Distributed system for the support of mobile Web-based services[1]

Claudia Canali, Michele Colajanni, Riccardo Lancellotti
Dip. di Ingegneria dell'Informazione
Università di Modena e Reggio Emilia
{canali.claudia, colajanni, lancellotti.riccardo}@unimo.it

Philip S. Yu
IBM T. J. Watson Research Center
psyu@us.ibm.com

Technical Report 2007-10-12,
Dept. of Information Engineering,
University of Modena and Reggio Emilia

Oct. 2007

[1]This Report has been submitted for publication and will be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. No part of its text nor any illustration can be reproduced without written permission of the Authors.

Abstract

Personalized services are a key feature for the success of the next generation Web that is accessed by heterogeneous and mobile client devices. The need to provide high performance and to preserve user data privacy opens a novel dimension in the design of infrastructures and request dispatching algorithms to support personalized services for the Mobile Web. Performance issues are typically addressed by distributed architectures consisting of multiple nodes. Personalized services that are often based on sensitive user information may introduce constraints on the service location when the nodes of the distributed architecture do not provide the same level of security. In this paper, we propose an infrastructure and related dispatching algorithms that aim to combine performance and privacy requirements. The proposed scheme may efficiently support personalized services for the Mobile Web especially if compared with existing solutions that separately address performance and privacy issues. Our proposal guarantees that up to the 95% of the requests accessing sensitive user information are assigned to the most secure nodes with limited penalty consequences on the response time.

Index Terms: Personalized services, Privacy, Mobile Web, Dispatching algorithms, Distributed architectures, Performance evaluation.

1 Introduction

Mobile portable devices have already outnumbered traditional desktop computers and will mold the view of future Web-based services. This evolution is even more important since it is combined with the growing amount of personalized services offered to the users. Tailoring Web resources to the user preferences, context, location and to the capabilities of their heterogeneous client devices requires on-the-fly content generation, because a pre-generation of formats for any combination of devices, user needs and contexts is simply unfeasible.

From a computational point of view, personalized services for the Mobile Web typically require expensive tasks for the generation and adaptation of contents to client devices and user preferences [6, 7, 10]. For this reason, much interest of the research community has been focused on high performance systems consisting of multiple nodes to provide the user with efficient services. Besides computational cost, we should consider that most personalized services are based on information concerning the user [10] (the so-called *user profile*). The user profile may contain information about the user, such as his/her preferences, information on user click history, and lists of previous user interactions with the system. Furthermore, the user profile may contain information about the *user context*, such as user location and current activity.

Managing sensitive user information requires an infrastructure with a high security level, resulting in high maintenance costs, especially in the case of systems consisting of geographically distributed nodes. The trade-off of the solutions should be clear. The reduction of costs related to privacy management suggests to centralize services and user information on very few locations. On the other hand, performance goals suggest to spread services and information among geographically distributed nodes, with a consequent replication of sensitive data and an increase of the number of locations that must adhere to high security standards, such as the Payment Card Industry Data Security Standard (PCIDSS) [24].

Different approaches to solve this trade-off lead to a plethora of solutions for the deployment of Web systems supporting personalized services for the Mobile Web, ranging from fully centralized to peer-to-peer infrastructures. In this paper we propose an intermediate infrastructure that exploits a central component, typically a cluster, that we call *core node*. To improve the performance of the offered services, the central component is integrated with distributed *edge nodes* that are located close to the clients (Figure 1). Similar infrastructures have been proposed in distributed Web systems for traditional content generation and delivery [27, 26], but they represent a novelty in the Mobile Web scenario. In the proposed infrastructure we guarantee that the core node has the highest security standards, while we assume that it is more difficult or expensive to guarantee the same level of security to every edge node, that may be hosted or housed and not directly controlled. We also propose request dispatching algorithms that aim to solve the trade-off between performance and privacy. To the best of our knowledge this is the first paper that proposes infrastructures and algorithms that take into account both performance and privacy requirements, because performance and privacy issues have been typically addressed separately in literature (for example, see [31, 22, 17] for performance and [19, 12] for privacy).

Through a prototype namely Distributed Architecture for Mobile Web-based Services (DAMWS), we demonstrate that the performance- and privacy-aware infrastructure is suitable to deploy effi-

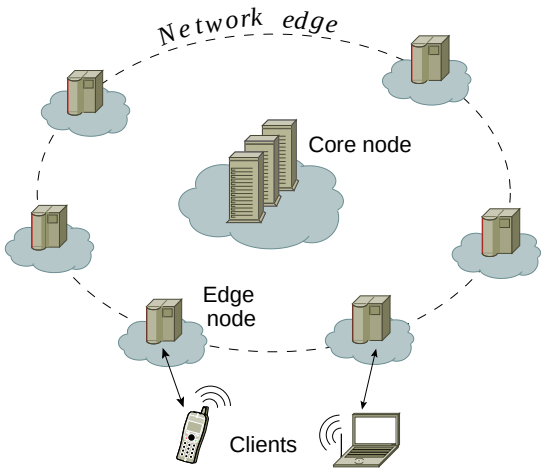


Figure 1: System supporting personalized services for the Mobile Web

cient and secure personalized services for the Mobile Web. Unlike the existing solutions that separately consider performance and privacy requirements, the proposed scheme guarantees the assignment of up to the 95% of the requests accessing sensitive user information to the most secure nodes with limited penalty consequences on the response time. On the other hand, performance-oriented solutions do not guarantee the highest security for an amount of requests that is 4 times higher if compared to our proposal. Furthermore, privacy-oriented solutions suffer from significant performance penalty, with response times almost doubled with respect to our proposal.

The remainder of the report is organized as follows. Section 2 describes the geographically distributed infrastructure considered in this paper. Section 3 presents the requests dispatching algorithms. Section 4 describes the prototype and the experimental testbed for the evaluation of the considered schemes. Section 5 compares experimental results by distinguishing performance and privacy. Section 6 discusses the related work and Section 7 concludes the paper with some final remarks.

2 Infrastructure overview

In this section we describe the system-level details of the proposed infrastructure supporting personalized services for the Mobile Web and implmented in the DAMWS prototype.

This infrastructure consists of a centralized core node integrated with geographically replicated edge nodes, as shown in Figure 1. The infrastructure follows the recent trend of modern systems that exploits servers on the network edge to replicate Web-based services (possibly including personalized services), as can be observed in recent literature [27, 22, 21] and in CDN solutions [15, 12]. The use of replicated edge nodes is a viable solution also because most new mobile devices require some intermediary to access the Mobile Web.

The infrastructure model, that is described in Figure 1, is detailed in Figure 2. The core node provides a three-tier Web system with a first tier of *HTTP servers*, a second tier of *application*

servers performing content generation and adaptation, and a third tier of *back-end servers* hosting the application data. The core node also maintains a *static resources* repository and the *user profile database*. The user profiles are maintained on the core node because it guarantees high security standards while the edge nodes may be hosted or housed in locations with lower levels of physical and logical security. Each edge node consists of a two-tiered Web system with a *front-end server* and an *application server* that carries out generation and adaptation functions. As shown in Figure 2, each edge node hosts a *local data* repository with a replica of the static resources located on the core node. Every update of the static resources is propagated from the core node through push-based caching mechanisms to guarantee Web data consistency. The local data repository may also store partial user profile information that is necessary to generate and adapt Web contents on the edge nodes for a specific request. User information is cached on the edge nodes just for the user session to preserve data privacy and avoid any consistency problem related to the replication and the update of the profiles, that may change frequently in modern Web systems [14].

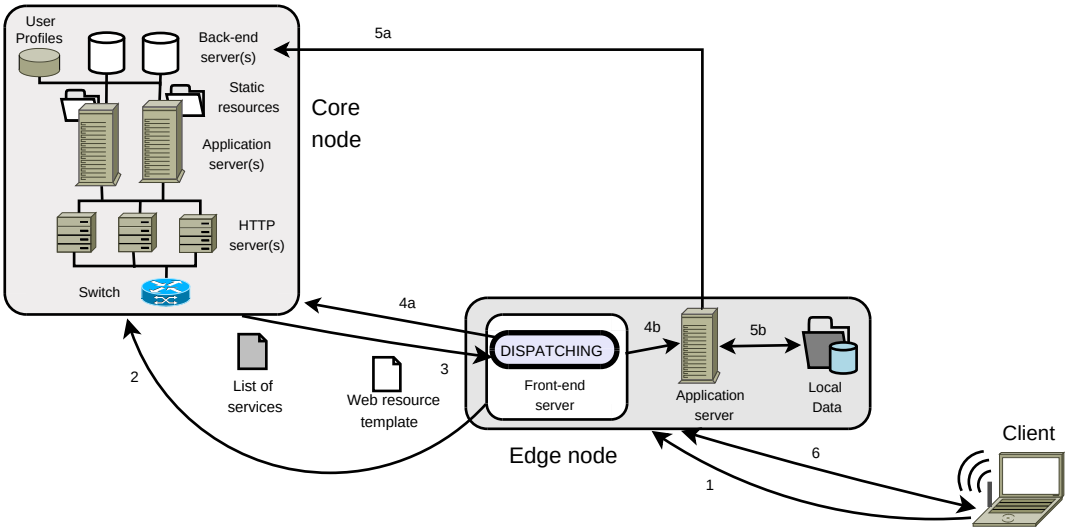


Figure 2: Detailed view of system supporting personalized services for the Mobile Web

The trade-off between preserving data privacy on the core node and improving performance by moving services on the edge nodes is addressed through novel request dispatching algorithms. The dispatching process is performed by the edge nodes, as shown in Figure 2, and exploits a fine-grained approach at the level of *Web resource components*. Each user interaction for a Web resource generates multiple requests for the associated *components*, where each of them may range from a simple fragment of text to a multimedia resource, such as an image or an audio/video stream [21], and may require a different personalized service. The portion of the user profile that is necessary for the personalized services determines the privacy requirements associated with that component. Throughout this paper we will consider three categories of privacy requirements. Components with *None* privacy requirements, that may be assigned to any node. Components with *Strong* privacy requirements (e.g., health information, credit card data), that must be processed on the core node since privacy is mandatory, and components with *Light* privacy requirements (e.g., user preferences, some information on user contexts), that should be assigned to the core node, but performance concerns may suggest to dispatch them to an edge node.

Figure 2 describes the steps to serve a client interaction for a Web resource. If the client request (Step 1) belongs to a new user session, the edge node contacts the core node (Step 2) to retrieve a template that enumerates the components of the Web resource and a list of the services that are required according with the user profile (Step 3). If the client request refers to an already established session, the edge node retrieves from the core node only the template of the resource. The load information about the core node is sent to the edge node along with the Web resource template. The edge node executes one of the dispatching algorithms described in Section 3 to assign the requests for Web resource components to the core node (Step 4a) or to the local edge node (Step 4b). For requests assigned to the edge node, the local application server retrieves the corresponding profile information directly from the back-end servers of the core node (Step 5a). In the same way, the application server obtains database information possibly required for the local generation process. The results of the queries to the back-end servers of the core node are cached by the edge nodes. The application server of the edge node may also interact with the local data repository (Step 5b). After the generation of all components, the Web resource is sent to the client by the edge node (Step 6).

3 Dispatching algorithms

Request dispatching is a key task for the deployment of efficient infrastructures supporting personalized services for the Mobile Web. We should consider that dispatching algorithms must be robust since they operate in a context where external and internal system conditions are subject to continuous changes [2, 16], including server load and network delays. The dispatching algorithms must also be able to handle highly heterogeneous workloads that may change in monthly, weekly or even daily patterns depending on user behavior and novel offered services. Let us analyze the information about the system and the client requests that a dispatching algorithm may access to distribute requests for Web resource components among the nodes. Table 1 shows the symbol used in the algorithms notation to represent these information.

For privacy concerns, dispatching algorithms may consider the security level of the distributed nodes and the privacy requirements of the Web resource components. We recall from Section 2 the three levels of privacy requirements: *Strong*, *Light* and *None*. Components with *Strong* privacy requirements must be dispatched to the core node; components with *Light* privacy requirements are preferably assigned to the core node; components with *None* privacy requirements may be dispatched to any node.

Performance is the other main issue that a dispatching algorithm should address. To this purpose, the algorithms may take into account performance-related information about the distributed Web system. As we have verified that most personalized services are CPU-bound operations, we consider the CPU utilization ρ of the application servers providing personalization as the most important performance-related index of the system status [1]. However, the main results of this paper are still valid with appropriate load index changes, if we consider a system where the generation of personalized contents is based for example on disk-bound operations.

In this section we present three dispatching algorithms, namely *Performance and Privacy*,

Performance-oriented and *Privacy-oriented*. The first algorithm represents an innovative request dispatching that aims to combine performance- and privacy-aware objectives, while the latter two alternatives separately address performance and privacy requirements.

Table 1: Algorithms notation

Identifier	Meaning
$\mathbf{R}$	List of Web resource components
$\mathbf{r}$	Web resource component
$\mathbf{P_N}$	List of Web resource components with *None* privacy requirements
$\mathbf{P_L}$	List of Web resource components with *Light* privacy requirements
$\mathbf{P_S}$	List of Web resource components with *Strong* privacy requirements
$\rho_E(t)$	CPU utilization of the edge node
$\rho_C(t)$	CPU utilization of the core node
$\mathbf{n_r}$	Dispatching solution for r

3.1 Performance and Privacy

The Performance and Privacy algorithm follows a load sharing strategy that is based on a maximum utilization threshold ρ^*. The privacy goal is subordinated to the load sharing constraint. To satisfy this constraint, the algorithm must estimate the contribution to the node utilization due to the dispatching decision $\rho_n^{\mathbf{R}}$. The algorithm initial assumption is that, when only privacy is considered in a dispatching decision, $\rho_n^{\mathbf{R}} = 1 - \rho^*$ for both edge and core nodes, so that the load sharing condition becomes $\rho_n(t) < \rho^*$.

If the utilizations of the edge and core nodes $\rho_E(t)$ and $\rho_C(t)$ are below the threshold ρ^*, the load sharing constraint is satisfied. If the utilization of a node is higher than ρ^*, the initial assumption on $\rho_n^{\mathbf{R}}$ is relaxed, and the algorithm dispatches more requests on the least loaded node even if this may have a negative impact on privacy results. For this reason, the algorithm determines the components R_C^N with *None* privacy requirements that are directed on the core node, and the components R_E^L with *Light* privacy requirements that will be served by the edge node (the privacy penalty comes from these latter assignments). Other details of the algorithm are described through the following cases.

1. If both edge and core nodes have a utilization below the threshold ρ^* (or in the opposite case where both nodes have a utilization beyond the threshold), the dispatching satisfies privacy requirements (lines 3–4 of Algorithm 1)

2. If only the edge node has a utilization $\rho_E(t)$ beyond the threshold ρ^*, the algorithm first dispatches the components with *Light* privacy requirements to the core node and then assigns R_C^N components with *None* privacy requirements to the core node (lines 6–14).

3. When only the core node has a utilization $\rho_C(t)$ beyond the threshold ρ^*, the behavior is similar to the previous case (lines 16–25), with the difference that an amount of components R_E^L with *Light* privacy requirements are assigned to the edge node to alleviate the core node load.

It is interesting to analyze the case when an excessive load on the edge node occurs (case 2). In this instance, the algorithm assigns R_C^N components with *None* privacy requirements to the core node. R_C^N increases as a function of the edge node CPU utilization $\rho_E(t)$. As shown by the representation of the formula in line 8 of the Algorithm 1 Figure 3, as the CPU utilization of the edge node increases beyond ρ^*, R_C^N grows linearly between 0 and $|P_N|$. We have considered other functions (e.g., linear, quadratic, and exponential), but all our preliminary experiments show that the performance results are scarcely affected by the specific function.

In the case of excessive load on the core node (case 3, line 17), the number of components R_E^L that are assigned to the edge node is computed through a similar function.

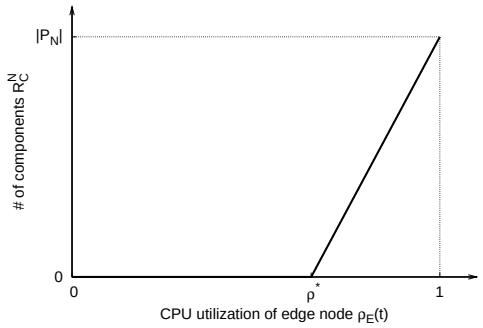


Figure 3: Value of R_C^N in Performance and Privacy algorithm as a function of $\rho_E(t)$

The computational complexity of the algorithm is $\mathcal{O}(n)$.

3.2 Performance-oriented algorithm

We carried out a preliminary comparison of the algorithms available in literature (e.g., round-robin, dynamically weighted round-robin [5]) with other algorithms designed by the authors with the goal of optimizing the user-perceived performance through an efficient use of the system computational resources. In this section we consider only the best performing solution(s) as an example of a *Performance-oriented* algorithm. In the considered algorithm, privacy-related constraints are not considered, except for the dispatching of requests for components with *Strong* privacy requirements, that must be assigned to the core node.

To achieve good performance, the Performance-oriented algorithm exploits a load sharing strategy considering also the computational cost of resources, that depends on the estimation of the service time of each Web resource component. Indeed we define the computational cost of each Web resource component as $\frac{1}{\mu_r T}$, where $\frac{1}{\mu_r}$ is the service time of the Web resource component r and T is the observation period used in the estimation of the CPU utilization. This algorithm aims to maximize the number of components processed on the edge node without exceeding its residual capacity. This algorithm tends to leave on the network edge the components with the lowest processing time, because for these components the network delay (reduced by the use of edge nodes) has a non-negligible contribution to the user-perceived response time.

Algorithm 1 Performance and Privacy

Require: $\mathbf{P_S}, \mathbf{P_L}, \mathbf{P_N}, \rho_E(t), \rho_C(t), \rho^*, R_M^*$
Ensure: $\{n_r\}$ (Performance and Privacy)

$n_r \Leftarrow$ "C", $\forall r \in \mathbf{P_S}$
if $(\rho_E(t) \leq \rho^*$ **and** $\rho_C(t) \leq \rho^*)$ **or** $(\rho_E(t) > \rho^*$ **and** $\rho_C(t) > \rho^*)$ **then**
    $n_r \Leftarrow$ "E", $\forall r \in \mathbf{P_N}$
    $n_r \Leftarrow$ "C", $\forall r \in \mathbf{P_L}$
else
    if $\rho_E(t) > \rho^*$ **then**
        $n_r \Leftarrow$ "C", $\forall r \in \mathbf{P_L}$
        $R_C^N \Leftarrow \lfloor |\mathbf{P_N}| \frac{\rho_E(t) - \rho^*}{1 - \rho^*} \rfloor$
        for all $r \in \mathbf{P_N}$ **do**
            if $R_C^N > 0$ **then**
                $n_r \Leftarrow$ "C"
                $R_C^N \Leftarrow R_C^N - 1$
            else
                $n_r \Leftarrow$ "E"
            end if
        end for
    else
        $n_r \Leftarrow$ "E", $\forall r \in \mathbf{P_N}$
        $R_E^L \Leftarrow \lfloor |\mathbf{P_L}| \frac{\rho_C(t) - \rho^*}{1 - \rho^*} \rfloor$
        for all $r \in \mathbf{P_L}$ **do**
            if $R_E^L > 0$ **then**
                $n_r \Leftarrow$"E"
                $R_E^L \Leftarrow R_E^L - 1$
            else
                $n_r \Leftarrow$"C"
            end if
        end for
    end if
end if
return $\{n_r\}$

The requests dispatching must solve a single knapsack optimization problem, where the knapsack has a capacity equal to the edge node residual capacity, that is $1 - \rho_E(t)$. Each object placed in the knapsack corresponds to a Web resource component with a volume equal to its contribution to the node utilization $\frac{1}{\mu_r T}$. We aim to maximize the number of objects in the knapsack, that is the number of Web resources dispatched on the edge nodes: $Obj = |\{r : n_r =\text{“E”}\}|$ subject to the bound of not exceeding the knapsack capacity. This means that we do not want to overload the edge node, that is,

$$\sum_{r:n_r=E} \frac{1}{\mu_{c_r} T} \leq 1 - \rho_E(t)$$

The optimization problem is solved by means of a greedy algorithm proposed by Martello and Toth [23]. The Web resource components are allocated on the edge node, starting from the requests with lower computational cost. For each dispatching decision we update the residual capacity of the edge node, which is represented through the variable S_E. When the residual capacity is exhausted, the remaining requests are dispatched to the core node.

The computational complexity of the algorithm is $\mathcal{O}(n \log n)$ due to the sorting operations.

Algorithm 2 Performance-oriented

Require: $\mathbf{P_S}, \mathbf{P_L}, \mathbf{P_N}, \mathbf{C}, \rho_E(t)$
Ensure: $\{n_r\}$ (Performance-oriented)
1: $n_r \Leftarrow$ “C”, $\forall r \in \mathbf{P_S}$
2: $A_E \Leftarrow 1 - \rho_E(t)$
3: **sort** $\mathbf{P_L} \cup \mathbf{P_N}$ by increasing C
4: **for all** $r \in \mathbf{P_L} \cup \mathbf{P_N}$ **do**
5: **if** $\rho_{c_r} < A_E$ **then**
6: $n_r \Leftarrow$ “E”
7: $A_E \Leftarrow A_E - \frac{1}{\mu_{c_r} T}$
8: **else**
9: $n_r \Leftarrow$ “C”
10: **end if**
11: **end for**
12: **return** $\{n_r\}$

3.3 Privacy-oriented algorithm

The *Privacy-oriented* algorithm aims to satisfy all privacy requirements of each Web resource component. In literature, there are not other examples of privacy-driven algorithms, hence we propose a specific algorithm that guarantees privacy requirements without caring of performance. The algorithm assigns every component with *Light* and *Strong* privacy requirements to the core node, and the components with *None* privacy requirements typically to the edge node. The computational complexity is $\mathcal{O}(n)$ because the core of the algorithm is a loop over each Web resource component.

It is worth to note that in scenarios where the majority of the components has *None* privacy requirements, an algorithm that assigns all these resources to the edge node may easily overload it. To avoid the risk of saturating the edge node capacities, we introduce a parameter R_E^* that represents the maximum number of components that can be dispatched on the edge node. In our implementation, we choose $R_E^* = \lceil 0.5 \cdot |\mathbf{R}| \rceil$ (line 2 of Algorithm 3). Hence, at most 50% of the

components are dispatched to an edge node (lines 4–6), while the other requests are assigned to the core node (line 8).

Algorithm 3 Privacy-oriented

Require: $\mathbf{P_S}, \mathbf{P_L}, \mathbf{P_N}$
Ensure: $\{n_r\}$ (Privacy-oriented)
1: $n_r \Leftarrow$ "C", $\forall r \in \mathbf{P_S}$
2: $R_E^* \Leftarrow \lceil 0.5(|\mathbf{P_S}| + |\mathbf{P_L}| + |\mathbf{P_N}|) \rceil$
3: **for all** $r \in \mathbf{P_L} \cup \mathbf{P_N}$ **do**
4: **if** $p_r =$ "None" & $R_E^* > 0$ **then**
5: $n_r \Leftarrow$ "E"
6: $R_E^M \Leftarrow R_E^M - 1$
7: **else**
8: $n_r \Leftarrow$ "C"
9: **end if**
10: **end for**
11: **return** $\{n_r\}$

4 Experimental testbed

The dispatching algorithms have been integrated into the DAMWS prototype supporting Mobile Web-based services that are accessed by fixed and mobile clients.

4.1 Services and workload

The prototype application is a personalized Web portal modeled after *my yahoo*, but the provided services are enriched to support innovative features, such as user mobility and user social networks. DAMWS generates and adapts content for three main Mobile Web-based services:

- **Banner insertion**. This service uses a database to associate banners to one or more keywords. It extracts from the database a list of banners according to the user interests, that are represented through keywords associated to the user or by keywords referred to other users marked as *friends* in the user social network. A set of banners is randomly selected from the list and inserted into the Web resource.

- **Aggregation of RSS feeds.** The portal Web resources are generated through the information collected from multiple blogs. The blog sources are explicitly provided by the user; alternatively, they represent blog entries selected on the basis of the social network stored in the user profile, so that friend's blogs are automatically added to the blog sources. The RSS feeds are downloaded off-line and periodically refreshed (the RSS feeds are stored as static Web content). The conversion from the RSS-XML code to HTML depends on the user profile and is carried out dynamically for every request.

- **Adaptation of Web content to the user device**. Both static HTML code and embedded images of Web resources are processed according to information on the client device stored

in the user profile or on the basis of the default profile. For example, a transcoding service can adapt images to the client display size and color depth. Furthermore, for a device with wireless connection the system can enable a data compression function to reduce the bandwidth consumption related to HTML and Javascript code download. The DAMWS implements this service through operations of string replacement, image transcoding based on the Imagemagick library [20] and stream compression based on the gzip algorithm.

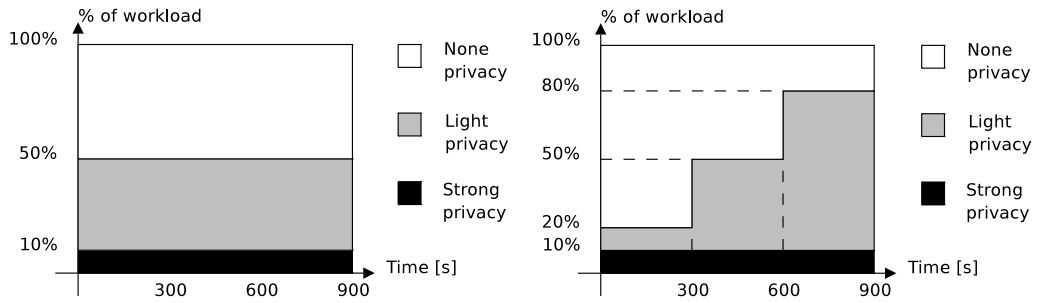
The three services are characterized by increasing computational costs that may involve service times of different orders of magnitude. A banner insertion typically requires few milliseconds, a feed aggregation task may be in the order of tens of milliseconds, while the adaptation of an embedded image may take from hundreds of milliseconds up to one second [7]. In our experiments, we consider that client requests are evenly distributed among the three offered services, respectively 33%, 34% and 33%.

We define two versions of each service provided to the users: *personalized* and *not personalized*. The *personalized* version of the service operates on the basis of the user profile and device characteristics. We consider that the personalized version of each service is characterized by *Light* privacy requirements. On the other hand, the *non-personalized* version of the services uses a *default profile* generated through off-line data mining on the most popular users setups. In this case, the non-personalized services share the same computational characteristics of their personalized version, but the services have *None* privacy requirements.

To provide a full spectrum of privacy requirements, we also define an additional service that is characterized by *Strong* privacy requirements. The service allows the user to visualize and edit every field of its stored profile, thus enabling a Web interface for the information stored in the user database. The service is characterized by an intermediate computational load, in the order of tens of milliseconds which is mainly due to the generation of a significant amount of HTML code.

We consider two workloads related to the privacy scenarios, namely *static* privacy and *dynamic* privacy. The static privacy workload is characterized by a distribution of privacy requirements that does not change during the experiment. Figure 4(a) shows the privacy requirements throughout the experiment: for the whole duration of the tests we have 10% of requests with *Strong* privacy requirements, 40% of requests with *Light* privacy requirements and 50% of requests with *None* privacy requirements. The dynamic privacy scenario is characterized by an average amount of privacy requirements that corresponds to that of the static privacy workload, with the main difference that the amount of components with *Light* privacy requirements changes over time. As shown in Figure 4(b), the components with *Light* privacy grow throughout the experiment from 10% to 70% in a staircase pattern with three steps (10%, 40% and 70%), while the components with *None* privacy shrinks from 80% to 20%. This workload aims to capture the Web variability, where the workload is subject to changes even in very short periods (e.g., in the order of few minutes) [2, 16].

To exercise our DAMWS prototype we use synthetically generated traces, since none of the existing benchmarking models (e.g., Spec-Web, TPC-W) includes features for Mobile Web-based services. The requests in the traces are divided into sessions, each related to a different user. Sessions are initiated at the rate of 5 sessions per second and are evenly distributed among the edge nodes. Each session is modeled by a bi-modal distribution, where 80% of the sessions are composed by only one resource request and the other sessions consist of an average (of an exponential



(a) *Static* privacy workload (b) *Dynamic* privacy workload

Figure 4: Privacy evolution

distribution) of ten resource requests [30]. Each Web resource consists of ten components on average, where the distribution is derived from [29]. Each resource component with *Light* or *None* privacy requires one of the three adaptation services (banner insertion, aggregation of RSS feeds, adaptation of Web content). The traces used in our experiments generate a workload intensity that is lower than the overall system capacity, because the goal of our experiments is to evaluate the effectiveness of the dispatching algorithms in a system which is correctly dimensioned with respect to the offered load.

4.2 System

The DAMWS prototype is implemented through open sources technologies for multi-tier Web systems on the core and the edge nodes. For our experiments we consider a distributed system consisting of one core node and multiple edge nodes. The core node is a cluster implemented as a three-tier Web system with HTTP-servers, application servers (which provide also the content generation and adaptation functions for the personalized services) and back-end servers. The front-end component of the core node is a Web switch implemented as an Apache Web server extended with a modified mod_proxy that distributes requests among the HTTP servers. The software handling dynamic requests and content adaptation on the application servers is implemented in Perl. The Perl scripts are processed by a second tier of Apache Web servers equipped with the mod_perl module to accelerate the execution of the Perl scripts. The back-end servers on the core node run a MySQL DBMS. The database is replicated on each back-end server and contains the data required by the offered services, for example, a list of banner images that is used by the banner insertion service. Furthermore, the DBMS handles the user profiles, where a list of attributes and preferences corresponds to each registered user.

Each edge node is composed by one application server with an additional front-end that implements the request dispatching task. The application server has the same functions of the application servers on the core node. Moreover, each application server hosts a query cache that aims to accelerate queries frequently issued to the back-end. The query cache is similar to the caching module proposed in other middleware systems [13, 25]. A BerkeleyDB file is used to store the data sets returned by each query. The SQL predicate of the query is used as the identifier of each data set. A time-to-live is associated with each entry of the query cache and is used to refresh stale data.

The prototype for the experiments is deployed on 16 physical servers with the same capacity, where 8 servers are used for the edge nodes. Each edge node consists of 1 front-end server for the request dispatching and 1 application server. The core node consists of 2 HTTP servers, 4 application servers, and 2 back-end servers. Incoming traffic is distributed to the HTTP servers by a Web switch.

Since the considered personalized Web-based services are characterized by a significant computational cost, the main system load lies on the application servers. The performance related information about the system load refers to the CPU utilization of the application servers, that are collected through the System Activity Reporter (SAR). For the core node that hosts multiple application servers, the CPU utilization value considered by the dispatching algorithms refers to the application server that will handle the client request. For a fair comparison among the dispatching algorithms, with no influence due to specific architectural choices, the core node and the set of edge nodes have the same number of application servers.

4.3 Network

We emulate WAN effects among the edge and the core nodes through the *netem* packet scheduler [18] that creates a virtual link between each edge the core node and between each client and edge node. The netem scheduler introduces packet delay and loss to mimic WAN effects. We achieve a full WAN emulation by adding also a token bucket filter scheduler that adds bandwidth limitation effects.

In our experiments we consider different network scenarios. For every scenario, we model the round trip network delay through a distribution obtained from real samples, as suggested in [18]. The client-to-edge round trip network delay is set to a mean value of 50 ms and the packet loss is set to 1%. In most experiments, we consider a mean network delay on the edge-to-core links equal to 10 ms, without bandwidth limitation. This case aims to emulate a well connected scenario where the edge nodes are located in Autonomous Systems that are close to the region hosting the core node. To evaluate the sensitivity of the dispatching algorithms to network delays, we also consider three scenarios where the mean value for the delay on the edge-to-core links is set to 20, 50, and 100 ms. The emulated WAN effects include also bandwidth limitation (10Mbit/s). Round trip delays, loss rates and bandwidth are consistent with the datasets from real measurements (e.g., [33]).

5 Performance and privacy evaluation

5.1 Evaluation metrics

The evaluation of the proposed dispatching algorithms takes into account performance and privacy results. As the main performance metric, we consider the Web resource response time on the client nodes, that is measured as the elapsed time between the client request and the arrival of all the components of a Web resource at the client. The resource response times are aggregated for the evaluation of cumulative distributions, because average values are of little meaning in a context

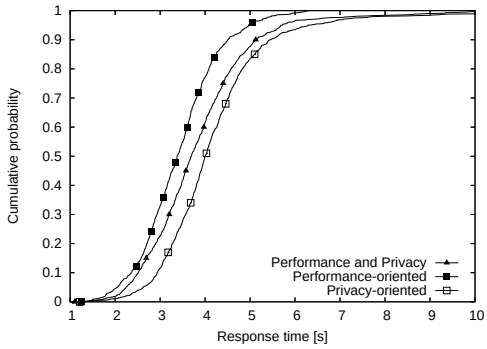


Figure 5: Cumulative distribution of response time (*static privacy* workload)

characterized by heavy-tailed distributions. For some analysis, we also consider the 90-percentile of the response time that is another common metric for the evaluation of the quality of service.

The privacy index considers how frequently an algorithm assigns the components with some privacy requirements to the edge nodes that are considered less secure than the core nodes. To this purpose, we evaluate the *dispatching mismatch*, that is defined as the amount of Web resource components with *Light* privacy requirements that are assigned to an edge node. These components should be processed on the core node, but performance reasons may force the dispatching algorithm to take sub-optimal decisions. It is worth to note that components with *Strong* privacy requirements do not contribute to the dispatching mismatch because all considered algorithms assign them to the core node. From the privacy point of view, we consider best the algorithms guaranteeing the lowest percentage of dispatching mismatch. In the first set of experiments, we disable the bound on the maximum allowed dispatching mismatch.

5.2 Static privacy workload

We initially consider the performance and privacy results of the three classes of dispatching algorithms in the unrealistic case of *static* privacy workload. Figure 5 shows the cumulative distribution of response time, while Table 2 presents the dispatching mismatch and the 90-percentile of response time throughout the experiment. From Figure 5, we observe that almost all algorithms achieve similar performance, as testified by the curves that run close to each other. The Privacy-oriented algorithm obtains a response time slightly higher if compared to the alternatives, but the performance penalty on the 90-percentile of response time with respect to the best performing algorithm is limited to 15%. This result suggests that, in the case of a stationary workload, a correct provisioning of the node resources of the underlying Web system guarantees that every algorithm can achieve adequate performance.

However, if we come to consider the privacy requirements we observe a clear difference among the algorithms. The third column of Table 2 summarizes the percentage of dispatching mismatch obtained by the algorithms for the static privacy workload. We observe that the Privacy-oriented algorithm, by definition, obtains a 0% mismatch for every workload. On the other hand, the Performance-oriented algorithm, which applies a privacy-blind dispatching, causes a mismatch

close to 30%.

Table 2: Performance and privacy evaluation for the static privacy scenario

Algorithm	90-percentile of response time [s]	Dispatching mismatch [%]
Performance & Privacy	5.08	4.6%
Performance-oriented	4.52	27.6%
Privacy-oriented	5.21	0%

The class of Performance and Privacy algorithms obtains dispatching mismatch percentages that, even if not reach the optimal value of the Privacy-oriented algorithm, are nearly 5 times lower than that of the Performance-oriented algorithm.

We can conclude that, in the (unrealistic) case of a stationary workload, where the content provider can provision the architecture with the right number of servers by taking into account the amount and type of requests, the Privacy-oriented algorithm is the best choice because it achieves perfect privacy results while preserving an acceptable level of performance. Even Performance and Privacy algorithms are a viable solution because each algorithm of this class achieves performance close to the lowest response time at the price of a little dispatching mismatch, that is never higher than 5%.

5.3 Dynamic privacy workload

The dynamic privacy workload is characterized by requests with privacy requirements that change throughout the experiment, as in Figure 4(b). This is a more realistic and more interesting case with respect to the static privacy scenario, because it captures the inherent variability of Web workloads and represents a more challenging scenario for the dispatching algorithms. Figure 6 shows that the performance of the algorithms significantly differ, with a big gap between the Performance-oriented and the Privacy-oriented algorithms. The Performance-oriented algorithm explicitly aims to optimize performance and avoids overload conditions. The consequence is a 90-percentile of the response time (4.57 s) that is half of that (9.12 s) characterizing the the Privacy-oriented algorithm that neglects performance goals. A summary of the main results is reported in the second column of Table 3, that presents the 90-percentile of the response time for all the considered algorithms. The Performance & Privacy algorithm achieves intermediate performance (6.19 s)

Let us now consider the amount of dispatching mismatch throughout the experiment. From the third column of Table 3, we observe that the main results confirm those of the static privacy workload: the Privacy-oriented algorithm achieves a perfect result in guaranteeing privacy requirements (even if this occurs at the expense of unacceptable performance penalties), while the Performance-oriented algorithm obtains poor results from a privacy point of view, because it is affected by a dispatching mismatch in the order of 50%. Performance and Privacy algorithms achieve intermediate results, with an amount of dispatching mismatch less than one fourth of that of the Performance-oriented algorithm.

The results for the dynamic privacy workload can be better explained if we consider separately

Table 3: Performance and privacy evaluation for the dynamic privacy scenario

Algorithm	90-percentile of response time [s]	Dispatching mismatch [%]
Performance & Privacy	6.19	10.8%
Performance-oriented	4.57	45.2%
Privacy-oriented	9.12	0%

the behavior of the algorithms in the three phases of the experiment, where the percentage of *Light* privacy requests is assumed to pass from 10% to 40% up to 70%. Figure 7 shows the 90-percentile of the response time (light gray column) and the dispatching mismatch (dark gray column) for each phase.

Focusing on the performance results, we observe that when the amount of components with *Light* privacy remains below 40%, all the algorithms provide similar performance that is, the 90-percentile of the response time is around 5 seconds. On the other hand, when the amount of components with *Light* privacy augments, the Privacy-oriented algorithm and the Performance-oriented algorithm must be discarded for opposite reasons. The problem for a Privacy-oriented algorithm with a doubled response time is twofold. First, uneven load sharing places a significant amount of resource components on the core node, thus saturating the capacity of this node. Second, a large amount of components dispatched to the core node adds a non-negligible latency in the response time because of edge-to-core network delays. On the other hand, the Performance-oriented algorithm, which is privacy-blind, achieves the same performance throughout all the experiment, but its dispatching mismatch grows proportionally to the amount of components with *Light* privacy. The class of Performance and Privacy algorithms obtains intermediate results between Performance-oriented and Privacy-oriented algorithms. Indeed, their ability to relax privacy constraints is fundamental to preserve an adequate level of performance during the increment of privacy requirements in the third phase. Moreover, this class of algorithms is effective in preserving a moderate amount of mismatch for the first two phases. The last phase of the staircase pattern is the main contribution to the mismatch percentage shown in Table 3 because the performance-awareness of the Performance and Privacy algorithms explicitly increases the amount of mismatch to preserve an adequate level of performance.

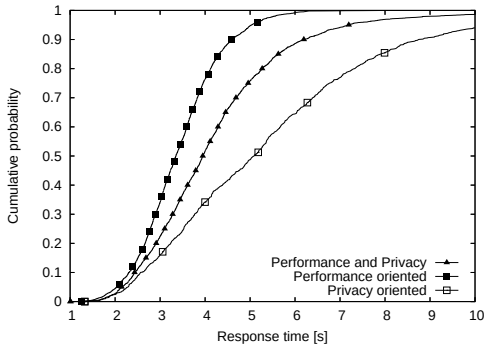


Figure 6: Cumulative distribution of response time (*dynamic* privacy workload)

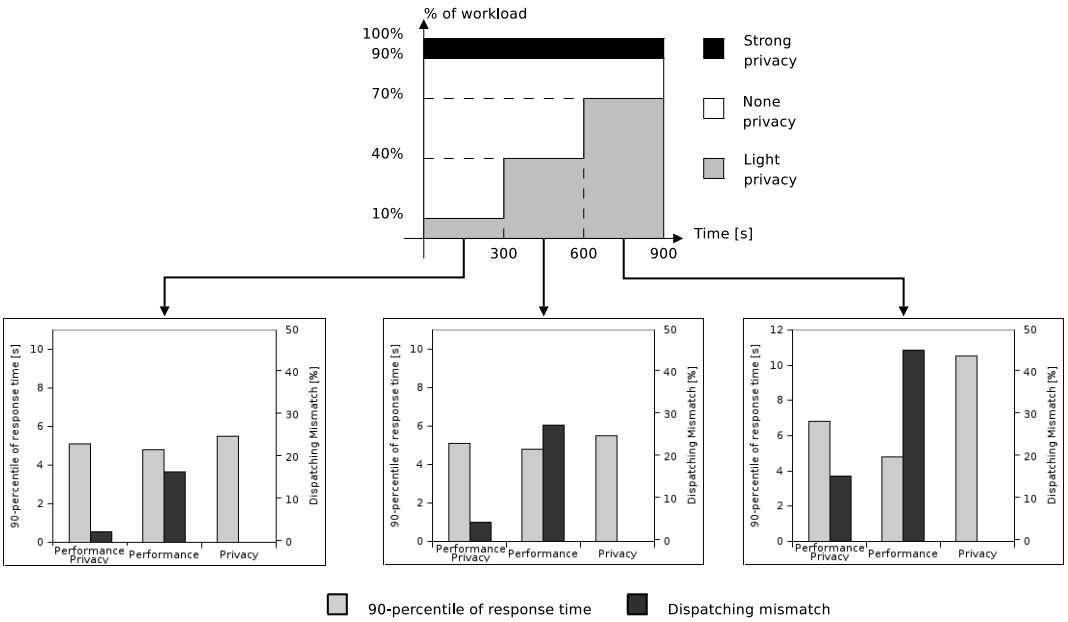


Figure 7: Response times for dynamic privacy workload

The experimental results allow us to draw multiple conclusions. In a Web context where the workload is subject to dynamic changes, any static choice in the number and type of nodes prevents a perfect combination of performance and privacy objectives and request dispatching plays a fundamental role. In particular it is important that dispatching algorithms to support Mobile Web-based services take into account both performance and privacy goals. When the workload is variable we can immediately exclude as valid dispatching algorithms the Performance-oriented and the Privacy-oriented classes because of their excessive dispatching mismatch and response time, respectively. On the other hand, our experiments show that the new class of Performance and Privacy algorithms may successfully combine performance and privacy objectives.

5.4 Sensitivity to Wide Area Network effects

Whenever the Web resource components are processed on the core node, the response time is increased because of the network delays on the link between the edge and the core node. Hence, it is interesting to evaluate the sensitivity of the dispatching algorithms to WAN effects. We take into account the impact on performance of different network delays on the links between the edge and the core nodes because, whenever a Web resource component is processed on the core node, it has to pay an additional delay with respect to a component that is processed on the edge node. For this analysis we refer to the dynamic privacy scenario. Figure 8 presents the 90-percentile of the response time achieved by the proposed infrastructure as a function of the mean network delay on the edge-to-core links. The three curves refer to the dispatching algorithms described in Section 3.

The curves confirms the performance comparison of the algorithms implemented on the proposed infrastructure. For every network delay, the Performance-oriented algorithm has the lowest

response time, while the Privacy-oriented algorithm achieves the highest response times. The proposed Performance and Privacy algorithm gets intermediate performance between the other two alternatives, as expected.

A further important result is the effectiveness of the proposed infrastructure when associated with the Performance and Privacy algorithm. Even if the provided services are computationally expensive, the response time remains in the order of 6-7 seconds for an edge-to-core for every considered delay, which may be considered acceptable for the users [9]. A further important result is the confirmation of the non negligible impact of network delay on performance, regardless of the dispatching algorithm. As the mean delay on the edge-to-core link passes from 10 ms to 100 ms, the performance degradation on the 90-percentile of the response time is in the order of 20% for every considered algorithm, as testified by the almost parallel curves of Figure 8.

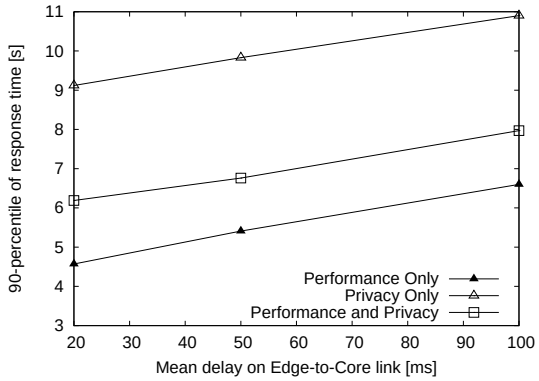


Figure 8: Effect of network delays

6 Related work

The Mobile Web has been recognized as a fundamental challenge by multiple authors [32, 28], but the importance of preserving privacy of user information while providing personalization services has been pointed out only by recent literature [8, 19]. Multiple distributed and parallel systems and related dispatching algorithms have been proposed with the main goal of improving performance, while scarce or no attention has been devoted to privacy requirements in request dispatching decisions. In many cases, privacy is not considered at all, while some systems adopt solutions with straightforward dispatching that strongly limit the possibility of distributing personalization tasks.

When the infrastructure is based on a cluster Web system [5], privacy requirements are not taken into account because this architecture may guarantee high levels of security for any of its nodes.

Other infrastructures for the delivery of Web content that are more similar to our proposal adopt a geographical distribution of nodes, with multi-clusters or single-cluster integrated with geographic replicated servers or even CDNs [26, 22]. These systems rely on request dispatching algorithms that are based on factors like network and geographic proximity, network link status or

server load. We introduce in this field a novel concept of algorithm for request dispatching that addresses both performance and privacy issues.

The need of taking into account privacy in the design of scalable distributed architectures is confirmed by P3P (Platform for Privacy Preferences) [11]. It is a W3C proposal that suggests a mechanism for Web sites to encode their privacy policies in a standardized format that can be easily retrieved and interpreted by user agents. However, these studies are more tailored to a server-side approach for the generation of personalized Web content rather than to the intermediary-based model for Web content adaptation considered in this paper. Some recent studies, that replicate the application logic on the edge servers [12, 27], propose the specialization of a subset of servers to handle a specific set of services. This mechanism may be used to preserve data privacy, but it has the drawback of limiting the flexibility of the infrastructure because only some nodes may provide personalization services. On the other hand, we propose a more flexible infrastructure demonstrating that privacy and performance requirements may be pursued together.

7 Conclusions

In this paper, we propose a distributed infrastructure to support personalized services for the Mobile Web and a related set of request dispatching algorithms. The infrastructure, that is composed by a central core node and geographically distributed edge nodes, exploits an innovative dispatching algorithm that takes into account both performance and privacy issues in the service of client requests.

Our experiments demonstrate that our proposal can successfully combine performance and privacy requirements in providing personalized services for the Mobile Web. For every considered workload and network scenario, the proposed scheme preserves from 89% to 95% of the requests privacy requirements, with a limited penalization on the response time if compared to solutions that just aim to optimize the user-perceived performance.

References

[1] T. Abdelzaher, K. Shin, and N. Bhatti. Performance guarantees for Web server End-Systems: a control-theoretical approach. *IEEE Trans. on Parallel and Distributed Systems*, 13:80–96, Jan. 2002.

[2] V. A. Almeida and D. A. Menasce. Capacity planning: An essential tool for managing Web services. *IT Professional*, 4:33–38, July 2002.

[3] M. Aron, P. Druschel, and W. Zwaenepoel. Efficient support for P-HTTP in cluster-based Web servers. In *Proc. of the USENIX 1999 Annual Technical Conf.*, Monterey, CA, Jun. 1999.

[4] M. Barra, R. Grieco, D. Malandrino, A. Negro, and V. Scarano. Texttospeech: a heavy-weight edge service. In *Proc.of 12th WWW Conf.*, Budapest, HU, 2003.

[5] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu. The state of the art in locally distributed Web-server systems. *ACM Computing Surveys*, 34(2):263–311, 2002.

[6] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, and W. Zwaenepoel. Performance comparison of middleware architectures for generating dynamic Web content. In *Proc. of 4th Middleware Conf.*, Jun 2003.

[7] S. Chandra. Content adaptation and transcoding. *Practical Handbook of Internet Computing*, 2004. (Munindar P. Singh ed.), Chapman Hall & CRC Press.

[8] R. K. Chellappa and R. G. Sin. Personalization versus privacy: An empirical examination of the online consumer's dilemma. *Information Technology and Management*, 6(2-3), April 2005.

[9] W. Chiu. Design pages for performance. *IBM HVWS*, 2001.

[10] M. Colajanni, R. Lancellotti, and P. Yu. Distributed architectures for Web content adaptation and delivery. *Web Content Delivery*, 2005. (Tang, Xu, Chanson eds.), Springer.

[11] L. Cranor. *Web Privacy with P3P*. O'Reilly, 2002.

[12] A. Davis, J. Parikh, and W. E. Weihl. EdgeComputing: extending enterprise applications to the edge of the Internet. In *WWW Alt. '04: Proc.of the 13th Int'l World Wide Web Conf.*, pages 180–187, 2004.

[13] L. Degenaro, A. Iyengar, I. Lipkind, and I. Rouvellou. A middleware system which intelligently caches query results. In *Middleware'00: IFIP/ACM Int'l Conf. on Distributed Systems Platforms*, pages 24–44, New York, USA, 2000.

[14] M. Eiriniaki and M. Vazirgiannis. Web mining for Web personalization. *ACM Trans. on Internet Technology*, 3(1), 2003.

[15] Edge Side Includes, 2002. `http://www.esi.org/`.

[16] S. Floyd and V. Paxson. Difficulties in simulating the Internet. *IEEE/ACM Trans. on Networking*, 9(4):392–403, 2001.

[17] L. Gao, M. Dahlin, A. Nayate, J. Zheng, and A. Iyengar. Application specific data replication for edge services. In *Proc. of 12th WWW Conf.*, Budapest, HU, 2003.

[18] S. Hemminger. netem: Network emulator, 2004. – http://developer.osdl.org/shemminger/netem/.

[19] R. Hull, B. Kumar, D. Lieuwen, P. F. Patel-Schneider, A. Sahuguet, S. Varadarajan, and A. Vyas. Enabling context-aware and privacy-conscious user data sharing. In *Proc.of 2004 IEEE Int'l Conf. on Mobile Data Management (MDM'04)*, 2004.

[20] ImageMagick, 2006. `http://www.imagemagick.org/`.

[21] A. Iyengar, L. Ramaswamy, and B. Schroeder. Techniques for efficiently serving and caching dynamic Web content. *Web Content Delivery*, 2005. (Tang, Xu, Chanson eds.), Springer.

[22] M. Karlsson. Replica placement and request routing. *Web content delivery*, 2005. (Tang, Xu, Chanson eds.), Springer.

[23] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[24] Master Card Int'l. *Payment Card Industry Data Security Standard*, Jan. 2005.

[25] C. Olston, A. Manjhi, C. Garrod, A. Ailamaki, B. Maggs, and T. Mowry. A scalability service for dynamic Web applications. In *Proc. of the 2nd Biennal Conf. on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, Jan 2005.

[26] M. Rabinovich and O. Spatscheck. *Web Caching and Replication*. Addison Wesley, 2002.

[27] M. Rabinovich, Z. Xiao, and A. Aggarwal. Computing on the edge: A platform for replicating Internet applications. In *Proc. of 8th Int'l Workshop on Web Content and Distribution*, Hawthorne, NY, Sept. 2003.

[28] D. Saha and A. Mukherjee. Pervasive computing: A paradigm for the 21st century. *IEEE Computer*, 36(3), Mar. 2003.

[29] W. Shi, E. Collins, and V. Karamcheti. Modeling object characteristics of dynamic web content. *Journal of Parallel and Distributed Computing*, 63(10):963–980, Oct. 2003.

[30] W. Shi, R. Wright, E. Collins, and V. Karamcheti. Workload characterization of a personalized Web site and its implications for dynamic content caching. In *Proc.of 7th WCW Workshop*, Boulder, Co, Aug. 2002.

[31] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. van Steen. Replication for Web hosting systems. *ACM Computing Surveys*, 36(3):291–334, 2004.

[32] G. C. Vanderheiden. Anywhere, anytime (+ anyone) access to the next-generation www. *Computer Networks and ISDN Systems*, 8(13), Sep. 1997.

[33] R. Zhang, C. Hu, X. Lin, and S. Fahmy. A hierarchical approach to Internet distance prediction. In *Proc. of the 26th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS'06)*, Washington, DC, USA, Jul. 2006.