

Self-Adaptive Revenue-Based Web Session Management

Nicolás Poggi¹, Toni Moreno^{2,3}, Josep Lluís Berral¹,
Ricard Gavaldà⁴, Jordi Torres^{1,2}

¹ Computer Architecture Department, U. Politècnica de Catalunya, Barcelona, Spain

² Barcelona Supercomputing Center, Barcelona, Spain

³ Department of Management, U. Politècnica de Catalunya, Barcelona, Spain

⁴ Department of Software, U. Politècnica de Catalunya, 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 is crucial for proper self-management and business efficiency. When server resources are exceeded, admission control systems are necessary to avoid denial of service and maintain high throughput in terms of properly finished sessions; this approach, however, does not maximize potential revenue as it treats all non-logged user sessions the same. In the present study we describe the architecture of AUGURES, a system that learns to predict web user's intentions 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 prioritize buying sessions or deny others when resources are insufficient, improving sales throughput in heavy load conditions. We test our approach on access logs from a high-traffic online travel agency, obtaining promising results.

1 Introduction

1.1 The Current Web Scenario

According to the "Internet World Stats" there is 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 the user 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 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 not serve any of the connections, as resources get locked and a race condition occurs. System administrators might get warnings from resource-monitoring services, but in general they get to the problem when the situation has already occurred, where controlling the rate users try to access the site is out of their reach. To address this issue,

session-based admission control systems [6, 3] allow to maintain QoS on overloads by keeping a high throughput in terms of properly finished sessions for a limited number of users. However, by denying access to exceeding users, the website loses potential revenue from customers.

To overcome this situation, in this study we propose a novel approach consisting of learning, from past data, a model for anonymous web user behavior in a real, complex website that does experience the overload situations mentioned above. The model can be used to support decisions regarding the allocation of the available resources, based on a revenue-related metrics. The learning phase captures in a model 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 maintain in the system even in the case of a severe overload. In this sense, we are proposing a per user-adaptive revenue based session management.

1.2 Motivation

Our preliminary experiments showed that the user's intention for visiting a site can be predicted to some extent from its 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 learn 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 web 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 [8][1]. Also, works such as [11] have studied logs from real and simulated auction sites and bookstores; there are no studies that we know

about 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, 12, 7, 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 vast 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 [14, 2] 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 [15, 5] 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 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 [6, 3, 16]; the work presented here is an extension to these approaches. 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.

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 that takes the historical logfile and produces the *predictor*, and a real-time component, the *selector*, implemented as a service that runs along 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 Markov models

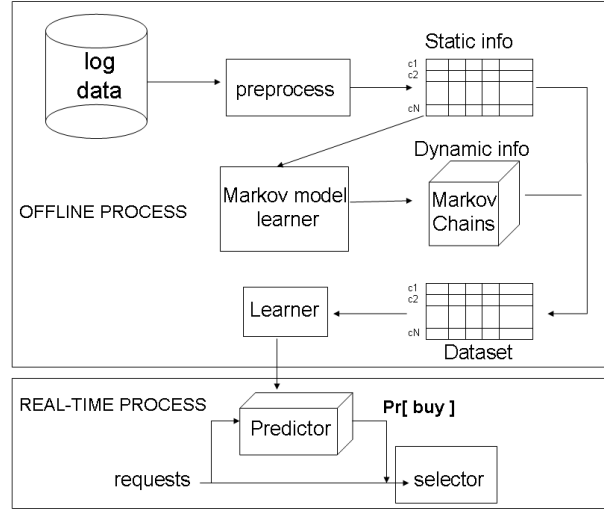


Fig. 1. AUGURES architecture

of both buying and non-buying sessions; the prediction of these models for each individual request is added as extra information to each line of the preprocessor. 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.

The real-time component, the *selector*, runs side-to-side with the session manager of the firewall. When an incoming HTTP/S request arrives, the *selector* reads the entry produced by the firewall and it is run against the *predictor* (built offline), if available information collected previously about the same session and user is also used to compute the buying probability. The computed probability is written to the firewall's active session table along with other useful information such as: current load, server conditions, enterprise policies. This information is used by the firewall to prioritize and even discontinue some of the sessions according to the current load and policies. Note 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 provides additional information helping the firewall to make informed actions rather than blind or random decisions.

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. In the future, it should be possible to run 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 from 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 in 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 a “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. Instead, the class 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 at the time of prediction, since it is precisely what we are trying to predict in the future. Thus, the class can only be computed in datasets with past information, those used for off-line 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.

Recall that a k -th order Markov chain consists of 1) a set of states S and 2) for each state 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 . In particular, by using a chain-type rule, we can use a Markov chain M to assign a probability $\Pr[p|M]$ to each path p of any length. If the transition probabilities of M are inferred from a set of observed paths, then one can take $\Pr[p|M]$ as an approximation of the probability of path p in the data obtained by forgetting all history before the last k steps.

More precisely, for some parameter k , we create two k -th order Markov chains for each of the classes, which models the typical sequences of tags (requests) for each class.

In our case, we train two models: one for buyers and one for non-buyers. At prediction time, and given the path followed in the current session, these two chains can be used to compute probabilities $\Pr[p|buyer]$ and $\Pr[p|nonbuyer]$, where p is the sequence of previous k tags in the session. Using Bayes' rule, we can then estimate the converse probabilities $\Pr[buyer|p]$ and $\Pr[nonbuyer|p]$. 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. We have used $k = 2$ (second-order Markov chains) for the experiments reported.

The *buying* and *non-buying* probabilities are added as new variables (tuple elements) to the static information in the final log.

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 (numeric 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, it performed about as well as other more sophisticated methods, such as decision trees and boosting [13], 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 from according to their data and resource requirements.

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, 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, 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. We also noticed that there were transactions produced by automated bots, i.e. crawlers or web fetching from other sites, naturally 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 malicious, they should be assigned low priority or denied access.

From the cleaned dataset of 3.7 million transactions, we prepared a training and a testing dataset. The testing dataset was made from randomly selecting 25,331 sessions corresponding to approximately 200,000 transactions. The rest of the dataset was left for training containing 427,608 sessions and 3.5 million transactions.

Note: Because of time restrictions, some experiments in this report have been performed on a testing dataset of 500,000 transactions. Results on the complete dataset will be included in the final version.

4.2 Quantities of Interest and Controlling Admissions

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 4 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 did not.

The measures we are interested in this study are the classical *recall* and *precision*, 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 transactions that would be admitted into the system, according to available resources.
- the recall is $tp/(tp+fn)$, the fraction of real buyers that are admitted.
- the precision is $tp/(tp+fp)$, the fraction of admitted transactions that really end up in purchase.

Our ultimate goal is to use these predictions for prioritizing sessions, and deny the ones with lower priorities when server is under 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) at in the system. Therefore, these two figures should be looked at separately and carefully.

In our case, since we are using the Naïve Bayes classifier, we have a 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, it is easy to tabulate the relation of T to the actual %admitted, for future use.

4.3 Classifier Performance

A first set of results is 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 vary the percentage of admissions from 100% (no rejections) to 10%.

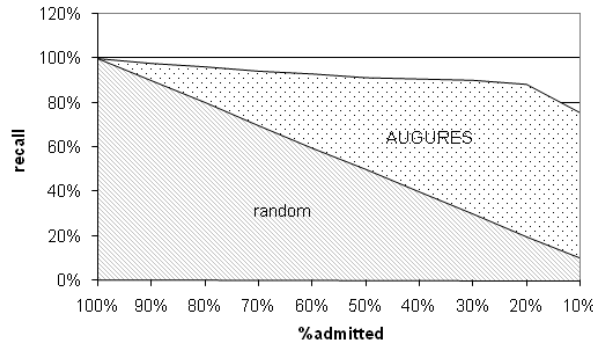


Fig. 2. %admitted vs. recall

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 some potential customers, but at a rate smaller than if we were choosing at random. For example, when we are admitting 50% of the transactions AUGURES will still admit 91% of those that will end in buy (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 4.8%. By using the AUGURES strategy, when we are admitting say 50% of all transactions, about 12% will end in a purchase, a 250% increase over the baseline 4.8%.

These results on the 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 700% 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 infrastructure capacity is exceeded, the proportion of admitted sessions that will end up in a purchase

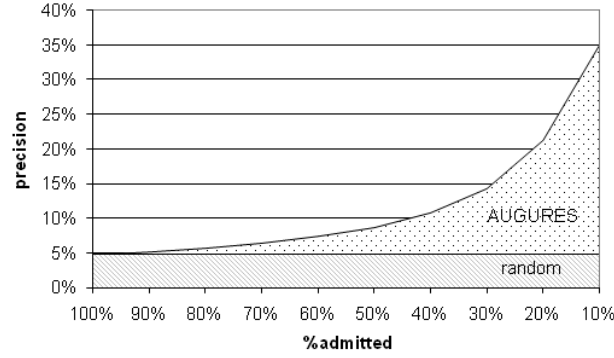


Fig. 3. %admitted vs. precision

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

Fig. 4. Recall and precision for first clicks 1-3.

increases. In other words, the system prioritizes the most profitable sessions when it becomes most necessary.

In Table 4 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 access to the site, improving predictions with the number of clicks. For the first access, we can detect 15% or better of buying sessions, opposed to a random strategy which would pick only 3% of buyers.

4.4 Performance in Real Time Prediction

Our next experiments tests the AUGURES prototype under a simulated environment over a 24-hour period. This dataset was taken from a different date and contained 112,000 transactions. Figure 5 presents the evolution of the rate of transactions/hour in this workload, sampled every 5 minutes. It averaged about 4,600 transactions/hour and has a peak of about 13,000.

More precisely, we compare 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 choose different values of a value MAX denoting the maximum rate of transactions/hour a given infrastructure can accept without throughput degradation. We also choose some time unit T in minutes; in our experiments we set T=5 minutes, but results do not vary much in the range T=[1 minute,60 minutes]. We

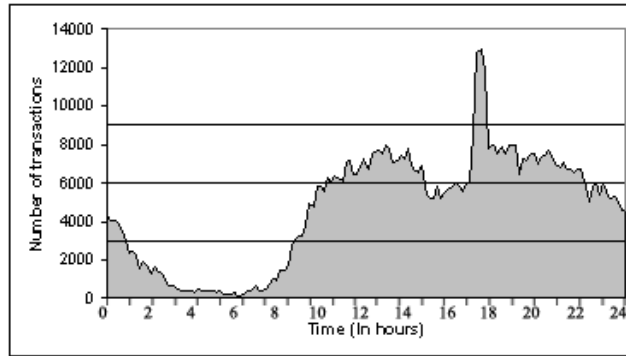


Fig. 5. Transactions/hour rate in the workload

feed AUGURES with the workload corresponding to the reference day, sequentially. Every T minutes, the rate transactions/hour is computed from the current load and, with this figure, the threshold of the classifier is recomputed so that at most (approximately) MAX transactions/hour are admitted during the next T minutes. That is, if the current rate is less than MAX the threshold is set to 0 so that all transactions are admitted. Otherwise, if the instantaneous load L is greater than MAX , we set the threshold so that a fraction of about MAX/L of the transactions are admitted.

The results of this simulation are presented in Table 6. 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%).

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

Fig. 6. Results of Simulation on Workload

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.

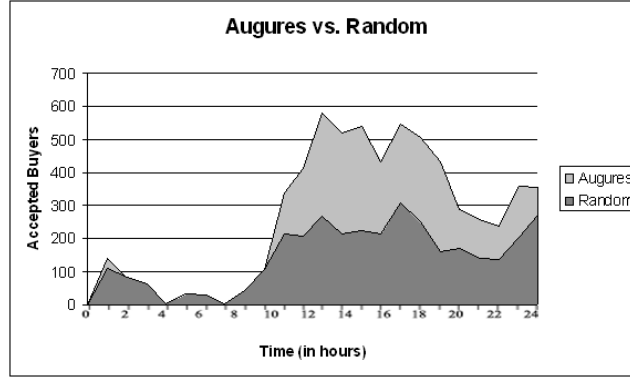


Fig. 7. Admitted Buys by AUGURES and Random Selection

Another view of the process is presented in Figure 7, where we choose $T=1$ hour as time unit and $MAX=3,000$. It presents the absolute number of accepted buying transactions by AUGURES and the random strategy. It can be seen that, as expected, at times when the workload is low, no rejections are made so the curves for both strategies coincide. On the other hand, as the workload exceeds MAX, AUGURES chooses better and admits a substantially higher number of buyers. In other words, the area between both curves is the %improvement column of Table 6.

5 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, so 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 tell apart, 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 going to explore other classification criteria, e.g., magnitude of the transaction, type of product, and profit margins, as well as their combinations and 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 to use the prediction system as a base layer for a revenue based resource management system.

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. *Session-Based Adaptive Overload Control for Secure Dynamic Web Applications*. Proc. 34th International Conference on Parallel Processing (ICPP'05), 341–349, june 2005.
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*. Proc. WEBKDD Workshop, 2001.
13. N. Poggi, T. Moreno, J. Berral, R. Gavalda, J. Torres. *Web customer modeling for automated session prioritization on high traffic sites*. Technical Report, UPC, 2006.

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*. International 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, 73–78, 2001.