Self-Adaptive Utility-Based Web Session Management

Nicolas Poggi [a 1] Toni Moreno [b 2] Josep Lluis Berral [a]
Ricard Gavaldà [c] Jordi Torres [a,d]

[a] *Computer Architecture Department, Technical University of Catalonia, Barcelona, Spain*

[b] *Department of Management, Technical University of Catalonia, Barcelona, Spain*

[c] *Department of Software, LARCA Research Group, Technical University of Catalonia, Barcelona, Spain*

[d] *Barcelona Supercomputing Center, Barcelona, Spain*

Abstract

In the Internet, where millions of users are a click away from your site, being able to dynamically classify the workload in real time, and predict its short term behavior, is crucial for proper self-management and business efficiency. As workloads vary significantly according to current time of day, season, promotions and linking, it becomes impractical for some ecommerce sites to keep overdimensioned infrastructures to accommodate the whole load. When server resources are exceeded, session-based admission control systems allow maintaining a high throughput in terms of properly finished sessions and QoS for a limited number of sessions; however, by denying access to excess users, the website looses potential customers.

In the present study we describe the architecture of AUGURES, a system that learns to predict Web user's intentions for visiting the site as well its resource usage. Predictions are made from information known at the time of their first request and later from navigational clicks. For this purpose we use machine learning techniques and Markov-chain models. The system uses these predictions to automatically shape QoS for the most profitable sessions, predict short-term resource needs, and dynamically provision servers according to the expected revenue and the cost to serve it. We test the AUGURES prototype on access logs from a high-traffic, online travel agency, obtaining promising results.

Key words: Utility computing, resource management, ecommerce, autonomic computing, Web mining, machine learning.

[1] Contact author. E-mail: `npoggi@ac.upc.edu`
[2] Currently at the Department of Operations and Information Management of The Wharton School, University of Pennsylvania

1 Introduction

1.1 The Current Web Scenario

According to the Internet World Stats [19], there are over 1 billion Internet users, and when online, they are a click away from any site. Certain events such as news, world events, or promotions announced in mass media or in the Web can cause a flock of users to visit related websites, creating peak loads in minutes. Furthermore, current Web applications rely on technologies such as XML-based Web services for B2B communication, SSL for security, media streaming, and AJAX for interactivity. While these technologies enhance users' experience and privacy, they also increase the demand for CPU and resources on the server side. As Web applications become more resource-intensive and the number of potential visitors increases, system overload incidence is also growing along.

Scaling the infrastructure of a website might not be simple; for cost reasons, scalability problems, or because some peaks are infrequent, websites may not be able to adapt rapidly in hardware to user fluctuations. When a server is overloaded, it will typically serve no connection, as connections compete for resources. System administrators might get warnings from resource-monitoring services, but in general they get to the problem when the situation has already occurred, and controlling the arrival of users is out of their reach. To address this issue, session-based admission control systems [3,6] are used to keep a high throughput in terms of properly finished sessions and QoS for a limited number of sessions. However, by denying access to excess users, the website loses potential revenue from customers.

In this study we propose a new approach to this problem which uses learning techniques applied to past data, to create a model for anonymous Web user behavior in a real, complex website that does experience the overload situations mentioned above. The model is then used to support decisions regarding the allocation of the available resources, based on a utility-related metrics. The learning phase captures the selected features in a model according to a utility goal. As a proof of concept, we have selected the features that make a customer more likely to make a purchase, and therefore more attractive — from the point of view of maximizing revenues — to keep in the system in the case of a severe overload. In most of the article, we take as a metric the number of completed sessions that end in purchase; Section 6 describes other possible metrics and applications. In this sense, we are proposing a per-user adaptive utility based session management.

We describe the architecture of the prototype we have developed, named AU-

GURES, and the simulation experiments we have performed on weblogs from the servers of a high-traffic online travel agency, Atrapalo.com. The experiments indicate that using AUGURES to prioritize customer sessions can lead to increased revenue in at least two situations: one, when overload situations occur; that is, the incoming transaction load exceeds the site's capacity and some sessions will have to be queued, redirected to a static site, or dropped; for this study, these should be mostly non-buying sessions, while we try to admit most buying ones. The second scenario is that in which keeping a server running has a quantifiable cost; in this case, one could try to group buying sessions on a small number of servers, possibly shutting down those other servers that would produce little or no revenue.

1.2 Motivation

Our preliminary experiments [13] showed that the users' intentions for visiting a site can be predicted to some extent from their navigation clicks, results of previous visits, and other session information. In this context, a revenue-based admission control policy can help to avoid revenue loss due to randomly delaying or dropping excess connections. Defining admission policies based on information generated from user behavior models can contribute to devising cost-effective infrastructures, and seems to have promising applications for resource management for medium to large Web infrastructures.

1.3 Our Methodology

In this paper we present a method for learning, from the analysis of session logs, how to assign priorities to customers according to some metric – in this study, to their purchasing probability in the current session. Our approach consists in using the Web server log files to develop learning models that make predictions about each class of user future behavior, with the objective of assigning a priority value to every customer based on the expected revenue that s/he will generate, which in our case essentially depends on whether s/he will make a purchase. Our learning methods combines static information (time of access, URL, session ID, referer, among others) and dynamic information (the Web graph of the path followed by the user), in order to make predictions for each incoming Web request.

2 Related Work

In the context of Web workload analysis, there are few published studies based on real e-commerce data, mainly because companies consider HTTP logs as sensitive data. Moreover, most works are based on static content sites, where the studied factors were mainly: file size distributions, which tend to follow a Pareto distribution [1]; and file popularity following Zipf's Law [1,8]. Also, works such as [11] have studied logs from real and simulated auction sites and bookstores; there are no studies that we know about which are concerned with intermediary sites, like the one studied here, where most of the information comes from B2B providers and which can potentially have a different behavior.

Recent works on Web user behavior prediction and profiling [2,7,12,17] have focused on Web caching or prefetching of Web documents, to reduce latency of served pages and improve the cache hit rate. Another studied approach is to model users for link prediction generating navigational tours and next-link suggestions to users. The mentioned approaches are best suited for large and mostly static Web pages, where users navigate through a vast amount of information such as an encyclopedia. Prefetching a dynamic page that includes database transactions might be too costly in the case of a miss or user exit. Other authors [2,14] focus on dynamic content adaptation, where the page adapts to the type of user; it could include images, colors and even products or links. Our approach could be applicable for dynamic content adaptation too, although in this study we are focusing on resource management.

Path analysis [5,15] and Customer Behavior Model Graphs (CBMG) such as [10] are similar to the dynamic part of our approach — we use the closely related Markov chain model. Menascé et al. [10] propose to build the CBMG using the k-means clustering algorithm, creating a probability matrix for the possible path transitions from a state. What we try to accomplish in this paper is *not* predicting what *the next* click will be; rather, we want to foresee the user's ultimate intentions for visiting the site, and in particular whether s/he will eventually buy.

Session-based admission control has been widely studied [3,6,16]; the work presented here is an extension to these aproaches. Related works on resource management, i.e. by Littman et al. [9] uses Naïve-Bayes for cost classification and a Markov chain feedback approach for failure remediation. Other works such as [4] also take into account costs and resource allocation; in contrast with previous approaches, in this paper we are focusing on the actual revenue that is lost by denying access to purchasing users, and not resource allocation costs.

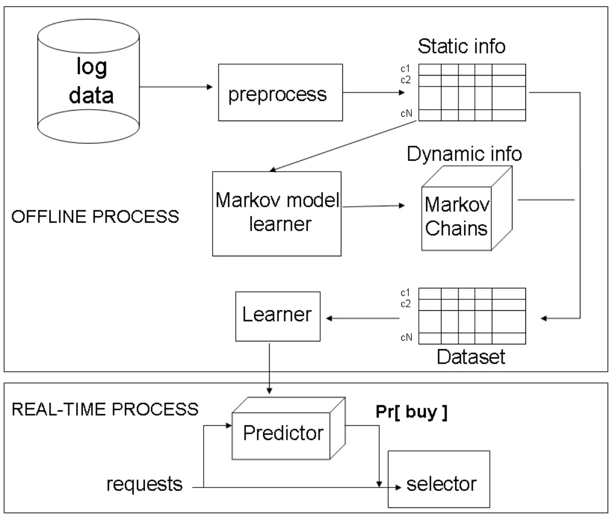

Fig. 1. AUGURES architecture

3 The AUGURES Prototype

In this section we describe the architecture of AUGURES, the prototype we have implemented to perform the experiments. AUGURES currently has two subsystems: an offline component (the *learner*) that takes the historical logfile and produces a predictive model or *predictor*, and a real-time component, the *selector*, implemented as a service that runs along with the session manager of the firewall. The selector analyses the incoming requests, runs them through the predictor, and outputs the priority along with other static information for the session. These two subsystems are presented graphically in Figure 1.

The input to the offline component is the logfile produced by the site's dynamic application. It contains non-ambiguous session and page actions (tags) as historical data, which is first cleaned and reorganized by a preprocessor. The preprocessor produces an intermediate file with one line for each transaction. These lines are largely computed independently from each other, so they do not contain information about the user navigational pattern; that is why we call the information in this file *static*. Next, this file is enriched with *dynamic* information reflecting the user navigation sequence, relating the different transactions of the same session. This is done by computing a Markov model for each value of the class, in our case buying and non-buying; the prediction of these models for each individual request is added as extra information to each line of the preprocessed file. Finally, this enriched dataset is passed to a learning module that produces a *predictor*, some mathematical function that, given a request, assigns it a buying probability. More details are given in the following subsections.

The real-time component, the selector, runs side-by-side with the session manager of the firewall. When an incoming HTTP/S request arrives, the selector reads the entry produced by the firewall, retrieves from a database existing information about the user and session (if any), and then evaluates the request through the predictor, the (offline built) model. The result is a predicted probability of purchase, which is written to the firewall's active session table along with other useful information such as: current load, server conditions, and enterprise policies. This information can then be used by the firewall (in ways outside the scope of this paper) to prioritize and even discontinue some of the sessions according to the current load and policies. We remark that the firewall is *not* a part of AUGURES: it is often a complex, and very sensitive, part of the infrastructure so we do not aim at replacing it. AUGURES, however, provides additional information to the firewall which helps it in taking informed decisions rather than blind or random ones.

In contrast to the selector, which has a real-time requirement, the offline component can be executed at scheduled intervals to rebuild the predictor (daily, weekly, etc.) at periods of low load, and even in an off-site machine. Therefore the requirements of speed and low memory use are not a limitation for this component, while the real-time part needs to be as efficient as possible. As future work we are considering running the learning module incrementally and in real time, so that the predictor is always as accurate as possible. In this case, the computational requirements of the learning method would also be of importance.

3.1 The Preprocessor: Generating Static Information

The goal of the preprocessor is two-fold: First, it should clean the logfile of static content i.e. images, CSS, javascript or other media files. It should also be cleaned of irrelevant and non-user-initiated transactions, such as AJAX autocomplete controls, background checks and offsite requests via Web services (B2B communication). The second goal is to add information that cannot be derived from the logfile only, such as background information on previous sessions and, if available, user details form the company's customer database.

The preprocessor reads the log and produces one output line for each input transaction, producing a dataset relevant to learning containing the following fields:

- Date and time.
- The tag, action performed by the page or non-ambiguous URL.
- Whether the user has already logged onto the system during this session.
- Whether the customer is a returning customer, retrieved from cookies or matching IP address.
- Whether the customer has purchased in the past, and if so how far back.

- Session length so far, in number of transactions (clicks).
- The referer tag, the tag of the previously visited page; this is an external page for the first click of each session.
- The *class* assigned to this session, that is, what the “correct” prediction should be for this log entry. In our case, there are two class values: buyer and non-buyer.

Note that all fields except for the class can be computed from information in the previous entries of the log, or from separately stored information. The class, however, can only be computed by looking *forward* in the same session, and checking whether it contains any tag indicating purchase. Clearly, this is not possible in the online process, since this information is precisely what we are trying to predict. Thus, the class can only be computed in datasets with past information, those used for offline learning.

3.2 Generating Dynamic Information

We use the information obtained from the user’s navigation sequence as the *dynamic information* of the session; it is the sequence of URLs followed by the user. Unfortunately, most machine learning algorithms are not well adapted to dealing with variables that are themselves sequences. In AUGURES we propose to use high-order Markov chains to address this issue.

A Markov chain describes a probability distribution on the set of all finite paths along a finite set of states S. In general, for a path $s_1 s_2 \dots s_n$ and any probability distribution we have the following rule

$$\Pr[s_1 s_2 s_3 \dots s_n] = \Pr[s_1] \cdot \Pr[s_2 \mid s_1] \cdot \Pr[s_3 \mid s_1 s_2] \cdots \Pr[s_n \mid s_1 \dots s_{n-1}].$$

For general distributions these probabilities can be all distinct. The assumption in a kth order Markov chain is that, given any previous history, only the k most recently visited states affect the future transitions, formally

$$\Pr[s_n \mid s_1 \dots s_{n-1}] = \Pr[s_n \mid s_{n-k} \dots s_{n-1}], \quad \text{for } n-k \geq 1.$$

As an example, in a Markov chain with $k = 2$ the rule above simplifies to

$$\Pr[s_1 s_2 s_3 \dots s_n] = \Pr[s_1]\cdot\Pr[s_2 \mid s_1]\cdot\Pr[s_3 \mid s_1 s_2]\cdot\Pr[s_4 \mid s_2 s_3] \cdots \Pr[s_n \mid s_{n-2} s_{n-1}].$$

Therefore, a k-th order Markov chain is described by giving, for each state $s \in S$ and path p of length at most k, a probability that the next state is s given that the k last visited states are those in path p. This implies that the distribution given by the Markov chain can be specified by giving at most $|S|^{k+1}$ numbers, rather than infinitely many.

Furthermore, given a set of data consisting of paths along S, one can build a kth order Markov chain that approximates their distribution as follows: compute all the empirical probabilities $\Pr[s_{i+1} \mid s_1 \ldots s_i]$ for $0 \leq i \leq k$ on the data. By the discussion above, these figures are enough to approximate $\Pr[p]$ for each path p of every length. Of course, whether the figure computed in this way approaches the real probability of p in the source of the data depends on 1) the amount of training data available (the more data, the better approximation), and on 2) the degree to which the Markovian assumption is valid for the source.

In our methodology, we define the set of states S to be the set of tags in our log data. Then, for some parameter k, we create a k-th order Markov chain for each of the classes, each one modelling the typical sequences of tags (requests) for that class. In our case, we train two models: one for buyers and one for non-buyers. Given the path followed in the current session, these two chains can be used to compute probabilities $\Pr[p \mid \mathit{buyer}]$ and $\Pr[p \mid \mathit{nonbuyer}]$, where p is the sequence of previous k tags in the session. Using Bayes' rule, we can then estimate the converse probabilities $\Pr[\mathit{buyer} \mid p]$ and $\Pr[\mathit{nonbuyer} \mid p]$. For example,

$$\Pr[\mathit{buyer} \mid p] = \Pr[p \mid \mathit{buyer}] \cdot \Pr[\mathit{buyer}] / \Pr[p]$$

where we approximate $\Pr[\mathit{buyer}]$ as the fraction of buyers in the data, and $\Pr[p]$ can be ignored because what matters really is the *ratio* of $\Pr[\mathit{buyer} \mid p]$ to $\Pr[\mathit{nonbuyer} \mid p]$. That is, given that the user has followed this path, the Markov chains guess the probabilities that later in the future s/he buys or does not buy. At training time, these two figures (the *buying* and *non-buying* probabilities) are added as new variables to the line describing the current transaction in the training set. At prediction time, these two figures are added as new variables to the information passed to the predictor for the current transaction.

We have used $k = 2$ (second-order Markov chains) for the experiments reported in the next sections. After some experimentation, this value seemed to provide the best results for our attempts. It is intuitively clear that remembering the last two visited pages gives more information than remembering only the last one. On the other hand, as k grows, each individual path is less frequent in the data, the approximations of the probabilities are coarser, and predictive accuracy is reduced (i.e., overfitting tends to appear). This effect is especially harmful on buying paterns which are rare in our datasets. In particular, $k = 3$ gave results comparable to $k = 2$, and predictions were significantly worse for $k > 3$. This conclusion may, of course, be different in other contexts.

3.3 Learning Module

The resulting sequence of transformed and enriched log entries can be treated as a dataset where the order of examples is irrelevant and each example is a tuple of simple values (numerical or categorical values). This is what is needed to apply most machine learning algorithms in the literature.

In this first prototype we have chosen the Naïve Bayes classifier as a learning algorithm, for a number of reasons: 1) it is easy to understand, has no user-entered parameters, and has very low CPU time and memory requirements, both for training and for prediction; 2) in preliminary experiments [13], it performed about as well as more sophisticated methods, such as decision trees and boosting; and 3) it assigns probabilities to its predictions, rather than hard buy/non-buy decisions, and this is essential for our prototype. Naturally, there is ample room for trying other and more advanced prediction methods in later versions, which administrators can choose according to their data and available resources.

4 Results and Evaluation

In this section we describe the data, our experiments with the prototype, and discuss the results obtained. We want to remark that our goal was to test a generic approach without fine tuning the experiments with domain knowledge, rather than obtaining the best possible figures for this particular dataset.

4.1 The Dataset

The data consisted of the transactions collected over approximately 5 days, from 01/22/2007 1am to 01/26/2007 11pm, consisting of 3.7 million transactions. We distinguish "transaction" and "request"; a transaction in this paper is a user-initiated action (click) to the site that s/he views as an atomic operation. Internally, each transaction in the dataset corresponds to an average of 13 requests (hits) to the server, including media files, CSS, Javascript and the final HTML output. To log user actions only, the dataset was produced by the site's dynamic application; additional code was added at the end of each executing script to log the transaction data after the actions were executed. By doing so, the data is already cleaned and more accurate, as opposed to the access log from a Web server where URL actions might be ambiguous. Furthermore, the application can log directly the user session, not only its IP address, allowing us to correctly differentiate NAT/firewalled users. A session is a sequence of transactions initiated by a user in a definite amount of time.

The cleaned data contained 218 different "tags", "pages" or user request types and about 3.7 million transactions, grouped in 452,939 sessions. Of these sessions, about 3% ended in purchase after it was cleaned, and 234,261 corresponded to returning users. The average session length was 8.2 transactions, increasing to 18.5 for buying sessions. Because buying sessions are longer, the percentage of transactions labeled as "buying" is larger than the percentage of buying sessions, namely about 6.7% rather than 3%.

From the cleaned dataset we prepared training and testing datasets. To force the learning algorithm to pay attention to the *buying* class, the training dataset was built by randomly selecting about 25% of the buying sessions, and about three times as many random nonbuying sessions. This way, we made sure that buying sessions were sufficiently visible while training as most machine learning algorithms tend to ignore underrepresented classes. The training dataset finally consisted of 25,331 sessions, corresponding to approximately 200,000 transactions. The rest of the sessions were included in the testing set, which thus contained 427,608 sessions and 3.5 million transactions.

We also noticed that there were transactions produced by automated bots, i.e. crawlers or Web fetching from other sites, of course never ending in purchase. We kept them in the dataset as it is important that our system learns to identify these as non-buyers: since search queries to B2B providers have a cost and the bots could be abusive or even malicious, they should be assigned low priority or denied access. In [18] we extended the experiments described in this article, to automatically detect and ban *content stealing bots*. The traffic caused by *content stealing bots* represents up 12% of the total traffic in the analyzed dataset; in case of an overload, these sessions should be the first to be discarded and have be correctly identified.

4.2 Quantities of Interest in Admission Control

After building a classifier using the training dataset, we can compute for each transaction in the testing set a "true" buying/nonbuying label and a "predicted" label. Thus, we can divide them into the four typical categories of true positives (tp), false positives (fp), true negatives (tn), and false negatives (fn). For example, false positives are the transactions that are predicted to be followed by purchase but that in fact are not.

We observed the classical *recall* and *precision* measures, as well as one that is specific to our setting, which we call *%admitted*.

- %admitted is (tp+fp)/(tp+fp+tn+fn), or the fraction of incoming transac-

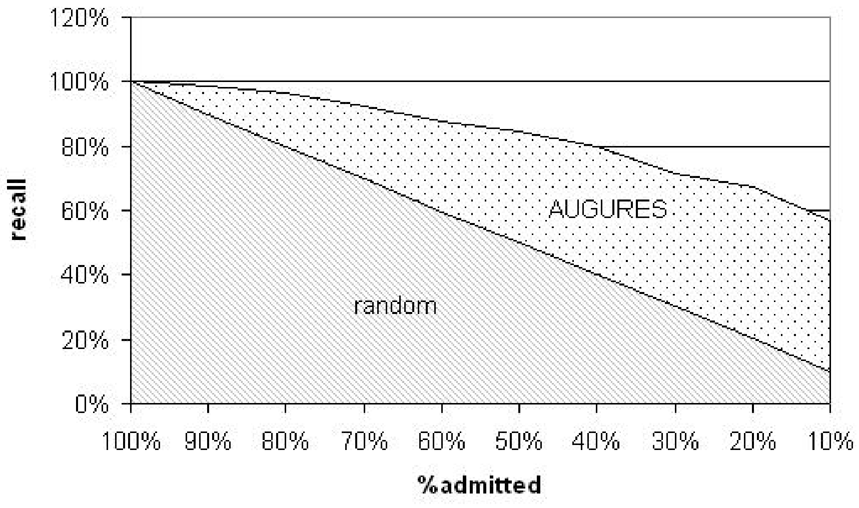

Fig. 2. recall vs. %admitted

tions that the system admits.

- the recall is tp/(tp+fn), the fraction of admitted buyers over all buyers.
- the precision is tp/(tp+fp), the fraction of admitted buyers over all admitted transactions.

Our ultimate goal is to use these predictions for prioritizing sessions, so that low priority sessions can be queued, redirected to a static page, or even dropped when the server is under a heavy load condition. The meaning of a false positive and a false negative in this context is very different. Rejecting a false negative (fn) session implies a substantial loss (in revenue), so it is preferable to accept it even at the cost of keeping many false positives (fp) in the system. Therefore, these two figures should be examined and evaluated separately.

In our case, since we are using the Naïve Bayes classifier, we have good control over the %admitted quantity. Indeed, this classifier provides a probability of buying $p(t)$ for each transaction t. Set some threshold value $T \in [0, 1]$; then we can decide to admit those transactions t such that $p(t) > T$. By increasing T, we will make it more difficult for a transaction t to pass this test, hence we will admit less transactions. Conversely, if we lower T, more transactions will be admitted. Once the Naïve Bayes classifier is built, we use the training set to tabulate the function of T to the actual %admitted, for future use.

4.3 Classifier Performance

A first set of results was obtained applying the learned Naïve Bayes classifier (containing the Markov models prediction) on the testing dataset. Figures 2 and 3 present the evolution of recall and precision as we vary the percentage

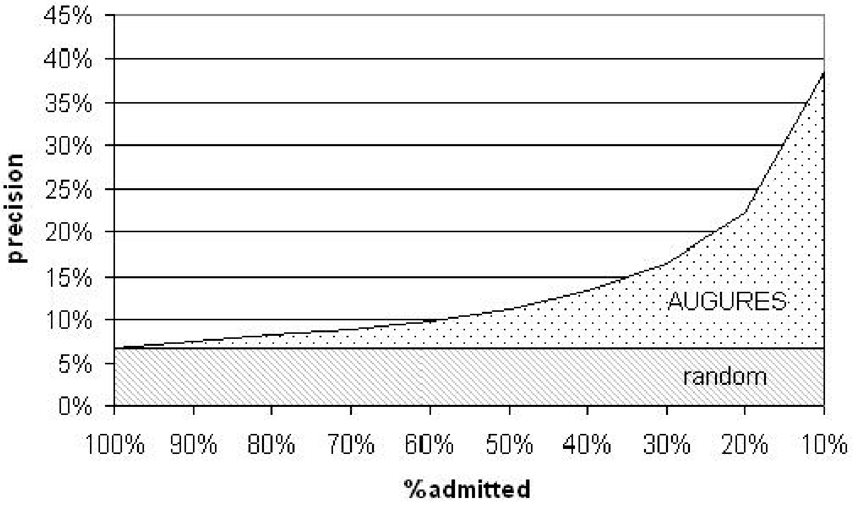

Fig. 3. precision vs. %admitted

of admissions from 100% (no rejections) to 10%.

As predicted, there is a nontrivial relation between %admitted, recall, and precision. Naturally, as we become more restrictive in the number of admissions, we loose true customers, but at a rate smaller than if we were choosing at random. For example, if we choose to admit 50% of the transactions, AUGURES will still admit 91% of those that will end in purchase (rather than 50% as we would if we were selecting them randomly). Similarly with precision: no matter the percentage of admissions we fix, if we choose transactions randomly, we will choose buying ones in the same proportion as there are buying transactions in the dataset, namely about 6.7%. By using the AUGURES strategy, when we are admitting say 50% of all transactions, about 12% will end in a purchase, an improvement by a factor of almost 2.

These figures become even more interesting as we restrict %admitted more and more: when admitting only 10% of transactions, AUGURES will still admit 76% of all real buyers and 35% of admitted users will really buy. This means an increase by a factor of over 5 in precision over random selection. The results demonstrate the potential of the predictor module in a self-adaptive system: as more users arrive and the capacity of the infrastructure is exceeded, the proportion of admitted sessions that will end up in a purchase increases. In other words, the system prioritizes the most profitable sessions when it becomes most necessary.

In Table 1 we present the *recall* and *precision* for clicks 1 through 3. *Recall* represents the fraction of real buyers that are admitted by the predictor, while *precision* is the fraction of predicted buyers. With this experiment we wanted to show that there is enough information to prioritize sessions right from their

	First Click	Second Click	Third Click
%recall	47.6	51.1	53.0
%precision	15.2	18.6	21.0

Table 1
Recall and precision for clicks 1 to 3.

first access to the site, improving predictions with the number of clicks. For the first access, we detected 15% or better of buying sessions, in contrast with a random strategy which would pick only 3% of buyers.

4.4 Performance in Real-Time Prediction

Our next experiments test the AUGURES prototype under a simulated environment over a 24-hour period. This dataset belonged to different dates for the same system, and contained 112,000 transactions. Figure 4 presents the evolution of the rate of transactions/hour in this workload, sampled every 5 minutes. It averaged about 4,600 transactions/hour and hasd a peak of about 13,000.

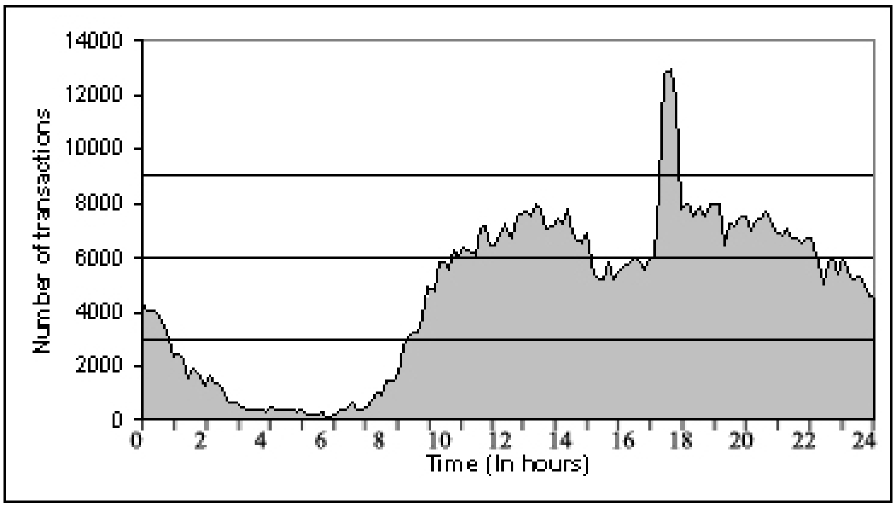

Fig. 4. Transactions/hour rate in the workload

More precisely, we compared the number of transactions that would end up in purchase if admitted with the AUGURES prototype and if admitted with a random selection strategy. For the simulation we chose different values of a parameter MAX denoting the maximum rate of transactions/hour a given infrastructure can accept without throughput degradation. We also chose some time unit T in minutes; the results we report are for T=5 minutes, but results did not vary significantly in the range T=[1 minute, 60 minutes]. We fed AUGURES with the workload corresponding to the reference day, sequentially. Every T minutes, AUGURES computes the rate transactions/hour from the

MAX	%admitted	%recall	%precision	%improve
13500	100	100	6.61	0
9000	99.31	99.97	6.65	0.66
7500	98.56	99.82	6.69	1.28
6000	90.65	97.26	7.09	7.29
4500	75.46	92.82	8.13	23.01
3000	63.81	88.27	9.14	38.32
1500	39.11	77.20	13.04	97.36

Table 2
Results of simulation on real workload

current load and, with this figure, it recomputes the threshold of the classifier so that it admits at most (approximately) MAX transactions/hour during the next T minutes. That is, if the current rate is less than MAX, it sets the threshold to 0 so that all transactions are admitted. Otherwise, if the instantaneous load L is greater than MAX, it sets the threshold so that a fraction of about MAX/L of the transactions are admitted.

The results of this simulation are presented in Table 2. Rows correspond to the different values of MAX tried, ranging from one exceeding the peak (in which no transaction is rejected) to one where MAX is almost 1/10 of the peak. Columns correspond to

- % of transactions admitted,
- % of recall obtained, i.e., % of all buying transactions that are admitted,
- % of precision, i.e., % of admitted transactions that lead to purchase,
- and %improvement over the random strategy (e.g., if the random strategy admits 1,000 buying transactions and AUGURES admits 1,200, % improvement is 20%).

Thus, for example, when the maximal load MAX is set to 6,000 (about 50% of the peak), we still accept 90.65% of transactions, miss less than 3% of the buying ones (recall=97.26%), and all in all we accept 7.3% more buying transactions than the random strategy. When setting MAX to 3,000 (i.e., assuming an infrastructure able to handle less than 25% of the peak), we still accept 63.8% of transactions, reject only 12% of the buying ones, and do about 40% better than the random strategy.

Another view of the process is presented in Figure 5, which presents the absolute numbers of buying transactions accepted by AUGURES and by the random strategy. Here we chose T=1 hour as time unit and MAX=3,000. It can be seen that, as expected, at times when the workload is low, AUGURES did not act, so the curves for both strategies coincide. On the other hand,

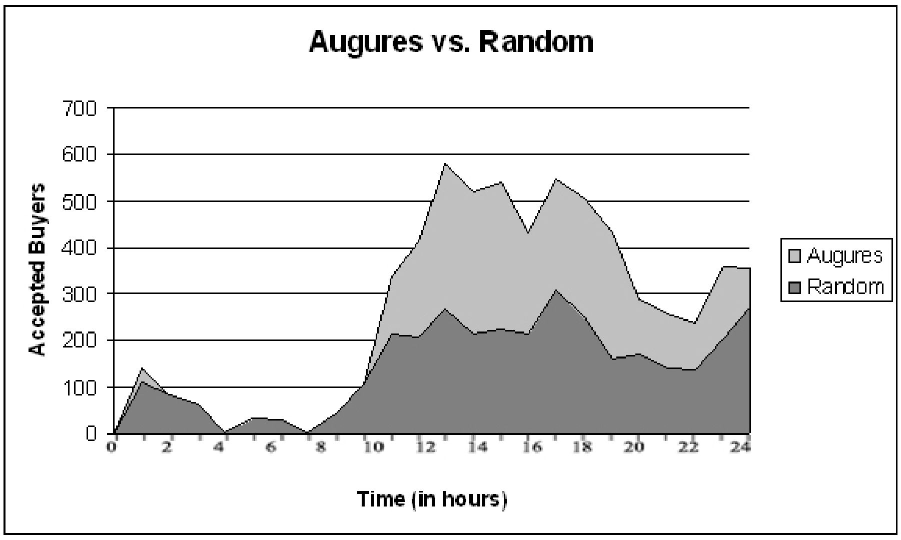

Fig. 5. Admitted buys by AUGURES and random selection

when the workload exceeds MAX, AUGURES chose better and admitted a substantially higher number of buyers. In other words, the area between both curves is the %improvement column of Table 2.

5 The Effect of Dynamic Server Activation

In the next experiment we wanted to simulate the benefit of our technique in an environment where not all servers have to be active at all times, but where they can be turned on and shut down dynamically. This possibility is today routinely used in all centers having a reasonable number of in-site servers: in this case, shutting down unnecessary servers results in immediate savings in power (both for the server itself and for cooling), hence lower costs. Also, dynamic provisioning of external servers is becoming mainstream too. Services such as Amazon's Elastic Computing, Sun's Grid, or Google's Cloud, enable ecommerce sites to be configured at a minimum number of server resources and provision according to the incoming load, with a *de facto* unlimited number of servers. We phrase the rest of the section in terms of dynamic provisioning for clarity.

For this experiment, we used the 24-hour dataset in the previous section and studied the number of servers that would be needed at different times of the day to serve the incoming load at that time. Then, we compared the number of servers that would be needed if our only interest was to serve all (or most) buyers, possibly dropping many nonbuyers. The AUGURES system makes a big difference between both situations: using the predictions from AUGURES, the most promising customers can be allocated to server 1, the next most

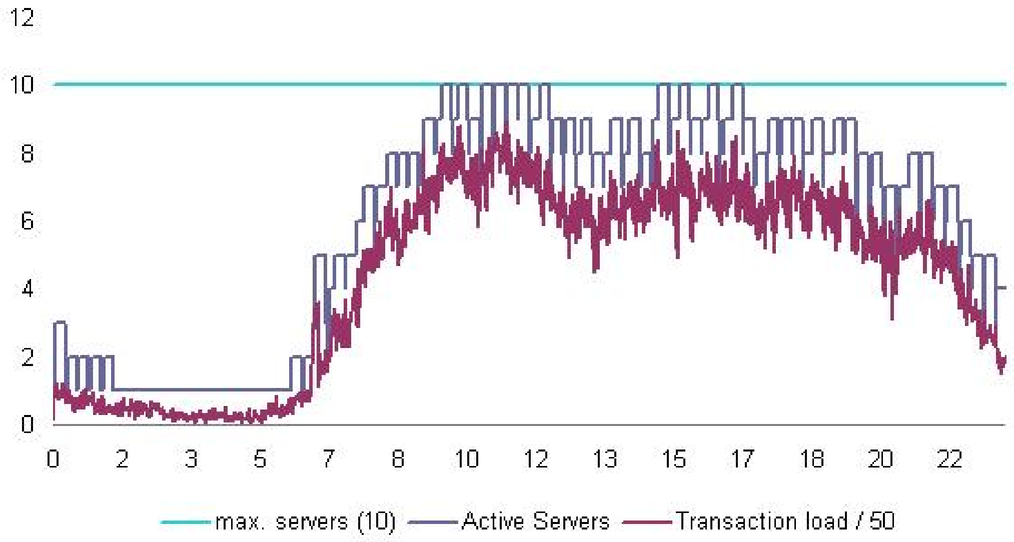

Fig. 6. Evolution of load and number of dynamically provisioned servers

promising ones to server 2, etc. At some point, the highest-numbered active servers are likely to be receiving mostly non-buyer sessions. If the number of buyers in a server is sufficiently low with respect to the cost of having the server running, it is beneficial to the company to shut it down, maybe failing to serve a few buyers, but mostly discarding nonbuying traffic. Conversely, if all servers are receiving a non-negligible number of buyers, it may be time to request a new server to the dynamic provisioner.

While we did not simulate this mechanism strictly speaking, we carried out an experiment to let us visualize its potential benefits. Note that the intention of the experiment was not solving the problem of deciding when acquiring or releasing a new server. There is extensive literature on the topic, but we used a simple yet effective approach: we average the load change in the last 10 minutes of requests, and multiply it by the time it would take to start a new server (chosen to be 10 minutes for the experiments); we request a new server if the expected load is higher than the current capacity. The same mechanism was applied to release a server, but with the constraint that a newly added server must stay on for at least 15 minutes.

In the experiment, we assumed that each server can handle 50 requests per minute. This would mean, in our dataset, that 10 servers would be needed to serve the peak load. Figure 6 shows the evolution of the number of active servers when the strategy above was used to acquire/release servers dynamically. It can be seen that it closely parallels the evolution of the load, and it fact it overprovisions by a slight amount. The area between the plot and the 10-server line above the peak represents the savings in server cost with respect to keeping all 10 servers up at all times.

#servers	Buyers Static & Dynamic	Buyers Augures	Benefit Static	Benefit Dynamic	Benefit Augures
1	3166	12961	2355	2355	12150
2	5968	13812	4345	4512	12361
3	8624	14606	6190	6612	12593
4	11131	15361	7886	8575	12805
5	13474	16064	9417	10391	12982
6	15465	16659	10597	11888	13083
7	16703	17026	11024	12665	12988
8	17115	17146	10624	12700	12700
9	17157	17157	9855	12422	12422
10	17157	17157	9044	12323	12323

Table 3
Buyers served and benefits of different strategies

So far, this is independent of the usage of AUGURES. To see the potential benefit of combining dynamic server activation (or dynamic provisioning) and server prioritization, one has to study the amount of buyers served at different servers. Table 3 shows some figures for analysis, accumulated over the 24-hour period in the dataset. It compares three strategies: the “static” strategy, in which a fixed number of servers (from 1 to 10) are permanently active, and each new session is assigned to the least-numbered server with available capacity; the “dynamic” strategy, in which servers are turned on and off as the load requires, and sessions are assigned to servers as above; and the combination of the dynamic strategy which AUGURES, in which incoming sessions are assigned to servers 1, then 2, etc. in order of purchase probability (so most promising sessions are assigned to server 1). Each table row represents a number of servers, from 1 to 10. For each number of servers i, the five columns list: how many buyers are served by servers $1 \ldots i$ in the static and dynamic strategies (note they have to be the same); how many buyers are served by servers $1 \ldots i$ in the AUGURES strategy; and the actual benefit generated by servers 1 to i in each of the three strategies. To compute this benefit, we have assumed that each (served) buyer produces a benefit of 1 and each server a cost of 0.33 per minute, a ratio which approximates the average benefit per buyer in our dataset's agency and the actual cost of Amazon's Elastic Computing current prices.

From the table one can observe a number of phenomena: first, even in the static strategy, server 10 serves no buyers. This is because our strategy for predicting future load tends to overestimate, so server 10 is basically overprovisioning and serves no load. Also, as is to be expected, in the static and dynamic strategies, server 1 gets about 19% of the buyers (i.e., more than 1/10): this is because the session assignment strategy tries to keep it fully busy, so it receives the largest share of the load. In contrast, in the AUGURES strategy, lower-numbered servers get a fraction of buyers larger than their fraction of load share. In particular, server number 1 alone serves 75% (=12961/17157) of the buyers.

For the three strategies, the benefit grows with the number of servers up to 6-8 servers, at which point each new server produces more cost than benefit. However, AUGURES' maximum benefit (13082) exceeds slightly that of the dynamic strategy (12699), and both exceed by far that of the static one. The point to note here is that the actual figures depend on the values we have assumed on the ratio of benefit per buyer / server costs. In a situation where the ratio is much lower (e.g., server time is expensive), AUGURES' benefit could potentially exceed the dynamic strategy by far. In the extreme case, the company could serve 75% of its buying customers with just 1 server, while the dynamic strategy would need at least 5 servers for the same effect.

Admittedly, there is a strong practical consideration that we have not taken into account in this simulation: companies are reluctant to loosing not only buyers, but also non-buying sessions because that means a loss in user satisfaction and prestige (hence, future revenue). On the other hand, marketing techniques produce estimates of the value of a lost session, which could be incorporated into the computations above.

6 Other Potential Applications

This section describes other potential applications for the techniques described in the article. We summarize as using machine learning techniques to characterize individual anonymous Web sessions in real time from available session information and past data.

A first approach to extended the experiments presented in this article, is to use other metrics than if the session will end up in a purchase, to prioritize it. Two that we plan to investigate in the immediate future are:

- Expected purchase value and profit margin for the session, in case not all purchases have the same value or profit.
- Number of served sessions. This is an important metric, applicable to sites that don't sell products, but where user satisfaction is important. The idea is the following: Assume that resources used by a session i.e., computing time, memory, bandwidth, database searches, among others, can be predicted with some accuracy in the way we predicted purchases; preliminary experiments indicate that this is indeed the case. Then the system could

prioritize sessions that use less resources and penalize resource-consuming ones. In this way, more sessions per time unit could be served, increasing the number of served users. This is important since being served at all is probably the most important factor for user satisfaction.

For ecommerce sites, features that we have not considered but that could be taken into account are: magnitude of the transaction, type of product, profit margins, type of customer (new, returning, gold), demographic group, and a combination of these features according to the type of site and path. The idea is that by predicting the benefit to be obtained from the session, it could be contrasted with infrastructure costs, and shape QoS accordingly.

Information, media streaming, or social-networking sites can definitely benefit from predictions about session costs and user contribution to the network, to change the QoS of the session. Users that contribute more, or that seem to be impatient in information sites, could automatically be granted more bandwith and better response times. On the other hand, *leachers* and *content stealing bots* [18] should see their QoS lowered if system resources are low, and even denied access in case of overloads. For sites that do not rely on external providers or pages that are mainly static or cacheable, prefetching can be used as in [7,12,17]. Predicting user intentions can be use in the field of *dynamic content adaptation* [2,14], where the page content, layout, links, and banners change according to the predictions about the user.

7 Conclusions and Future Work

Websites might become overloaded by certain events such as news events or promotions, as they can potentially reach millions of users. When a peak situation occurs most infrastructures become stalled and throughput is reduced. To prevent this, load admission control mechanisms are used to allow only a certain number of sessions; but as they do not differentiate between users, users with intentions to purchase might be denied access. As a proof of concept, we have taken data from a high-traffic online travel agency and learned to predict users' purchasing intentions from their navigational patterns.

In our experiments, we are able to train a model from previously recorded navigational information that can be used to determine, with nontrivial probability, whether a session will lead to purchase from the first click. The maximum number of allowed users to the site can be regulated, according to the infrastructure's capacity and goal specification, by placing a threshold over the predicted buying probability of incoming transactions. That is, the model can adapt itself dynamically to the workload while maintaining reasonable recall and precision.

As future work we plan to investigate other models, including hidden Markov models, Bayesian Networks, and k-means clustering, to improve predictions. We are also plan to explore other classification criteria (see Section 6) and resource usage, as well as their combinations for flexible configuration based on high-level policies. At the same time we plan to test the applicability of the predictor models, by testing their real-time performance. As a first step we plan to use the prediction system as a base layer for a utility-based resource management system.

Acknowledgements

Thanks to the reviewers for interesting comments and contributions, to Atrapalo.com, who provided the experiment datasets and domain knowledge.

This research work is supported by the Ministry of Science and Technology of Spain under contract TIN2007-60625.

R. Gavaldà is partially supported by the 6th Framework Program of EU through the integrated project DELIS (#001907), by the EU PASCAL and PASCAL2 Networks of Excellence, and by the DGICYT MOISES-TA project, TIN2005-08832-C03.

For additional information about the authors, visit the Barcelona eDragon Research Group Web site [20].

References

[1] V. Almeida, M. Crovella, A. Bestavros, A. Oliveira. *Characterizing Reference Locality in the WWW.* Proc. IEEE/ACM International Conference on Parallel and Distributed System (PDIS), december 1996.

[2] D. Bonino, F. Corno, G. Squillero. *A Real-Time Evolutionary Algorithm for Web prediction.* Proc. IEEE/WIC International Conference on Web Intelligence (WI'03), 139–145, october 2003.

[3] L. Cherkasova, P. Phaal. *Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites.* IEEE Transactions on Computers 51(6), 669–685, june 2002.

[4] G. Cheliotis, C. Kenyon. *Autonomic economics: a blueprint for selfmanaged systems.* IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems, august 2003.

[5] M. Deshpande, G. Karypis. *Selective Markov Models for Predicting Web Page Accesses.* ACM Transactions on Internet Technology (TOIT) 4:2, 163–184, may 2004.

[6] J. Guitart, D. Carrera, V. Beltran, J. Torres, E. Ayguadé. *Designing an Overload Control Strategy for Secure e-Commerce Applications.* Computer Networks, 51(15), 4492-4510, october 2007.

[7] B. Lan, S. Bressan, B. C. Ooi, K. Tan. *Rule-assisted prefetching in Web-server caching.* Proc. ACM Intl. Conference on Information and Knowledge Management (CIKM'00), 504–511, november 2000.

[8] M. Levene, J. Borges, G. Loizou. *Zipf's Law for Web Surfers.* Knowledge and Information Systems 3, 120–129, 2001.

[9] M.L. Littman, T. Nguyen, H. Hirsh. *A Model of Cost-Sensitive Fault Mediation.* IJCAI Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems, august 2003.

[10] D.A. Menascé, V.A.F. Almeida, R. Fonseca, M.A. Mendes. *A Methodology for Workload Characterization of E-commerce Sites.* Proc. 1st ACM Conference on Electronic Commerce, 119–128, 1999.

[11] D. Menascé, V.Almeida, R. Riedi, F. Ribeiro, R. Fonseca, W. Meira, Jr. *In Search of Invariants for e-Business Workloads.* Proc. 2nd ACM Conference on Electronic Commerce, 56–65, 2000.

[12] A. Nanopoulos, D. Katsaros, Y. Manolopoulos. *Effective Prediction of Web-user Accesses: a Data Mining Approach.* Conference on Mining Log Data Across All Customer Touchpoints (WebKDD'01), San Francisco, 2001.

[13] N. Poggi, T. Moreno, J. Berral, R. Gavaldà, J. Torres. *Web Customer Modeling for Automated Session Prioritization on High Traffic Sites.* 11th Intl. Conference on User Modelling (UM2007). Springer Lecture Notes in Computer Science 4511, 450–454.

[14] M. Rabinovich, O. Spatschek. *Web Caching and Replication.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2002.

[15] R.R. Sarukkai. *Link Prediction and Path Analysis Using Markov chains.* Computer Networks: The International Journal of Computer and Telecommunications Networking, 33:1–6, 366–386, 2000.

[16] Y. Wei, C. Lin, F. Ren, E. Dutkiewicz, R. Raad. *Session Based Differentiated Quality of Service Admission Control for Web Servers.* Proc. Intl. Conference on Computer Networks and Mobile Computing (ICCNMC'03), 112–116, 2003.

[17] Q. Yang, H. H. Zhang, T. Li. *Mining Web Logs for Prediction Models in WWW Caching and Prefetching.* Proc. 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 73–478, 2001.

[18] N. Poggi, J.L. Berral, T. Moreno, R. Gavaldà, J. Torres. *Automatic Detection and Banning of Content Stealing Bots for E-commerce.* In NIPS 2007 Workshop on Machine Learning in Adversarial Environments for Computer Security, december 2007.

[19] WebPage. Internet World Stats, Usage and Population Statistics. `http://www.internetworldstats.com/`, 2008.

[20] WebPage. Barcelona eDragon Research Group. Technical University of Catalonia. `http://research.ac.upc.edu/eDragon`, 2008.