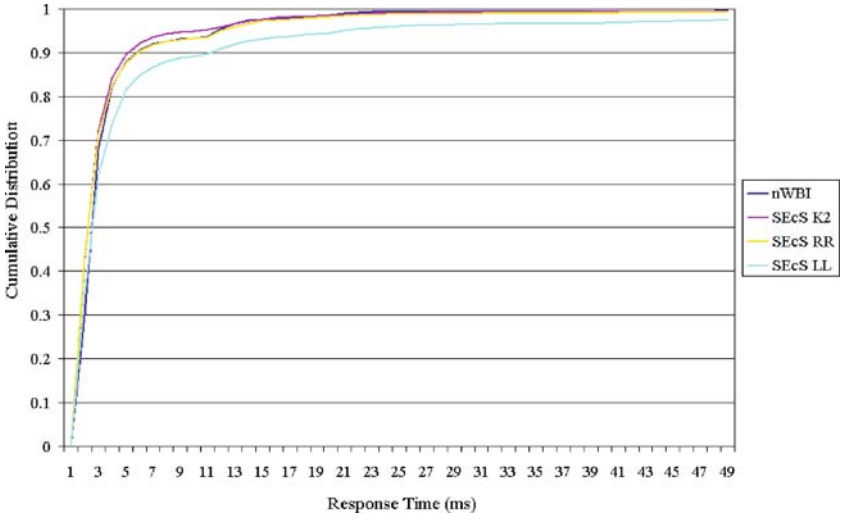


**Figure 7** Compression edge service. Real load distribution (ms*10).

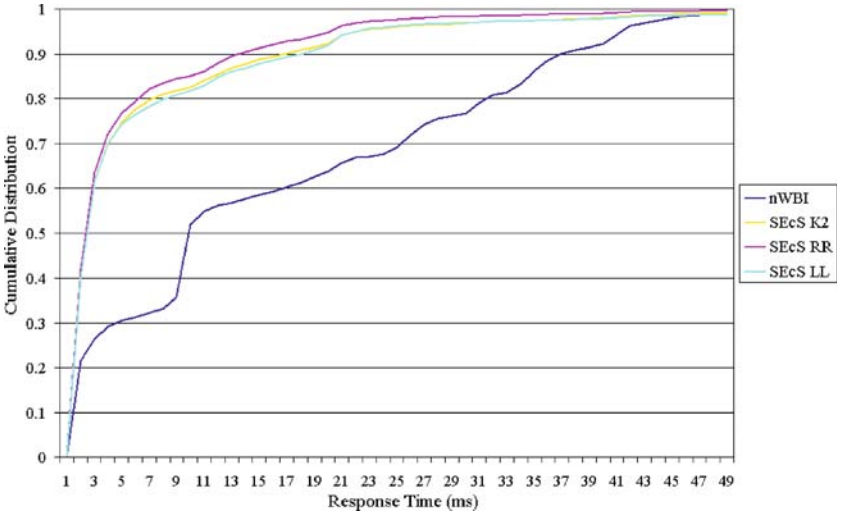


**Figure 8** Transcoding edge service. Constant load distribution (ms*10).

Also this test shows the better results achieved by SEcS. Obviously SEcS K2 and SEcS LL perform better than *n*WBI and SEcS RR since the applied distribution is the Real Load one. 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.

Finally, in the last test 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

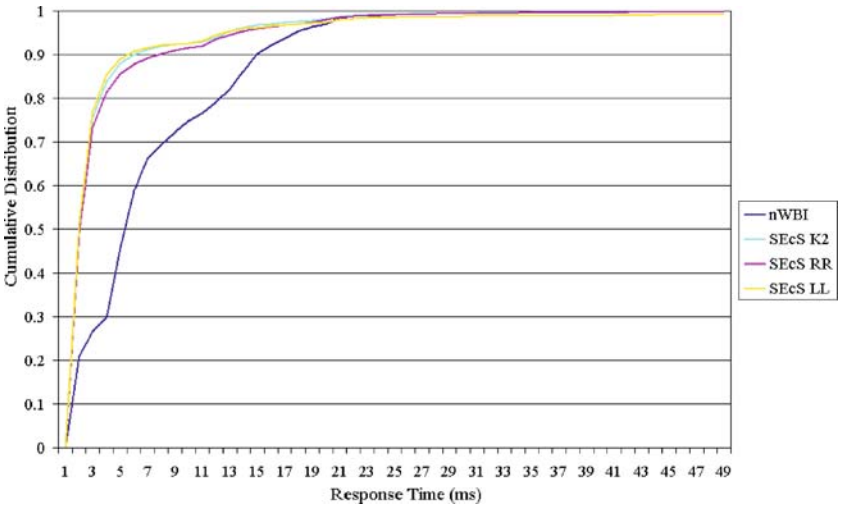


**Figure 9** Transcoding edge service. Real load distribution (ms*10).

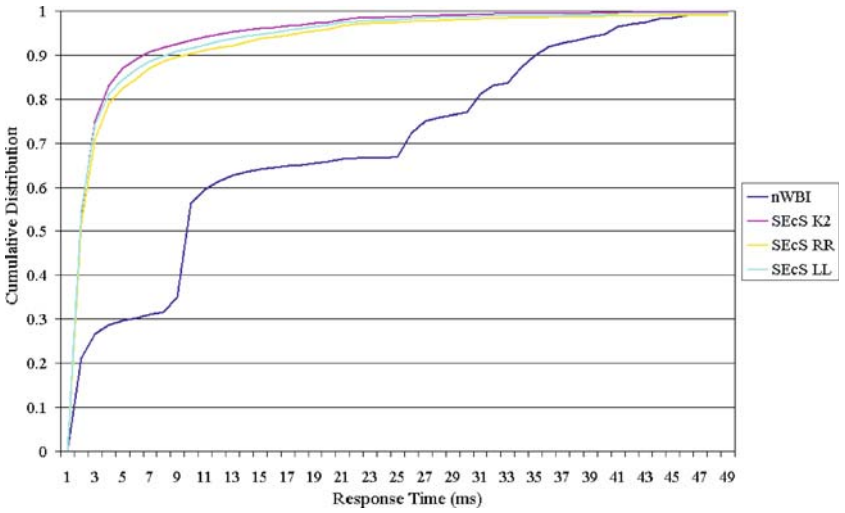


**Figure 10** Transcoding edge service. *HeavyLight* distribution (ms*10).

the HTTP requests, by handling only a small fraction of the overall traffic. Our goal, with this test, it to prove that SEcS K2 or SEcS LL outperform SEcS RR.

As we expected, from the plot in Figure 10 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 *n*WBI.

Our conclusions about this performance study 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 (*n*WBI) even when authentication and even distribution of requests on machines are given for free.

## 5. Conclusions

In this paper we have presented a framework to build and deploy Edge-computing Services, based on IBM Web Based Intermediaries and 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 added-value services, that also exhibit scalability, fault tolerance, high availability and robustness, and that, finally, allow programmable functionalities.

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. In addition SEcS allows an easy, "on-the-fly" and per-user configuration of services.

By using a workload model that includes the most recent results in Web Proxy workload characterization, we conducted experiments 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 [12] that is coming out from the most recent activities of IETF Working Group[4] for "Open Pluggable Edge Services" (OPES).

Their goal is to define an open standard that allows intermediaries to provide services on the data flow. In OPES, intermediaries can also employ local or remote servers (called "callout servers") in order to facilitate the efficient delivery of complex services. OPES ruleset are applied in order to choose which service to apply to the data flow. Communication between dispatchers and callout servers occurs through a protocol that must obey the requirements for OPES Callout Protocols specified in [23].

Our project proposes an architecture that will easily fit in the standard (once it becomes an accepted standard) and adds also a load balancing scheme and a remote communication protocol between MEGs to aim at an efficient and scalable solution for design and deployment of Edge-computing Services. To this end we have realized software components that are able to interact with other existing systems supporting: (*a*) OPES as framework for building *edge* services, (*b*) the ICAP protocol [39] as remote communication protocol and (*c*) the IRML language [22] (an XML-based language) to describe service-specific execution rules, that is, rules that specify when and how to execute intermediary services. Our future work, in fact, includes the development of an Edge services overlay network of intermediary entities distributed on wide area networks.

In fact, the architecture of our framework strongly relies on the assumptions that the communication within the Edge Proxy-Server (both FrontEnd and Remote) and between EPS and the Manager occurs in a local area network setting (typically, a cluster of workstations) where quick, reliable and high bandwidth communication is warranted. A different scenario would impose different architectural choices and, as a consequence, an extension of our results to the wide area network setting is not trivial and requires further research.

**Acknowledgments** We gratefully thank all the feedback provided by the members of IsisLab of our Department. In particular, fruitful discussions were done with Maria Barra and Alberto Negro. We also thank Michele Colajanni for several interesting comments on preliminary versions of our work. This research work was financially supported by the Italian FIRB 2001 project number RBNE01WEJT "WEB-MiNDS" (Wide-scalE, Broadband MIddleware for Network Distributed Services), http://web-minds.consorzio-cini.it/.

## References

1. Akamai, Inc. http://www.akamai.com
2. Almeida, V., Bestavros, A., Crovella, M., de Oliveira, A.: Characterizing Reference Locality in the WWW. In: Proceedings of the International Conference on Parallel and Distributed Information Systems (PDIS'96)., pp. 92–103. (December 1996)
3. Anderson, C., Domingos, P., Weld, D.: Personalizing Web Sites for Mobile Users. In: Proceedings of the $10^{th}$ International World Wide Web Conference. ACM, Hong Kong, (2001)

[4] See [12] for the description of the architecture and the Web site http://standards.nortelnetworks.com/opes for the overall activities of this WG.

4. Andreolini, M., Cardellini, V., Colajanni, M.: Benchmarking models and tools for distributed web-server systems. In: Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures, pp. 208–235. Springer, Berlin Heidelberg New York (2002)
5. Ardon, S., Gunningberg, P., LandFeldt, B., Ismailov, Y., Portmann, M., Seneviratne, A.: Mobile Aware Server Architecture: A distributed proxy architecture for content adaptation. In: INET 2001 Proceedings. ISOC (June 2001)
6. Ardon, S., Gunningberg, P., LandFeldt, B., Ismailov, M.P.Y., Seneviratne, A.: MARCH: a distributed content adaptation architecture. International Journal of Communication Systems, Special Issue: Wireless Access to the Global Internet: Mobile Radio Networks and Satellite Systems. **16**(1), (2003)
7. Arlitt, M.: A performance study of Web servers. Master's thesis, University of Saskatchevan (1996)
8. Arlitt, M.F., Williamson, C.L.: Internet Web servers: workload characterization and performance implications. IEEE/ACM Trans. Netw. **5**(5), 631–645 (1997)
9. Arlitt, M., Friedrich, R., Jin, T.: Workload characterization of a Web proxy in a cable modem environment. SIGMETRICS Perform. Eval. Rev. **27**(2), 25–36 (1999)
10. AT&T Labs-Research. iProxy: a Programmable Proxy. http://www.research.att.com/sw/tools/iproxy/
11. Balachander, K., Rexford, J.: Web Protocol and Practice. HTTP/1.1, Networking Protocol, Caching, and Traffic Measurements (July 2001)
12. Barbir, A., Chen, R., Hofmann, M., Orman, H., Penno, R.: An Architecture for Open Pluggable Edge services (OPES). http://www.ietf.org/internet-drafts/draft-ietf-opes-architecture-04.txt (June 11 2003)
13. Barbir, A., Burger, E., Chen, R., McHenry, S., Orman, H., Penno, R.: Open Pluggable Edge services (OPES) Use Cases and Deployment Scenarios (April 2004). http://www.ietf.org/rfc/rfc3752.txt
14. Barford, P., Crovella, M.: Generating representative Web workloads for network and server performance evaluation. In: Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, pp. 151–160. ACM (1998)
15. Barra, M., Negro, A., Scarano, V.: Distributed Systems for Group Adaptivity on the Web. In: Proceedings of International Conference on Adaptive Hypermedia and Adaptive Web-based Systems (AH 2000). ACM (August 2000)
16. Barra, M., Maglio, P., Negro, A., Scarano, V.: GAS: Group Adaptive System. In: Proceedings of International Conference on Adaptive Hypermedia and Adaptive Web-based Systems (AH 2002). ACM (May 2002)
17. Barra, M., Grieco, R., Malandrino, D., Negro, A., Scarano, V.: TextToSpeech: a heavy-weight Edge computing Service. In: Poster Proc. of $12^{th}$ International World Wide Web Conference. ACM (May 2003)
18. Barrett, R., Maglio, P.P.: Adaptive Communities and Web Places. In: Proceedings of $2^{th}$ Workshop on Adaptive Hypertext and Hypermedia, HYPERTEXT 98. ACM, Pittsburgh (USA) (1998a)
19. Barrett, R., Maglio, P.P.: Intermediaries: new places for producing and manipulating web content. Comput. Netw. ISDN Syst. **30**(4), 509–518 (1998b)
20. Barrett, R., Maglio P.P.: Intermediaries: an approach to manipulating information streams. IBM Syst. J. **38**(4), 629–641 (1999a)
21. Barrett, R., Maglio, P.P.: WebPlaces: Adding people to the Web. In: Proceedings of $8^{th}$ International World Wide Web Conference. ACM, Toronto (Canada) (1999b)
22. Beck, A.: IRML: A Rule Specification Language for Intermediary Services. Internet Draft
23. Beck, A., Hofmann, M., Orman, H., Penno, R., Terzis, A.: Requirements for OPES Callout Protocols An Architecture for Open Pluggable Edge Services (OPES). (August 2nd 2002). http://www.faqs.org/ftp/internet-drafts/draft-ietf-opes-protocol-reqs-02.txt
24. Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S.: Web caching and Zipf-like Distributions: Evidence and Implications. In: Proceedings of IEEE Infocom Conference, pp. 126–134. IEEE Computer Society (March 1999)
25. Brewer, E., Katz, R., Amir, E., Balakrishnan, H., Chawathe, Y., Fox, A., Gribble, S., Hode, T., Nguyen, G., Padmanabhan, V., Stemm, M., Seshan, S., Henderson, T.: A Network Architecture for Heterogeneous Mobile Computing. In IEEE Personal Communication Magazine **5**(5), 8–24 (October 1998)
26. Brisco, T.: Dns Support for Load Balancing. Technical report, Network Working Group, (April 1995). Technical Report RFC 1794

27. Busari, M.: ProWGen. Technical report, University of Saskatchewan (2000)
28. Calabrò, M.G., Malandrino, D., Scarano, V.: Group Recording of Web Navigation. In: Proceedings of the HYPERTEXT'03. ACM (August 2003)
29. Canali, C., Cardellini, V., Colajanni, M., Lancellotti, R., Yu, P.: Cooperative Architectures and Algorithms for Discovery and Transcoding of Multi-version Content. In: Proceedings of the $8^{th}$ Web Caching Workshop (WCW 2003) (September 2003)
30. CAST: eReader. http://www.cast.org/tools/teachingtoolsreader.html
31. Cisco systems: localdirector. http://www.cisco.com/warp/public/751/lodir/index.html
32. Cooper, I., Melve, I., Tomlinson, G.: Internet Web Replication and Caching Taxonomy (January 2001). RFC 3040
33. Crovella, A.B.M.: Self-similarity in World Wide Web Traffic: Evidence and Possible Cause. In: Proceedings of the DSIGMETRICS Conference on Measurements and Modeling of Computer Systems (May 1996)
34. Crovella, M., Lipsky, L.: Long-Lasting Transient Conditions in Simulations with Heavy-tailed Workloads. In: Proceedings of the 1997 Winter Simulation Conference (1997)
35. Dahlin, M.: Interpreting Stale Load Information. In: Proceedings of the 19th IEEE International Conference on Distributed Computing Systems, pp. 285. IEEE Computer Society (1999)
36. Dikaiakos, M.: Intermediary infrastructures for the world-wide web. Comput. Netw. ISDN Syst. **45**(4), 421–447 (July 2004)
37. Dikaiakos, M., Zeinalipour-Yiazti, D.: A distributed middleware infrastructure for personalized services. Technical Report TR-2001-4, University of Cyprus (December 2001a)
38. Dikaiakos, M., Zeinalipour-Yiazti, D.: WebRACE: A Distributed WWW Retrieval, Annotation, and Caching Engine. In: Proceedings of PADDA01: International Workshop on Performance-oriented Application Development for Distributed Architectures (April 2001b)
39. Eslon J., et al.: Internet Content Adaptation Protocol (ICAP). (April 2003). RFC 3507. http://www.ietf.org/rfc/rfc3507.txt
40. Fielding, R., Gettys, J., Mogul, J., Nielsen, H.F., Berners-Lee, T.: HTTP version 1.1, (January 1997). RFC 2616
41. Fox, A., Brewer, E.A.: Reducing WWW latency and bandwidth requirements by real-time distillation. In: Proceedings of the $5^{th}$ International World-Wide Web Conference. ACM (May 1996)
42. Fox, A., Gribble, S., Brewer, E.A., Amir, E.: Adapting to Network and Client Variability via On-demand Dynamic Distillation. In: the $7^{th}$ International Conference On Arch. Support for Prog. Lang. And Operating Systems. (ASPLOS-VII). ACM (October 1996)
43. Fox, A., Gribble, S., Chawathe, Y., Brewer, E.A., Gauthier P.: Cluster-based scalable network services. In: Proceedings of the sixteenth ACM symposium on Operating systems principles, pp. 78–91. ACM (1997)
44. Fox, A., Chawathe, Y., Brewer, E.A.: Adapting to network and client variation using active proxies: lessons and perspectives. IEEE Pers. Commun. **5**(4), 10–19 (1998)
45. Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Peach, P., Luotonen, A., Stewart, L.: HTTP Authentication: Basic and Digest Access Authentication, (June 1999). RFC 2617
46. Fry, M., Ghosh, A.: Application level active networking. Comput. Networks **31**(7), 655–667 (1999)
47. Gribble, S.D., Welsh, M., von Behren, R., Brewer, E.A., Culler, D., Borisov, N., Czerwinski, S., Gummadi, R., Hill, J., Joseph, R.H.K.A.: The Ninja architecture for robust internet-scale systems and services. Comput. Networks **35**(4), 473–497 (March 2001). http://ninja.cs.berkeley.edu/
48. Hori, M., Kondoh, G., Ono, K., Hirose, S., Singhal, S.: Annotation-Based Web Content Transcoding. In: Proceedings of the $9^{th}$ International World Wide Web Conference. ACM, Amsterdam (The Netherlands) (2000)
49. IBM.: Home Page Reader. http://www.austin.ibm.com/sns/hpr.html
50. IBM Websphere Transcoding Publisher.: http://www-3.ibm.com/software/webservers/transcoding
51. ImageMagick 5.5.7 (2003). http://www.imagemagick.org
52. JavaSoft.: The Java Remote Method Invocation (RMI) specification. Technical report, Sun Microsystems Incorporated (2001). http://java.sun.com/j2se/1.4/docs/guide/rmi/spec/
53. JMagick 5.5.6-0 (2003). http://www.yeo.id.au/jmagick/
54. KaRMI. Efficient RMI for Java. http://www.ipd.uka.de/JavaParty/KaRMI
55. Katz, R.H., Brewer, E.A., Amir, E., Balakrishnan, H., Fox, A., Gribble, S., Hodes, T., Jiang, D., Nguyen, G.T., Padmanabhan, V., Stemm, M.: The bay area research wireless access network (BARWAN). In: Proceedings of the 41st IEEE International Computer Conference, pp. 15. IEEE Computer Society (1996)

56. Luotonen, A., Altis, K.: World-wide web proxies. Comput. Netw. ISDN Syst. **27**(2), 147–154 (1994)
57. MacLarty, G., Fry, M.: Policy-based content delivery: an active network approach. Comput. Commun. **24**(2), 241–248 (2001)
58. Mahanti, A., Williamson, C., Eager, D.: Web Proxy Workload Characterization. Technical report, Department of Computer Science, University of Saskatchewan (February 1999)
59. Mahanti, A., Williamson, C., Eager, D.: Characterization of a web caching hierarchy. In: Mobile Networks and Applications (2000)
60. Mitzenmacher, M.: How useful is old information? In: Proceedings of the sixteenth annual ACM symposium on Principles of distributed computing, pp. 83–91. ACM (1997)
61. Mitzenmacher, M.: How useful is old information? IEEE Trans. Parallel Distrib. Syst. **11**(1), 6–20 (2000)
62. Mosberger, D., Jin, T.: httperf, A Tool for Measuring Web Server Performance
63. Netscape.: Navigator Proxy Auto-Configuration File Format, (March 1996). http://www.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html
64. Open Pluggable Edge services (OPES) Working Group. http://standards.nortelnetworks.com/opes/index.htm
65. Pitkow, J.E., Crovella, M.E.: Summary of WWW characterization. In: Proceedings of International World Wide Web Conference. ACM (1999)
66. RabbIT proxy. http://rabbit-proxy.sourceforge.net/
67. Rao, C., Chen, Y., Chang, D.-F., Chen, M.-F.: imobile: A proxy-based platform for mobile services. In: Proceedings of the First ACM Workshop on Wireless Mobile Internet (WMI 2001). ACM (2001)
68. Stardust.com. Content Networking and Edge Services: Leveraging the internet for profit (September 2001). http://www.speakerforums.com/Uploads/Stardust/pdfs/CDN_whitepaper.PDF
69. SuperProxy Script. http://naragw.sharp.co.jp/sps
70. Tanenbaum, A.S.: Modern operating systems (1992)
71. Web Based Intermediaries (WBI). http://www.almaden.ibm.com/cs/wbi/
72. Web Content Accessibility Guidelines 1.0, W3C Recommendation (May 1999). http://www.w3.org/TR/WCAG10/
73. Weinreich, H., Lamersdorf, W.: Concepts for improved visualization of Web link attributes. In: Proceedings of the $9^{th}$ International World Wide Web Conference. ACM, Amsterdam (The Netherlands), (2000)
74. Zipf, G.K.: Human Behaviour and the Principle of Least Effort. Addison Wesley (1999)