UNIVERSITAT POLITÈCNICA DE CATALUNYA

AUGURES, Profit-Aware Web Infrastructure Management

by

Nicolas Poggi M.

Advisor: David Carrera

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

at the

Department of Computer Architecture (DAC)

March 2014

Abstract

Over the last decade, advances in technology together with the increasing use of the Internet for everyday tasks, are causing profound changes in end-users, as well as in businesses and technology providers. The widespread adoption of high-speed and ubiquitous Internet access, is also changing the way users interact with Web applications and their expectations in terms of Quality-of-Service (QoS) and User eXperience (UX). To remain competitive in this environment, online businesses need to adapt their applications in an agile-way to service demand and market changes. These rapid and unstructured changes often result in rich UX, but complex Web applications, with high server resource requirements. Moreover, Web traffic is not only composed of end-user clicks, but is a mix of Business-to-Business (B2B) Web services, crawler and other automated requests, where less than 2% correspond to sessions with the intention to purchase a product. Therefore, understanding the workload and user behavior —the demand, poses new challenges for capacity planning and scalability —the supply, and ultimately for the success of a Web site.

Recently, Cloud computing has been rapidly adopted to host and manage Web applications, due to its inherent cost effectiveness and on-demand scaling of infrastructures. However, system administrators still need to make manual decisions about the parameters that affect the business results of their applications *i.e.*, setting QoS targets and defining metrics for scaling the number of servers during the day. Additionally, not understanding the workload and how users and applications behave under load —the resource consumption, may result in an ineffective or even destructive scaling configuration *i.e.*, when requests come from automated crawlers. This scenario opens a new range of challenges in order to understand how resources are consumed in the Web, and how to relate demand with supply to sales. In regards to this, Machine Learning (ML) research field offers an attractive approach to learn models *i.e.*, user behavior, revenue, and cost models; such models that can be trained offline, from log data already collected by most Web sites. ML trained models can be leveraged later to make real-time predictions of future behavior; thus, be applicable for automating infrastructure management according high-level policies.

This thesis contributes to the *self-configuration* of Web infrastructures according to profits by providing: *i)* a methodology for predicting Web session revenue; *ii)* a methodology to determine high response time effect on sales; and *iii)* a policy for *profit-aware* resource management, that relates server capacity to QoS and sales. The approach leverages ML techniques on Web datasets, and it is divided into three contributions:

First, a methodology to produce a revenue model for Web sessions, which is trained from past HTTP navigational information. The resulting model can be used to tell apart, the type of user and whether a session will lead to purchase starting from the first click. We show how such information can be leveraged to provide a per-class QoS and admission control when scalability is not possible, resources are limited, or it is not cost effective. Second, a per-application method to determine high response time effect on sales. Here, we introduce a non-intrusive technique to characterize response times and predict the volume of sales that would be lost at each service response time during different hours of the day. Using such technique, we can quantify the impact of different response times to sales and automatically set the best performance targets. We show how user behavior changes according to date and time, and what are the different user satisfaction thresholds per application. And third, a policy is presented for *profit-aware* Web infrastructure management. This policy leverages revenue and costs metrics from the online classification of sessions and predicts the impact of each possible response time target for a workload. We show how such economical metrics enable *profit-aware* resource management, allowing the *self-configuration* of cloud infrastructures to determine an optimal number of servers. In this way, hosting costs for an incoming workload are minimized under a variety of conditions *i.e.*, server costs, performance targets, available capacity, budget-constraints, and high-level policies of business expectations.

Our approach is tested on custom, real-life datasets from a Ecommerce retailer representative of the aforementioned Web scenario[1]. The datasets contain over two years of access, performance, and sales data from popular travel Web applications. The results obtained show how ML trained, user behavior and server performance models can be built from offline information to determine how demand and supply relations work as resources are consumed. This thesis contributes to the current state-of-art in Web Infrastructure management, as well as provides several insights applicable for improving Autonomic infrastructure management and the profitability of Ecommerce applications.

[1]The presented work derives from a technology-transfer collaboration with a Online Travel and Booking Agency. For which, we have introduced a custom module in their production systems to generate detailed application performance datasets. The datasets are used both in the experiments of this thesis and in the company's production systems to monitor and improve the performance of their application.

Contents

Abbreviations

AUGURES	the name for the Thesis prototype implementation
QoS	Quality-of-Service
UX	User Experience
B2B	Business-to-Business, *i.e.*, B2B Web Services
IaaS	Infrastructure-as-a-Service (Cloud Computing)
ML	Machine Learning, a subfield of Artificial Intelligence
NB	Nïve-Bayes, a simple classifier based on Bayes' rules
J48	C4.5 decision-tree based classifier
M5P	M5 Model trees and rule based classifier
OTA	Online Travel Agency
RT	Response Time
CR	Conversion Rate, ratio of buyers
CDF	Cumulative Distribution Function
PDF	Probability Density Function

Chapter 1

Introduction

1.1 Motivating scenario

Over the last years, high-speed, Internet access has become commodity both at home and work in many countries, with numbers reaching 91% in the US [1], similar numbers in Europe [2] at the workplace, and increasingly in mobile devices [3]. High speed, ubiquitous, Internet access is changing the way users interact with websites, their expectations in terms of performance —response time— and patience levels [4] to slowness or crashes. A current consumer report by Forrester Research [1], shows that users expect Web sites to load faster than in previous years; that about 23% of dissatisfied online shoppers attribute their dissatisfaction to slow Web sites, while 17% to crashes or errors.

Web industry leaders such as Google, Bing, AOL, and Amazon have been releasing results on how performance affects their business: Google reports that by adding half a second to their search results, traffic drops by 20% [5]; AOL reports that the average pageviews can drop from 8 to 3 in the slower response time decile [6]; Amazon reports that by adding 100ms, sales drop by 1% [7]. Furthermore, Google has announced [8] that it takes into account response time in their page ranking algorithm affecting positioning on search results, a major income source for online retailers. Web site performance has become a key feature in determining user satisfaction, and finally a decisive factor in whether a user will purchase on a Web site or even return to it.

At the same time, with the Internet at around two billion users [9], Web applications are becoming richer and more dynamic to enhance User eXperience (UX) and privacy,

they also become more resource-intensive on the server side [10]. Moreover, events such as world news and promotions, social linking *i.e.*, *flash crowd effect*, and increasing popularity can create a capacity surge within seconds. These short-term variations in the workload and rapid Web site growth call for automated policies of infrastructure configuration [11] to reduce the complexity of managing systems. This situation makes crucial the understanding of Web workloads for devising cost effective infrastructures, preventing denial of service, improving users QoS across the application, and ultimately the success of the Web site [11].

Due to the rising complexity of these systems, an attractive solution is to try to make the system components able to manage themselves. The Autonomic Computing research area, in its essence aims for *self-management*, and to free system administrators from operation details while the service runs at top performance [12]. However, the large number of potential visitors to a site makes scalability and capacity planning still a manual, complex, and costly task, while system overload incidence is growing along [11, 13]. A recent survey of 1,780 data center managers in 26 countries by Symantec [14] shows that data center management is still increasing in complexity, and that reducing the cost of managing data centers and Web applications is one of their most critical objectives.

From a business perspective, the most important metric for an Ecommerce application is profitability [15]. However, *conversion rates*, the ratio of users that reach a certain goal such as buying a product on the site are decreasing —less than 2% of visits result in a purchase on most sites [16, 17]. A low *conversion rate* is influenced by factors including affiliation programs, lowering advertising returns, changes in user habits such as comparing different sites at the same time, and meta-crawling. For example, Kayak.com and similar meta-crawlers present the user the best results gathered from several sites, thereby lowering the visits to each site and the conversion rate for those searches. Therefore, Web sites have to increasingly support more users, that expect a high QoS, for less revenue. This scenario opens a new range of challenges to understand how resources are consumed in the Web, and how to relate demand with supply and to sales [18]. Under this situation, Machine Learning techniques offers an attractive approach to learn from log data collected by most Web sites, revenue and cost models of user, sales, and server behavior; that can be leveraged to automate infrastructure management and increase profits.

In recent years, the Cloud Computing paradigm has been rapidly adopted to host Web applications due to its inherent cost effectiveness [19, 20]. It has also proven effective in scaling dynamically the number of servers according to simple performance metrics and the incoming workload for *Infrastructure-as-a-Service* (IaaS) consumers. However, not understanding the workload and how users and applications behave under load —the resource consumption, may result in an ineffective or even destructive scaling configuration *i.e.*, when requests come from automated crawlers. Additionally, for applications that are able to scale horizontally [21], system administrators still need to make manual decisions about the different parameters that affect the business results of their applications such as: what is the best performance goal in response time for each application? What metrics for dynamic scaling will warranty user satisfaction and high sales? What is the maximum number of servers that the company may afford on peak hours or surges, and for how long? What would be the effect, in business terms, of limiting the resources and degrading its performance slightly to reduce the bill of the hosting?

1.2 Thesis Approach

To answer the questions about dynamic scaling presented in the previous section, this thesis contributes to the *self-configuration* of Web infrastructures according to profits by providing: first, a methodology for predicting Web session revenue, second, a methodology to determine high response time effect on sales, and third, a policy for *profit-aware* resource management, that relates server capacity to QoS and sales.

The approach leverages Machine Learning (ML) techniques to learn from past log data revenue and cost models that enables Cloud resource provisioners *self-configure* themselves to the most profitable IaaS configuration according to the incoming workload, available resources, and high-level policies of business expectations.

The approach it is divided into three contributions detailed below:

C1.) A methodology to produce a ML model for predicting Web session revenue. Based on the analysis of Web server logs, to classify the type of user at each Web request and predict the intention of the current visit *e.g.*, its intention to purchase a product (Chapter 4).

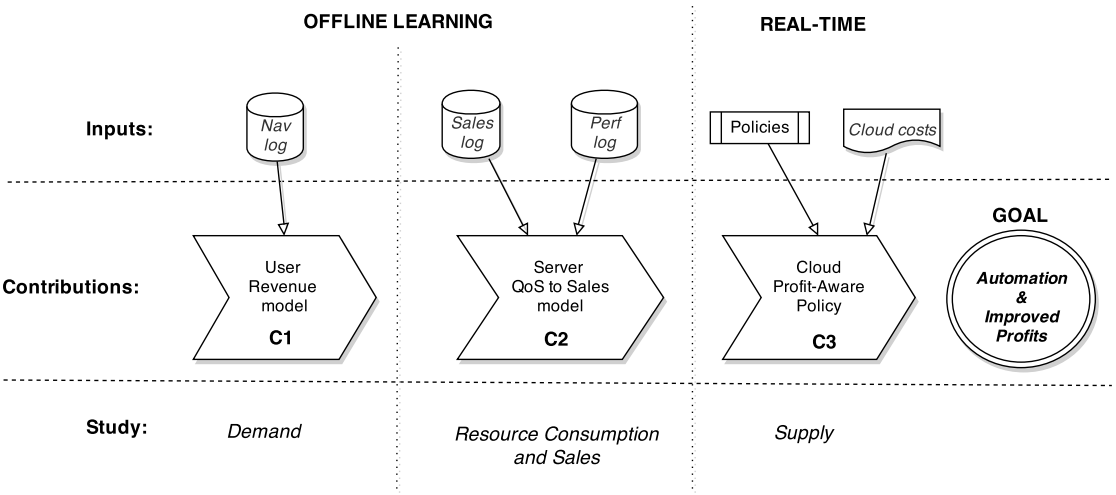


FIGURE 1.1: Overview of the thesis contributions

C2.) A methodology to determine high response time effect on sales. By performing a detailed workload characterization identifying the causes of performance degradation, quantifying the effect of server load and response time degradation to sales and user satisfaction. Producing a QoS-to-Sales model (Chapter 5).

C3.) A dynamic policy for *profit-aware* resource management, by producing economical metrics from *C1*, *C2*, along with the current Cloud costs. Where the metrics are integrated in an autonomic resource provisioner prototype, *AUGURES*, that maximizes profits for incoming Web workloads according to system defined policies *i.e.*, server costs, performance targets, available capacity, budget-constraints, and high-level policies (Chapter 6).

A diagram of the general process followed by the thesis is presented in Figure 1.1. Where the first contribution (*C1*) uses as input a user navigation log, and studies the *demand* of the website. The second contribution (*C2*), uses the input of *C1*, a sales log, and a performance dataset and studies how server resources are consumed by the demand, and how QoS affects sales as response time increases. Both *C1* and *C2* represent the offline learning phase of the thesis. While the third contribution (*C3*), leverages the models produced in *C1* and *C2*, along with system defined policies and the current Cloud costs, to produce an online, *profit-aware* policy for server resource allocation optimizing the *supply*. The three contributions are complimentary to reach the goal of the thesis: automation and improved profits for Web infrastructures.

Obtained results have led to publications following the three main contributions, as well as several insights applicable for improving autonomic infrastructure management and the profitability of Ecommerce applications. The contributions are presented in detail the following section.

1.3 Contributions

1.3.1 Predicting Web session revenue

Web sites might become overloaded by certain events such as news, events, or promotions, as they can potentially reach millions of users. Scaling their infrastructure 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, most infrastructures become stalled and throughput is reduced. 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 [22, 23] are used to keep a high throughput in terms of properly finished sessions and QoS for limited number of sessions. However, as most users navigate anonymously, by denying access unclassified, excess users, the website loses potential revenue from customers.

To overcome this situation, the **first contribution** of the thesis proposes a novel approach that consists in learning, from past data, a model for anonymous Web user behavior from a real, complex Web site. The approach consists in using the Web server access 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 it will generate, which in our case essentially depends on whether it will make a purchase and if the user might be an automated crawler.

The learning method combines static information *i.e.*, time of access, URL, session ID, referer; and dynamic information, the Web graph of the path followed by the user, in order to make predictions for each incoming Web request. 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, even in the case of a severe overload. The learner is also trained to predict if the user might be a Web scraper or content-stealing crawler to ban such requests to avoid non-user Web Services costs (see 6.7.5).

Results from experimentation show that the method can be used to tell apart, with non-trivial probability, whether a session will lead to purchase from the first click. Using models of user navigation, we have been able to show that in overload situations we can restrict the access to a Web application to only a proportion of all the demanding customers while only reducing the revenue that they generate by a factor significantly lower. In this way, 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.

Using the proposed technique 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; these should be mostly non-buying and crawler sessions, while it admits most buying ones. The second scenario is that in which keeping extra servers running has a quantifiable cost; in this case, one could try to group buying sessions a small number of servers, possibly shutting down those other servers that would produce little or no revenue *e.g.*, for crawler and bot traffic, or to provide differentiated QoS per server.

The work performed in this area has resulted in the publications listed below, as well as the extensions and collaborations found in [20, 24–26]:

- Nicolas Poggi, Toni Moreno, Josep Lluis Berral, Ricard Gavaldà, and Jordi Torres. **Web Customer Modeling for Automated Session Prioritization on High Traffic Sites**. *Proceedings of the 11th International Conference on User Modeling*, pages 450–454, June 25-29, 2007

- Nicolas Poggi, Toni Moreno, Josep Lluis Berral, Ricard Gavaldà, and Jordi Torres. **Automatic Detection and Banning of Content Stealing Bots for E-commerce**. *NIPS 2007 Workshop on Machine Learning in Adversarial Environments for Computer Security*, December 8, 2007

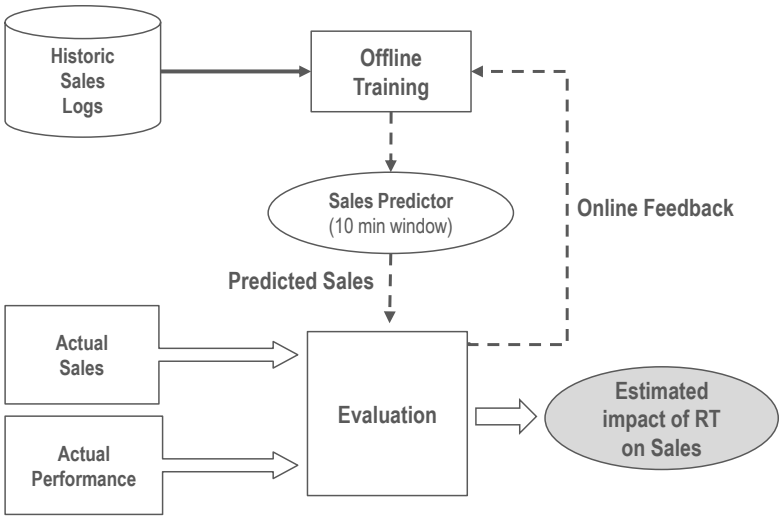


FIGURE 1.2: Sales prediction process

- Nicolas Poggi, Toni Moreno, Josep Lluis Berral, Ricard Gavaldà, and Jordi Torres. **Self-Adaptive Utility-Based Web Session Management**. *Computer Networks Journal*, 53(10):1712–1721, 2009. ISSN 1389-1286

1.3.2 Response time effect on sales

Web site performance has become a key feature in determining user satisfaction, and finally a decisive factor in whether a user will purchase on a Web site or even return to it. However from a business perspective, *conversion rates*, the fraction of users that reach a certain goal such as buying a product on the site are decreasing —less than 2% of visits result in a purchase on most sites [16, 17]. Furthermore, as competition, affiliation, lower advertising returns, and new user habits tend to lower the conversion rates the marginal gains for each visit, in this way, having to increasingly support more users, that expect a high QoS, for less revenue. This situation makes crucial the understanding of Web workloads for devising cost effective infrastructures, preventing denial of service, improving users QoS across the application, and ultimately the success of the Web site [11].

During the preliminary workload characterization of the supplied datasets (see Section 3.2), we have found that the full effect of response time degradation can be hidden by the fact that peak load times can coincide with high conversion rates, *e.g.*, when a higher fraction of visitors have the intention to purchase. To overcome this effect, the **second contribution** of the thesis introduces a novel method for studying what is the

total volume of sales lost in an online retailer due to performance degradation without modifying its application in production.

The proposed technique starts with a study and characterization of the sales log, to identify user purchasing patterns for the different applications of the site, by time and date, as well as seasonal trends from a 7 years sale history dataset. Then, leveraging Machine Learning techniques constructs a model of sales that allows for an accurate prediction of expected sales in short time frames. The model is then used to contrast actual sales with expected sales over time, and determine the impact in sales of overload periods with QoS degradation over the 2 years performance datasets for each of the different applications of the site. An overview of the process is shown in Figure 1.2.

Results from this contribution show that the user's tolerating *response time* thresholds are higher for most applications of the site that from previous literature, especially industry reports. Where our response times are in line with Miller's 1968 work on "Threshold Levels of Human Interaction and Attention with Computers" [30] . Furthermore, we have found that user tolerating times are different for each application, *e.g.*, the *events* application has exclusive content that cannot be purchased in online competitors, making it more inflexible to performance degradation than other applications *i.e.*, *flights*, that has multiple competitors.

Having different conversion rates and thresholds per application poses new challenges for dynamic, per-application QoS management during the day. The process to obtain response time thresholds for each application enables online retailers set QoS below the point where sales start to be affected by the application's *response time*. Where resulting values can be applied on autonomic resource managers to optimize the number of servers and reduce infrastructure costs in cloud computing environments. Most importantly, optimizations should not be made to accommodate all load, but to provide the best QoS when conversion rates are higher, generally at peak loads. Considering the current trend in Web ecosystem to observe lower conversion rates, online retailers will progressively support more traffic for less direct revenue by visit, increasing the importance of optimizing the number of servers without sacrificing sales.

The work performed in this area has resulted in the publications listed below.

- Nicolas Poggi, David Carrera, Ricard Gavaldà, Jordi Torres, and Eduard Ayguadé. **Characterization of Workload and Resource Consumption for an Online Travel and Booking Site**. *IISWC - 2010 IEEE International Symposium on Workload Characterization*, December 2 -4, 2010

- Nicolas Poggi, David Carrera, Ricard Gavaldà, and Eduard Ayguadé. **Non-intrusive Estimation of QoS Degradation Impact on E-Commerce User Satisfaction**. In *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, pages 179–186, 2011. doi: 10.1109/NCA.2011.31

- Nicolas Poggi, David Carrera, Ricard Gavaldà, Eduard Ayguadé, and Jordi Torres. **A methodology for the evaluation of high response time on E-commerce users and sales**. *Information Systems Frontiers*, pages 1–19, 2012. ISSN 1387-3326

1.3.3 Policies for Profit-Aware resource management

A problem that arises with dynamic server provisioning or *elastic scaling* as it is referred by Amazon Web Services (AWS) [32], is deciding when to scale in number of servers and to what number. Web applications can be served at different response times depending on the number of concurrent users by Web server. One of the main obstacles to automate dynamic scaling is selecting the appropriate metrics for scaling. In AWS this is especially a problem, as the only usable metric by default is CPU utilization [33], which for Web applications is not a good indicator of QoS or server load as the CPU is rarely a bottleneck [10]. Furthermore, not understanding how an application behaves under load, or what the true limiting factors of the application are, may result in an ineffective or even destructive auto scaling configuration [33] *e.g.*, when malicious or spurious bot traffic creates the load. Under this situation, system administrators still need to make manual decisions about the parameters that affect the business results of their applications such as: what is the best performance goal in response time for each application? What metrics for dynamic scaling will warranty user satisfaction and high sales? What is the maximum number of servers that the company may afford on peak hours or surges, and for how long? What would be the effect, in business terms, of limiting the resources and degrading its performance slightly to reduce the bill of the hosting?

As third and **final contribution**, to answer the questions of dynamic scaling, we propose a profit-driven policy to automatically find the best performance targets and server configuration that will maximize profits. The approach is based on custom economical metrics from user behavior and server performance models, by extending the results of the previous contributions. We apply the technique into an autonomic resource provisioner prototype, AUGURES, to maximize profits for the incoming Web workload in our production datasets under a varying of conditions *i.e.*, server costs, performance targets, available capacity, and high-level policies.

The prototype uses as inputs a server capacity model to determine the maximum concurrent sessions per server to offer a specific response time, and also the current Cloud costs. From the online classification of sessions and by predicting the impact of each possible response time target for a workload, we are able to produce revenue and costs metrics. We then apply these metrics in the prototype by re-running the workload of the available datasets. In addition, Web sessions are also classified by predicting their revenue potential for the site from previously learnt navigational models in order to provide a finer-grain of optimization. We then compare the profits to be obtained under different response time thresholds referenced by the literature, and present our own strategy to vary the thresholds along the day.

On the one hand, AUGURES outperforms the baseline policy of maintaining two-second response time as performance target in profits. On the other hand, we also show that if the company had a policy of keeping the lowest possible response time for users, this policy would reduce profits as it will require 40% more servers without producing significantly more sales. Our profit-aware policy changes the target response times along the day according to the expected conversion rates. The effect of session admission control was also studied, and applied to the classification of unwanted scraper bot traffic, which in the case of the presented dataset and prediction technique prevented a significant portion of total requests. Session admission control can improve profits, as it saves the costs of extra servers in our experiments.

Results from our prototype implementation enables *profit-aware* resource management allowing the *self-configuration* of IaaS to an optimal number of servers. In this way, potentially reducing hosting costs for an incoming workload following high-level policies of business expectations.

The work performed in this area has resulted in the following publications:

- Nicolas Poggi, David Carrera, Eduard Ayguadé, and Jordi Torres. **Profit Oriented Fine-Grain Web Server Management**. *Technical Report: UPC-DAC-RR-2013-60*, November, 2013
- Nicolas Poggi, David Carrera, Eduard Ayguadé, and Jordi Torres. **Profit-Aware Cloud Resource Provisioner for Ecommerce**. *Sent for consideration*, 2013

1.4 Collaboration and datasets

The presented work derives from a technology-transfer collaboration with a popular Online Travel Agency (OTA), Atrapalo.com. We have introduced a custom module in their production systems to generate detailed application performance datasets. The datasets are used both in the experiments of this thesis and in Atrapalo.com production systems to monitor and improve the performance of their application. The Ecommerce scenario of Atrapalo.com is presented along Chapter 3 and the datasets in Section 3.2.

As an additional contribution, results from this thesis had led Atrapalo.com to make important changes in their infrastructure to avoid high response times, especially at peak times, producing positive results. Also, to reduce unwanted bot traffic, saving in server resources and extra Web Services costs.

1.5 Thesis structure

The rest of this thesis is organized as follows:

- Chapter 2 presents review of the current state-of-art and related work.
- Chapter 3 presents the Ecommerce scenario and execution environment of the modeled application, the provided datasets, preliminary work to process the datasets, and a characterization of the workload.
- Chapter 4 the first contribution: Predicting Web session revenue
- Chapter 5 the second contribution: Response time effect on sales

- Chapter 6 the third contribution: Policies for Profit-Aware resource management
- and finally, Chapter 7 the conclusions and future work

Chapter 2

State-of-Art

The following chapter presents the current state-of-art as well, the relevant related work of the thesis.

2.1 Autonomic Computing

Due to the rising complexity of these systems, an attractive solution is to try to make the system components able to manage themselves. This can be solved using proposals from the Autonomic Computing research area [36], that draw in an enormous diversity of fields within and beyond the boundaries of traditional research in computer science. The essence of Autonomic Computing is *self management*, and to free system administrators from operation details while the service runs at top performance. Figure 2.1 summarizes the framework and approach of Autonomic Computing.

This thesis is within the framework of Autonomic Computing. The objective is to provide metrics that relates server capacity to QoS and sales, to be used as a foundation layer of adaptive self-configuration according to the workload, available resources, costs, and business objectives. This trend towards more autonomic Web applications has two major implications on workload characterization. On the one hand, user modeling techniques must be enriched and adapted to use the data generated and logged by dynamic websites, and to capture relevant features in order to properly account for the actual behavior or the user in the site. On the other hand, the increased demand for CPU processing and other resources in this type of applications presents a scenario

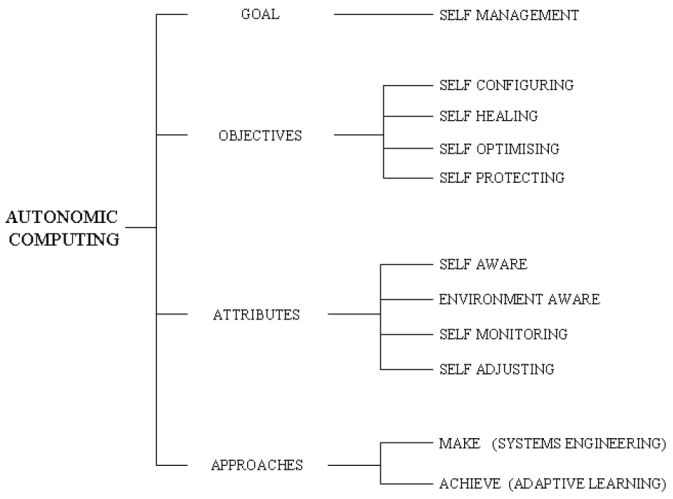


FIGURE 2.1: Autonomic Computing summary

where workload characterization can be applied in order to make a more efficient use of the available resources.

2.2 Session-based Admission Control

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 of users that try to access the site is out of their reach.

Server overload has another undesirable effect, especially in Ecommerce environments where session completion is a key factor. As shown in Figure 2.2, which shows the number of sessions completed successfully when running with different numbers of processors, when the server is overloaded only a few sessions can finalize completely. Consider the great revenue loss that this fact can provoke for example in an online store, where only a few clients can finalize the acquisition of a product [22].

To prevent loss in sales due to overloads several techniques have been presented such as Session-Based Admission Control (SBAC) systems [22, 23, 37–39] used to keep a high throughput in terms of properly finished sessions and QoS for limited number of sessions.

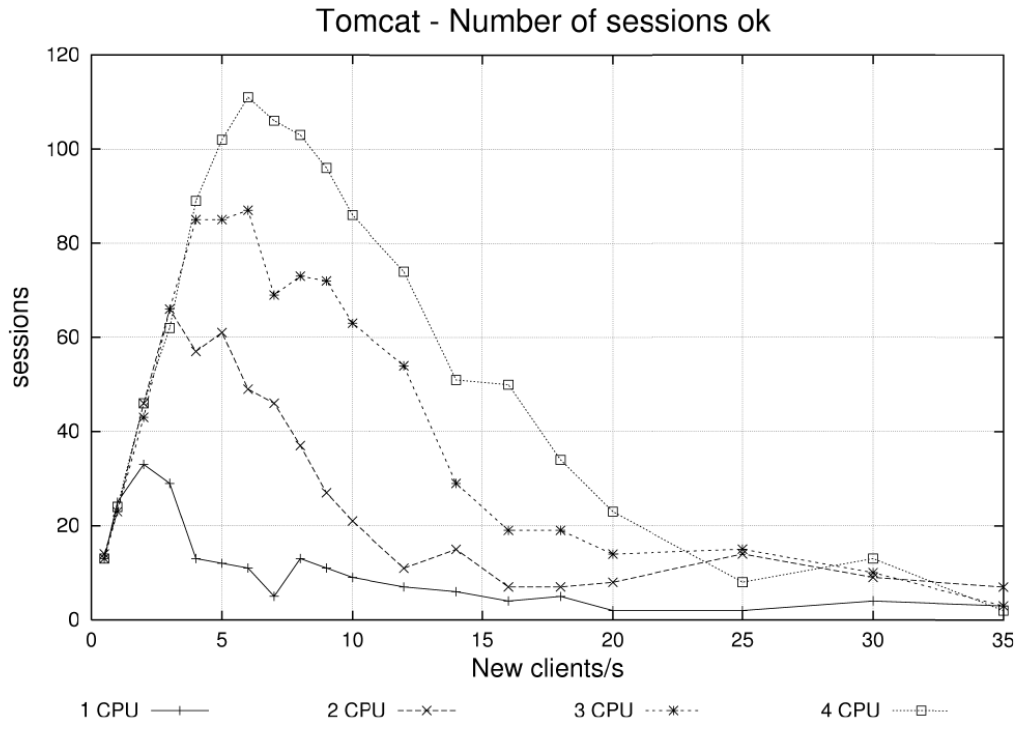


FIGURE 2.2: Completed sessions as new clients arrive to the system for different numbers of processors

Guitart *et al.* in [22], presents a session based adaptive mechanism for SSL as negotiating a new SSL connection involves generating a set of asymmetric keys for the new session; while resuming one is basically authenticating the user, which is less resource intensive. In case of an overload, Guitart *et al.* propose to prefer resumed connections over new ones as: it is less resource intensive and resumed sessions have been for longer on the site, thus closer to an eventual purchase. Figure 2.3 show the sustained throughput delivered by the server under unmodified workload conditions in replies per second, and user session completions respectively. While Abdelzaher *et al.* [39] describe performance control of a Web server using feedback control theory to achieve overload protection, performance guarantees, and service differentiation in the presence of load unpredictability.

The reviewed approaches present mechanisms to prevent session throughput degradation, allowing servers to work at top performance even when there are more users than resources. However, by denying access to excess users, the Web site still loses potential revenue from customers. In Chapter 4, the first contribution, we are proposing to go a level further, automating session prioritization, to pick most profitable sessions on overloads, using a prediction process based on Web mining.

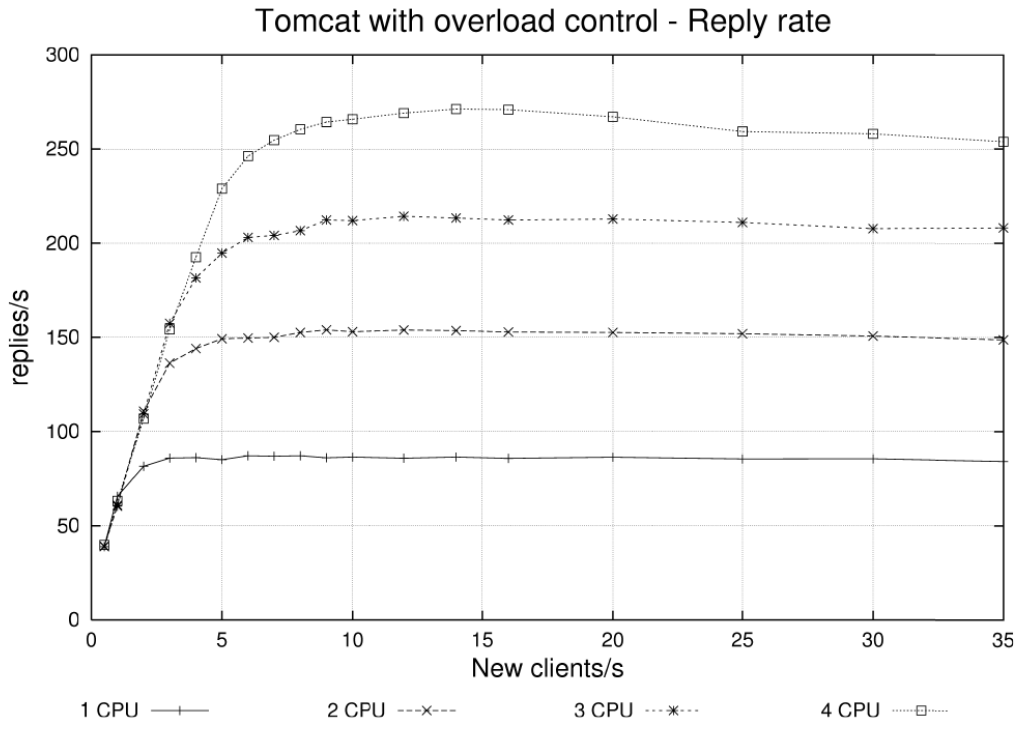


FIGURE 2.3: Throughput with overload control with different numbers of processors

2.3 Web Mining

Most online businesses rely on free Web analytic tools to inform their Web marketing campaigns and strategic business decisions. However, these tools currently do not provide the necessary abstracted view of the customer's actual behavior on the site. Without the proper tools and abstractions, site owners have a simplified and incorrect understanding of their users' real interaction patterns on the site, and how they evolve.

In the context of Web mining, there are few published studies based on real Ecommerce data, as the datasets presented in this thesis, mainly because companies consider Web logs as sensitive data and for privacy and competitive concerns. Web mining is the technique to extract information primarily from Web server log files. The main purpose of structure mining is to extract previously unknown relationships between Web pages. Sharma *et al.* [40] classifies Web mining into usage, content, and structure Web mining. While this thesis falls within the scope of structure Web mining, most of the literature in this topic focus on recommendation systems and Web personalization [41, 42].

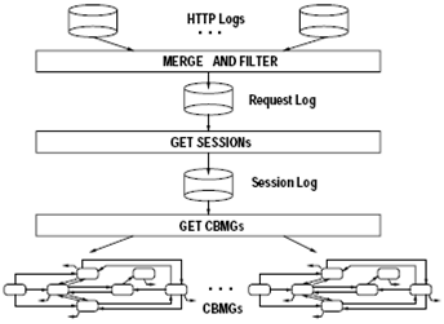


FIGURE 2.4: Methodology to obtain a CBMG

2.3.1 User Modeling

Works on Web user behavior prediction and profiling [43–46] 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 as well as path analysis [47, 48]. 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 or external transactions might be too costly in the case of a miss or user exit.

Other authors [43, 49] focus on dynamic content adaptation, where the page adapts to the type of user; it could include images, colors an even products or links. The user prediction approach presented in Chapter 4, to predict user intentions could be applicable for dynamic content adaptation too, as we would be characterizing users; although this thesis focuses on resource management and QoS.

2.3.2 Customer Behavior Model Graphs

Customer Behavior Model Graphs (CBMG) can be used to provide an abstracted view on Web navigation. Menascé *et al.* [50] propose to build the CBMG using k-means clustering algorithm, creating a probability matrix for the possible path transitions from a state. Figure 2.4 presents the methodology to produce a CBMG.

CBMG represents an improvement over Session Based Admission Control (SBAC), frequent buyers are expected to have a higher combined probability of quickly transitioning

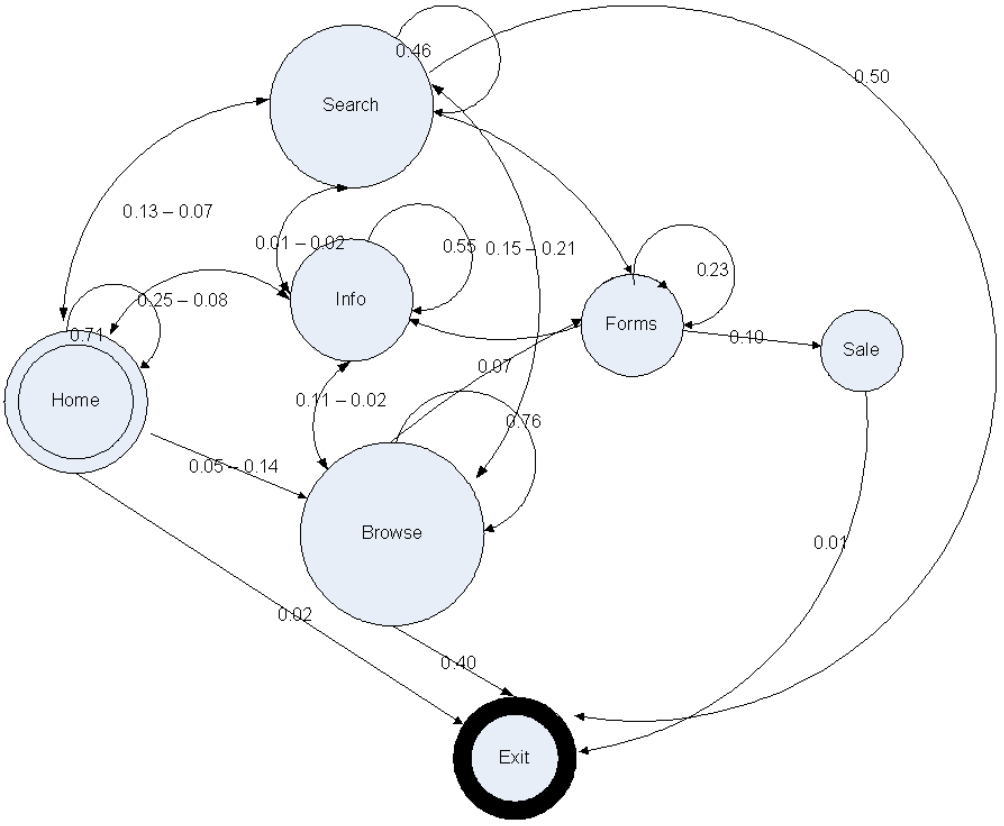


FIGURE 2.5: CBMG for our datasets

into the final stages of the session and the unconvinced users may leave the site from any state. In terms of session valuation, CBMGS allows user classification *i.e.*, frequent buyer, regular visitor, and information seeker, and provides a framework for predicting the value of a particular session [15]. While Totok *et al.* [51] presents a similar and later approach to Menascé's, performing predictions from CBMGs. CBMGs *i.e.*, [50] are similar to the dynamic part of the approach of the first contribution of this thesis, presented in Section 4.3.2 — we use the closely related Markov chain model (See Section 2.4.2). Figure 2.5 shows a CBMG basing on Menascé's technique presented from our datasets.

In contrast to works on CBMGs, in this thesis, we do not focus on predicting the user's next click, but seek to extract the most relevant critical paths occurring in the site and build the process model of customers. In particular, we are interested in the important events and workflows that lead to a user buying a product, presented briefly in the next subsection.

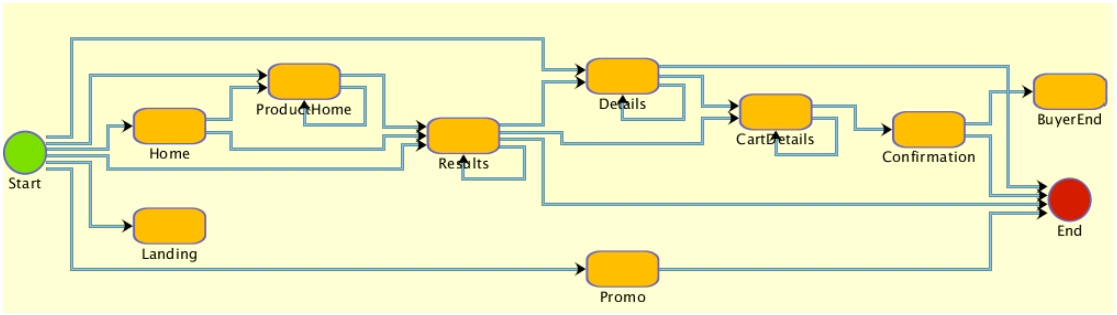


FIGURE 2.6: Abstracted process model of Web navigation including customer interactions

2.3.3 Process analytics

Unlike Web analytics [52], process analytics is concerned with correlating events [53], mining for process models [54–56], and predicting behavior [57]. Process mining algorithms are designed to extract the dominant behavior observed and filter out noise to keep the resulting mined process manageable.

In our preliminary work [26], we proposed treating a user's Web clicks as an unstructured process, and use process mining algorithms to discover user behavior as an improvement to CBMGs. The mined process model captures the causality and paths of user interactions that lead to certain outcomes of interest, such as buying a product. Such insights can be difficult to extract from traditional Web analytic tools. Figure 2.6 presents a sample process model from the Atrapalo dataset. More details in Section 3.3.

2.4 Machine Learning

Machine learning [58] is a broad subfield of artificial intelligence concerned with the design and development of algorithms to allow computers "learn". The McGraw Hill technological dictionary defines Machine Learning as:

> "The process or technique by which a device modifies its own behavior as the result of its past experience and performance".

Machine Learning is a well-established and studied science, in this project we are not proposing new algorithms, but to implement them for the applicability of resource management.

On the topic of cost based resource management, one approach by Littman *et al.* [59] uses Naïve-Bayes for cost classification and a Markov chain feedback approach for failure remediation. Other works such as [60] also take into account costs and resource allocation; in contrast with previous approaches, in this thesis we are focusing on the actual revenue that is lost by denying access to purchasing users, and not resource allocation costs as the mentioned works.

2.4.1 WEKA

WEKA [61] (Waikato Environment for Knowledge Analysis) is an open source project written in Java at the University of Waikato, which implements numerous machine learning algorithms. WEKA is currently the defacto application for Machine Learning in academic and even some commercial projects (under a commercial licence). For our experimentation, we have used WEKA to compare the performance of the different algorithms adapted to our data.

In WEKA, a *predictor* model is trained using a specially formatted *training dataset* that should contain the most relevant available variables to predict sales. After training the *predictor* with the training dataset, a *test dataset* with the same format is used to perform predictions. The predictor reads the test dataset, ignoring the *class* —the unknown variable, in our case the number of sales— if present, and according to the training and the algorithm used, outputs a prediction of the class.

Other tested algorithms for the data include:

- Nïve-Bayes, a simple classifier based on Bayes' rules
- BayesNet, a more sophisticated Bayesian classifier
- LogisticRegression, using linear functions for classification
- C4.5 (WEKA's J48), a decision-tree based classifier
- RandomForest, a classifier of several decision-trees simultaneously
- NearestNeighbors, a lazy similarity-based technique
- M5P, numerical, M5 Model trees and rules
- RepTree, numerical, fast decision tree learner.

2.4.2 Markov-chains

Since most machine learning algorithms are not well adapted to dealing with variables that are themselves sequences, and the path followed by a user in a session being a sequence, we have explored the performance of using Markov-chains. A Markov-chain is a series of states where each future state is conditionally independent of every prior state, given the current state. Markov-chains have been previously used for Web Mining, Sarukkai [47] used an implementation for Markov-chains to dynamically model URL access patterns for link prediction and obtaining navigational tours. More details on how we use Markov-chains in Section 4.3.2.

2.5 Workload and resource characterization

Evaluation of Web application resource consumption requires realistic workload simulations to obtain accurate results and conclusions. In the context of Web workload analysis, there are few published studies based on real Ecommerce 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 [62]; and file popularity following Zipf's Law [63][62]. Some works such as [64] have studied logs from real and simulated auction sites and bookstores; there are no studies that we know about intermediary sites, like the one we obtained, where most of the information comes from B2B providers which have a different behavior as shown in Section 3.4.

Recent studies have performed similar workload characterizations as the one presented in Section 3.4. In [65] Benevenuto et al. characterize user behavior in *Online Social Networks* and Duarte et al. in [66] characterize traffic in Web *blogs*. Previous work on the characterization of collaborative Web applications was conducted in [67]. Although both the *blogosphere* and the OTA application used in our work are similar in the sense that they are user-oriented, user behavior is different in these scenarios. Moreover a more detailed analysis is presented in this thesis, as the site is multi-application, and applications are further subdivided to perform a separate analysis by day, type of request, applications, as well as the resource consumption by each.

Other works come to the same conclusions as presented in Section 3.4, but from the point of view of generating representative workload generators, such as [68] for static workloads and [69, 70] for dynamic applications. None of these studies looked in detail at a complex multi-product Web 2.0 application in production of the scale of what is studied in this thesis, with over 2 billion requests for 2 year of datasets. Similar work was conducted in [71] but following a black box approach for the enterprise applications, not exclusively Web workloads, meaning that they show no information about the nature and composition of the studied applications. Their work was data-center provider oriented, while our work is application-centric.

While [72–75] discuss about the need of stationarity of arrival processes to study and characterize workload burstiness, in [76] the authors work on non-stationary properties of the workloads to improve performance prediction. In Section 3.4.2, we have leveraged the techniques presented in some of these studies to characterize workload burstiness in stationary periods, but have not extended this work.

Few studies present both a characterization of workload and resource consumption. In [77] Patwardhan *et al.* perform a CPU usage breakdown of popular Web benchmarks with emphasis on networking overhead, identifying that network overhead for dynamic applications is negligible, while not for static content. In [78] Ye and Cheng present a similar characterization of resource utilization as the one presented here, but for *Online Multiplayer Games*. In this thesis we also cover how response time affects user behavior in session length and number of clicks, validating results from previous studies [79, 80].

2.5.1 Response times

Response time effect on user behavior has been studied as long as 1968, where Miller [30] describes the "Threshold Levels of Human Interaction and Attention with Computers". In 1989, Nielsen [81] re-validated Miller's guidelines and stated that the thresholds are not likely to change with future technology. These thresholds being: 0.1 to 0.2 seconds, instantaneous; 1-5 seconds the user notices the delay but system is working; and 10 seconds as the threshold for user attention. Other authors [79, 80] adhere to what they call the 8 seconds rule, where no page should take longer than 8 seconds to reply. While the APDEX standard [82] sets the threshold for frustrated users at 4 seconds.

Menasce *et al.* [11], perform an assessment on how QoS of Ecommerce sites plays a crucial role in attracting and retaining customers, where they propose a workload manager using predictive queuing models of computer systems based on a *hill-climbing* method. Authors perform admission control based on the response time of the system. However, their work is based on a popular but synthetic workload, the TPC-W, and does not have sales into account in contrast to this thesis.

2.5.2 Workload prediction

The topic of workload prediction has been extensively studied, including works such as [83–85]. In [83] Zhang *et al.* make the use of statistical analysis and queuing theory to propose a framework for traffic prediction and monitoring, COMPASS. Andreolini *et al.* in [84] show that in the context of Web based systems it is inappropriate to predict workload solely on load resource monitors. For this reason, they propose a two-phase strategy that first tries to obtain a representative view of the load trend; then applies load change and load prediction to the trend representation to support online decision systems. In [85], authors propose to use time series analysis as it offers a broad spectrum of methods to calculate workload forecasts based on history monitoring data. In a previous work [86], we have used the CUSUM (cumulative sum) algorithm as well as Machine Learning to perform workload prediction. In this thesis we do not aim to provide a new method for prediction, but rather focus on Cloud resource management.

2.6 Cloud Computing

Cloud computing is primarily driven by economics [19, 20] and it has been rapidly adopted due to its inherent cost effectiveness. For this reason, there are a number of studies on the topic of market-based resource allocation for Grid and Cloud computing. Most noticeable on scheduling mechanisms including: FirstPrice [87], FirstProfit [88], and proportional-share [89]. However, as Cloud computing has first evolved out of Grid computing, where jobs to be executed where mostly *batch*, rather than having real-time requirement. Therefore, most of these works therefore targeted *supercomputing* workloads with a fixed number of resources and Service Level Agreements (SLA).

Currently cloud platforms are used to host almost any type of application, and in particular it has become the most popular platform for new Web applications presenting a very different, transactional based, workloads.

Frequently scalability terms used to define cloud activities include: *scale out*, *scale-down*, and *scale-up*. *Scale-out* refers to the ability to scale horizontally by adding more nodes to the system. *Scale-down*, the opposite, to reduce the number of servers in a cloud defined cluster. While *scale-up* refers to the vertical scalability, adding more resources to a system such as memory or CPUs and can include replacing instances to a different one with better resources. As with any architecture, scaling-up has limits depending on available resources, while scaling-out is not usually a limit on large cloud deployments. However there can still be some limitations on application scalability, cloud resources offering, and budget constraints.

Since applications cannot scale indefinitely, Majakorpi [21] presents a new metric to measure the *quality of elasticity* of applications to horizontal (scale-out) vertical scalability (scale-up). For this work we are considering perfect scalability, however the user prediction techniques can be used to compensate to resource limitations while still keeping a high profit as presented in our previous works [28, 29].

2.7 Utility-functions

Utility functions are well known as a form of preference specification [12]. Therefore, utility functions have been applied on the topic of profits and customer satisfaction. Chen *et al.* [90] argues that existing SLA based approaches are not sufficient to address performance variation and customer satisfaction. They present a model of customer satisfaction, leveraged by a utility-based SLA to balance performance and the cost of running services and two scheduling to make trade-offs between profit and customer satisfaction. In [7], Mazzucco proposes the use of utility functions to optimize auto-provisioning of Web servers.

In previous works [29, 91], we have also evaluated the use of utility functions to set SLA targets, however this work differentiates in two ways: first, we base our approach on real user satisfaction, while [90] based their approach on an synthetic satisfaction metric

based in utility; second, our experimentation is done to a production Web Ecommerce workload, while [90] is on video encoding workloads.

2.8 Profit-aware provisioning

Work related to improving the performance of Ecommerce servers has largely focused on providing better response times and higher throughput [22]. However, the most important performance metric from a business perspective is *profitability* [15, 28, 51].

Profitability of clouds has recently got attention; Liu *et al.* [92] propose a cost-aware approach to maximize the net profits that service providers may achieve when operating distributed cloud data centers in multi-electricity market environments. By capturing the relationship between SLA, cost on energy consumption, service request dispatching and resource allocation. Choon Lee *et al.* [19] introduces a pricing model to Cloud scheduling algorithms where profits are bound to a customer SLA specification. While most works [11, 19, 87, 92, 93] have focused on SLAs as a measure of optimization, in practice, from the IaaS consumer and business perspective, it is the final user satisfaction what finally leads to profits in Online businesses as we show along this thesis.

Xi Chen *et al.* [93] propose a profit-driven provisioning technique to maximize profits in the Cloud, by using auto-regressive performance model to predict the expected demand curve and determine when and how much resource to allocate and to maximize profits based on SLA specifications and costs generated by leased resources. While similar to our approach, there are also main differences: first, the workload employed for the study is the popular FIFA '98 dataset; composed of static requests, without customers in the workload; and second, it also uses static SLAs to calculate profits, while have demonstrated that the QoS that Web users expect varies during the day and is different by application, being one of the primary benefits of our approach.

Hong *et al.* [94] explores two different techniques to reduce further the cost of IasS consumers. They propose to scale vertically down servers according to the load to pay for cheaper servers when it is not needed (ShrinkWrap) and a combination of reserved vs. on-demand servers to benefit from lower costs of reserved instances. In our work we consider same-size instances, the approach provided by the authors is complementary to our technique, and can potentially drive costs further down. However, from the traces

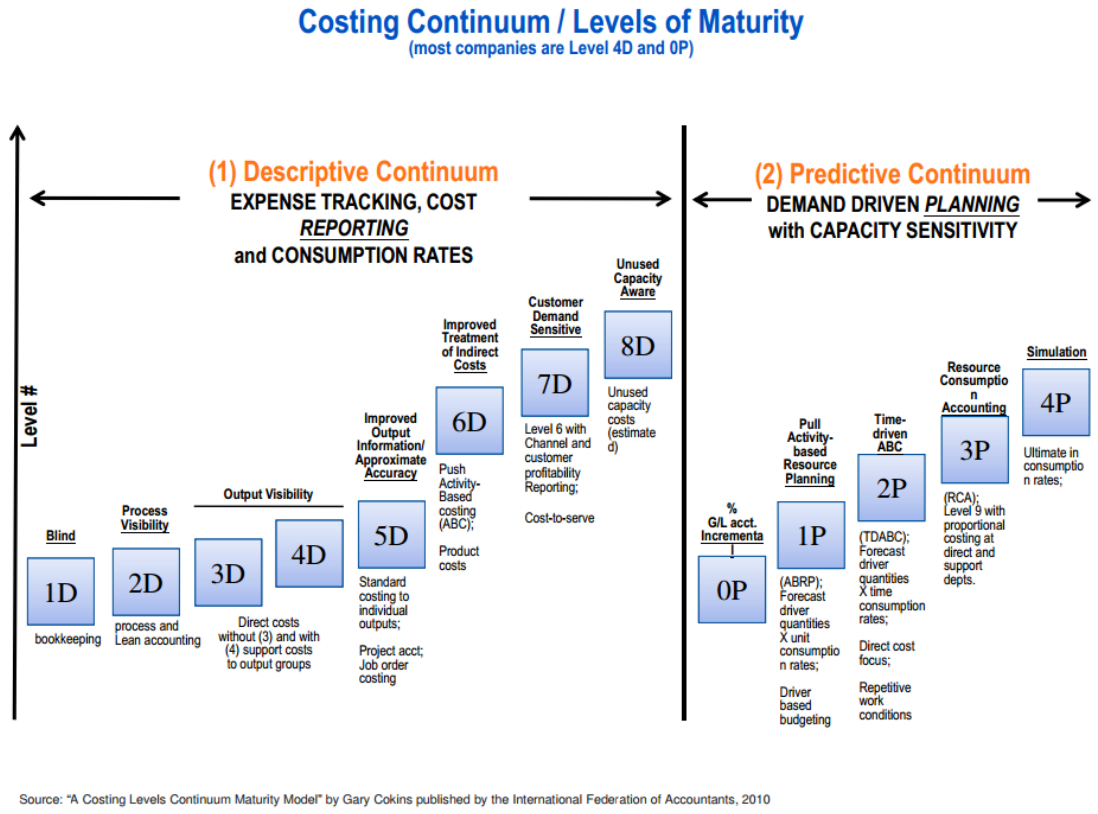


FIGURE 2.7: Costing Continuum

we have, night traffic is very low compared to day traffic, so having all-day reserved instances and hinders profitability in our scenario presented in the next chapter.

2.8.1 Costing Continuum

From a business accounting perspective, Cokins *et al.* [18] define 12 levels of *cost maturity*; the general overview is presented in Figure 2.7. Each costing technique level is distinguished by how more effectively an organization incorporates both demand with supply and the nature of cost as resources are consumed. In which the last 5 levels of *cost maturity* are defined as *demand driven planning with capacity*; while Level 11 representing resource consumption accounting and Level 12 simulation. Cokins *et al.* argues that organizations that plan at Level 12 have arguably attained the highest level of cost planning, as simulation can lead to optimization. This thesis deals first, with the accounting of consumed resources by Web visitors, while we use this information to

later perform simulations based on a real-life workload. The aim of this thesis is to automate *demand driven planning with capacity* for online businesses, this way obtaining the maximum level of cost planning, optimizing resources according to profits.

Chapter 3

Preliminary study of the scenario and datasets

Evaluation of Web application resource consumption requires realistic workload simulations to obtain accurate results and conclusions. The presented work derives from a technology-transfer collaboration, where we take the case of Atrapalo.com, a top national Online Travel Agency (OTA) and Booking Site. Atrapalo.com features popular Ecommerce applications found in the Web and its representative of the OTA industry. We have introduced a custom module in their production systems to generate detailed application access and performance datasets captured over a period of more than 2 years. Each dataset, featuring several million HTTP requests, from 2011 to 2013. The datasets are used both in the experiments of this thesis, as well for the company's production systems to monitor and improve the performance of their application. We have also been given access to a 7-year sales dataset of the OTA, including time and volume of each operation.

Section 3.1 describes the preliminary study of the Ecommerce scenario of Atrapalo.com and of Online Travel Agencies (OTA) in general. An in-depth analysis is necessary as there are not previous literature on intermediary Web sites such as the one presented in this thesis. As well as to understand the driving factors and particularities of the industry in order to assess its current problematic, and to obtain suitable and more applicable results. It also presents some domain knowledge how the inventories of the

different products work, the reasons for the variability of the traffic, as well as presenting computing infrastructure and the general concepts of the trade *e.g.*, Conversion Rates.

Section 3.2 presents the different datasets used throughout the thesis, and how to classify the large number of URLs to make them usable for analysis and prediction.

Section 3.3 presents some general techniques to be able to tell apart customer sessions from regular visits. Three different techniques are presented as preliminary work to the fist contribution in Chapter 4.

And Section 3.4 presents a characterization of the workload. The analysis is necessary to understand how the workload is composed and to extract the main characteristics and insights, that motivated the second contribution detailed in Chapter 5.

3.1 Ecommerce Scenario

3.1.1 Online Travel and Booking in figures

Online Travel Agencies (OTA) are a prominent sector in the online services market: according to the 2008 Nielsen report on Global Online Shopping [95], Airline ticket reservation represented 24% of last 3 month online shopping purchases, Hotel reservation 16%, and Event tickets 15%; combined representing 55% percent of global online sales in number of sales.

3.1.2 Products

Online Travel Agencies such as Atrapalo.com, present a wide range of *products i.e.*, flights, hotels, cars, restaurants, activities, vacation packages (or 'trips'), and event booking. For this purpose they rely on a wide range of technologies to support them: dynamic scripting, Javascript, AJAX, XML, SSL, B2B Web services, high-definition multimedia, Caching, Search Algorithms and Affiliation; resulting in a very rich and heterogeneous workload. While these technologies enhance users' experience and privacy, they also increase the demand for CPU, databases, and other resources on the server side. Furthermore, as Web applications become more resource-intensive and the large number of potential visitors on the Web, system overload incidence is growing along [13].

The workload of OTAs is not only dependent on their own user base and promotions, but to promotions of the products through the wholesalers. For example, when an airline company lowers their ticket prices, they generally mass announce their offers, without much notice probably for competition reasons. Another common situation is when a certain event opens their ticket sales such as a concert or football match, and users flock to the offering sites; this case is analogous to long queues on opening days for tickets. Visits to travel sites might not depend only on their popularity but to current year season, holidays, search engine ranking (SEO), linking and the proximity of an event, such as a concert (see Section 5.6.1). The variability of the traffic creates a bursty workload, making workload characterization and modeling crucial for devising cost effective infrastructures, preventing denial of service, and improving users Quality of Service (QoS) across the application.

These events make capacity planning and resource management a complex task: it is difficult to predict when these user peaks are going to occur before they start happening. For an OTA, these situations of large traffic fluctuations are frequent and not serving users is a loss in revenue.

For Atrapalo.com, some of the above mentioned products inventories are maintained internally *e.g.*, restaurant booking, some are completely external *e.g.*, flights, and some other products *e.g.*, hotels are mix of internal and external providers. Access to the external providers is performed via B2B Web services, which causes QoS to be dependent on external sites for some operations. The company itself is also a B2B provider for some customers and meta-crawlers, acting as their provider via a Web Service API.

As an example of external B2B requests, to offer search results, *e.g.*, flight availability, several providers are queried and results are offered according to different algorithms of product placement in a resource intensive operation. As some of this searches are costly —not only in terms of resources— but by contract of the Global Distribution Systems (GDS) services.

The application relies on state-of-art caching rules, to reduce request times and load generated by the external transactions. Although the company's main presence and clientele is in Europe, about 20% percent of the visits are from South America where it has offices, and few more visits from the rest of the world. Mobile traffic is growing

rapidly, representing about 10% of the total user traffic at time of writing. It is important to remark that the site is a multi-application Web site. Each *product* has its own independent application code base and differentiated resource requirements, while sharing a common programming framework.

3.1.3 Conversion Rates

In Internet marketing, the conversion rate (CR) can be generally defined as the ratio between the number of *'business goal' achievements* and the number of *visits* to a site. In the scope of this thesis, a *goal* is achieved when a customer purchases a product offered by the OTA during a browsing session. The CR is one of the most widely used indicators of business efficiency on the Web. A high CR indicates that a high number of visitors purchase on the site on each visit, while a low CR indicates that most visits use server resources without returning value in terms of direct revenue. Many factors can affect the conversion rate, *e.g.*, type of product, content, brand, SEO ranking, affiliation, availability, and QoS measured in response time. Values for CRs are different for each Web site, and are part of their marketing strategy and business nature; however values should remain within the same range over time for similar products on different sites.

A low CR is influenced by factors including affiliation programs, changes in user habits such as comparing different sites at the same time [10], and meta-crawling. For example, *Kayak.com* and similar meta-crawlers present the user the best results gathered from several sites, thereby lowering the visits to each site and the CR at least in 10 times. The next subsection presents content-stealing bots, another reason of lowering CRs.

3.1.4 Content Stealing Bots

Content stealing in the Web is becoming a serious concern for information and Ecommerce websites. In the practices known as *Web fetching* or *Web scraping* [96], a stealer bot simulates a human Web user to extract desired content off the victim's website. Stolen content is then normally stripped of copyright or authorship information and rendered as belonging to the stealer, on a different site. The incidence of *Web scraping*

is increasing for several reasons: the simplicity to simulate human navigation, the difficulty to tell bots apart, the grey area on its legal status [96] and, most importantly, the profitability of the business.

As in the case of spam and domain redirection, Web scraping is part of the breed of most common Internet abuses. During the years, Web scraping has shifted from simple plagiarism to profit-making using Web advertising techniques. Non-profit sites such as the Wikipedia have been particularly prone to scrapping for plagiarism [97], moreover, Ecommerce sites are increasingly being affected directly by their actions.

Meta-crawling sites such as Kayak and Travel Fusion, *scrap* travel websites simulating real user navigation in order to compare results from different sites. This situation creates the necessity to automatically identify and sometimes ban such crawlers [28], as the conversion rates for this requests are more than 10 times lower than direct visits.

3.1.5 Caching

One commonly used strategy across the industry is to heavily rely on caching to prevent excessive searches from meta-crawlers and speedup results for users. However, caching can result in changed availability, referred as *bookability* or even in increased prices, where the user finds about these changes in the moment of booking, sometimes after completing forms, compromising user satisfaction. So caching has to be used effectively to lower response times while providing accurate results.

Looking at the interaction between the OTA and external information providers from the datasets, it has been observed that the probability of accessing an external provider follows a Binomial distribution with parameters $n = 1,125,969; p = 0.1306$ for the *flights* product. For those requests that did involve access to an external site, Figure 3.1 shows the CDF of the time spent waiting and processing the information provided by the external source. As it can be derived from this information, caching techniques are effectively used for this application, avoiding in many cases (more than 75%) the cost of gathering information from external providers. For the cases in which accessing an external provider is required, the process is usually completed in less than 20s. Further discussion on CDFs in Section 3.4.

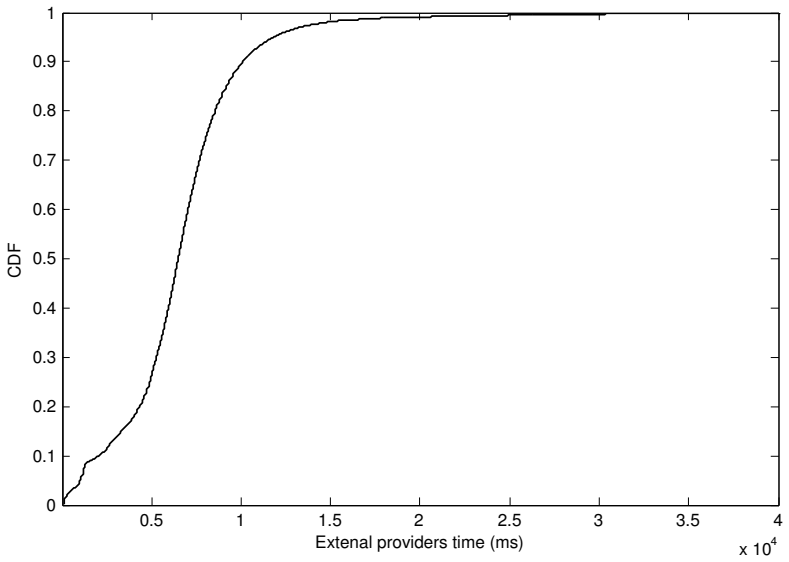


FIGURE 3.1: CDF of external providers time

We have identified that for every requested dynamic page, about 13 static resources are accessed in average. There are several caching mechanisms in place, for Web content: there is a reverse-proxy running *Squid* caching static content generated by the dynamic application ; there is a per-server caching Web template caching; distributed memory key-value storage (Redis), database query cache and scheduled static HTML page generators.

3.1.6 Computing Infrastructure

At the time of writing, Atrapalo.com infrastructure is composed of about 40 physical servers running Debian GNU/Linux, connected via Gigabit switches, including: a set of redundant firewall and load-balancer servers acting as entry points and SSL decoders; about 15 dynamic Web servers; 5 static content servers; 2 redundant file servers, 8 high-end database servers including masters and replicas running MySQL; plus auxiliary servers for specific functions such as monitoring and administrative purposes.

Web servers characterized in this study are equipped with 2x dual core Xeon processors, 16G RAM and SATA hard disks. The Web application runs on the latest version of PHP on Apache Web servers; load is distributed using a weighted round-robin strategy by the firewalls according to each server capacity. Database servers are equipped with 2x dual core Xeon processors, 32G RAM and SCSI hard disks with RAID 10 configuration. Access to databases is balanced by using DNS round-robin rules for replica slave servers,

most of the READ/WRITE strategy and data consistency is performed in the application itself, which also caches some queries in memory and local disc. Static content such as images and CSS is mainly served by Content Distribution Networks (CDNs), to reduce response time on the client end; the rest of the static content is served by servers running *ngineX*; notice that static content is not part of this study.

There are several caching mechanisms in place, for Web content: there is a reverse-proxy caching static content generated by the dynamic application running *Squid*; there is a per-server caching Web template caching; distributed memory key-value storage, database query cache and scheduled HTML page generators. The dataset file use in this study is produced by the PHP dynamic application.

3.2 Datasets

Evaluation of Web application resource consumption requires realistic workload simulations to obtain accurate results and conclusions. This section presents the different datasets provided by Atrapalo.com from their production environment to characterize their most relevant features on Web user sessions and their resource usage. We categorize datasets in: *Sales datasets*, which present sales over the last 7 years; and *Performance datasets*, custom generated performance logs of 2 years of detailed per request performance measurements.

3.2.1 Performance datasets

Performance datasets are used for evaluation of high response time effects both in user behaviour and sales; they were produced through existing probes in the PHP dynamic application and provided by Atrapalo.com. The datasets used for experimentation consists of two years of requests to Atrapalo site, from February 2011 to February 2013. The data is composed of over 2.1 billion requests, representing around 480 million sessions. Where the average server response time from all of the dynamic requests is 1.36 seconds, the average database service time is 0.13 seconds, and for requests that involved external Web services the average is 0.64 seconds per request.

These performance datasets are generated by the PHP dynamic application; at the end of each executing script code was added to record regular HTTP data: access time, URL, referring URL, client IP address, HTTP status code, user-agent (type of browser), replying server. Data from the application itself: total processing time, exact user session id, real page processed by the server (some URLs might be ambiguous or perform different actions), accessed product(s), type of request (regular, AJAX, Administrative, API, batch, etc), CPU percentage, memory percentage and peak memory, CPU time both in *system* and *user mode*, total database time, total external request time. As well as current server load (*load average*) and number of opened Apache processes.

3.2.2 Sales datasets

For the purpose of this study, we were given access to sale history datasets for the OTA's different *products*. For each *product* we have the exact date and time for each purchase that was made. However, we did not have given access to sales amounts or margins, just the moment of a user sale. Sales datasets range from 01/01/2007 to 31/01/2013, comprising a period of close to 7 years. There is a great variation of sales between products and times of the year due to seasonality effects (see Section 5.6.1). Vacation products suffer from great variation of sales and visits according to high and low seasons, day of the week, or the proximity of a holiday; while ticket sales varies according to the availability of the event on competing sites and the limited availability, sometimes causing rush periods.

Sales datasets are used studying conversion rate variation, as well as our sales predictor in Chapter 5. Sale volumes for these datasets are not reported due to confidentiality limitations.

3.2.3 Other Datasets

We also had access to the Apache logs and monitoring system access for the days covered in the performance dataset and other random weeks of 2009, 2010 and 2011 used in previous works. These auxiliary logs have been used to validate and explain obtained results from the workload. Notice that the studied datasets do not include pages that were cached by the reverse proxy or by the user's browser.

3.2.4 URL classification

The first challenge when analyzing Web logs is to classify the URLs of the site. Website URLs used to indicate the location of the file on the server directory system. The server path gave an indication of the classification, and the page name and type of content for that page. With time, static pages were replaced by dynamic pages produced by Web applications and several URL rewriting techniques were implemented for security, search engine optimization, and localization. With dynamic page generation, it became more convenient for security and organizational reasons to point all requests to a single page with internal rewrite rules and redirects, and the concept of the *front-controller* was developed. The *front-controller* presents a single point of entry to the Web application, and with it, URLs left being directly related to files on the server. At first, most front controller implementations relied on query string parameters (variables after the question mark in a URL) to indicate the *action* that the page would perform and its corresponding output. However, mainly for Web search engine positioning, URLs were changed to be descriptive. Most Web application frameworks implement URL rewriting engines to generate "friendly URLs", and delegate control to the application of what content is served by each URL. With Search Engine Optimization (SEO) friendly URLs, URLs started to be more descriptive to aid search engine positioning, making URLs more unique in the process. Furthermore, current frameworks support localization of pages, so not only are URLs more descriptive, but URLs are also translated to different languages, even if the same piece of code will produce the content. Some sites even implement a mix of static page URLs, query string base for non-indexable pages, and rewrite engines, or even a combination. To exemplify this situation, Atrapalo website implements a variety of strategies including legacy application code and code written by different teams. For the dataset used in the experimentation, there are 949 532 unique URLs. If we take the query string out of the URL, the number of distinct pages reduces to 375 245. Atrapalo URLs are also localized, and there are a few examples of static HTML pages that point to a file in the file system.

3.2.4.1 Classifying URLs into Logical Tasks

For the dataset used in the experimentation several URL rewriting techniques were implemented, which resulted in 949 532 unique URLs. If we take the query string out of

TABLE 3.1: Classification of URLs into logical tasks

Tag	Description
Home	Main home page
ProductHome	Home page for each product
Landing	Search engine landing pages
Promo	Special promotional pages
Search	General site search
Results	Product search and results
Details	Product detailed information
Opinions	Opinions about a product
Info	Site help or general information
CartDetails	Shopping cart details
CartPurchase	Shopping cart purchase forms
Confirmation	Confirmation page of a sale
UserAdmin	User self-reservation management

the URL, the number of unique pages reduces to 375 245.

In order to extract the *action* —type of process and output of a page— from a URL in Atrapalo dataset, we had to implement the rewrite engine used for the page classification. Rewrite engines usually perform regular expression matching to URLs. In Atrapalo URLs, the first element in the URL path indicates the name of the product, such as flights, hotels, cars, or events. Each product had custom implementations of the rewrite engine and how regular expressions were performed. About 20% of the URLs did not match any regular expression, and for these URLs query string classification was performed by looking for a custom parameter "pg", which specified the page *action*. Using the query string approach we were left with 5% of unclassified URLs that were manually analyzed and classified using string search and replace.

After the URLs where translated we were left with 533 different page actions or type of pages. However some of the page names occurred only once, a problem we attribute to noise and errors in the rewrite engine implementation. We then filtered the pages that did not have more than one occurrence, and ended with 233 page names. This means that across the products of the site there were 233 different types of pages. Some of the pages serve the same logical function, such as the search page for hotels, flights or cars, or the different home pages for each product. After a manual analysis on the site structure and URLs, we decided to classify them in 14 logical types of pages detailed in Table 3.1.

Although the classification in Table 3.1 is particular to Atrapalo dataset, many Ecommerce sites share similar structures especially for sites implementing travel and booking

TABLE 3.2: Classifier Evaluation

Algorithm	**Clusters**	**Error**
SimpleKmeans	14	39.90%
EM	14	41.88%
EM	Automatic	76.93%

products. It is important to remark that through the classification of pages no data is lost. Page classification is added as extra columns to the dataset. The URL and page types are kept in the dataset, so we can later use them to filter or to extract better path predictions. The next section presents a proposal for automating page classification.

3.2.4.2 Automating page classification

Classification of types of pages into logical groups is necessary to map user clicks occurring in a website to abstracted logical tasks to be consumed both by mining algorithms and final reports to humans. We noticed while reviewing the results that many page actions had similar names. There was at least a search page per product and different types of search pages, including flightsSearch, hotelsSearch, flightsCalendarSearch, hotelsSearchCity. To aid classification, we have tested the clustering of the page names using the WEKA open source machine learning framework (see Section 2.4.1). WEKA contains several popular ready to use algorithms for classification and clustering among other tools. As we had previously decided that the classification has 14 logical types of pages, K-means clustering was our first natural choice to test, as it performs in general scenarios with known number of clusters. We have used WEKA SimpleKMeans implementation and setting the number of clusters to 14 and the "classes to clusters" evaluation option. SimpleKMeans yielded an error of 39.90% in classifying the 233 names into 14 clusters. We have also experimented with the EM (Expectation-Maximisation) algorithm both with automated and manual numbers of clusters yielding 76.93% and 41.88% of classification errors, respectively. Table 3.2 summarizes the clustering results. If the number of classifications is known, K-means clustering can reduce the manual work needed to simplify page classification.

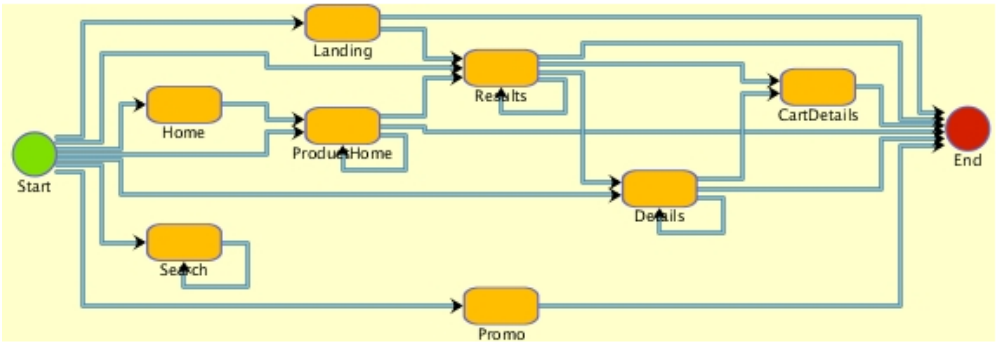
3.3 Mining for customers

This section details our preliminary experiments mining the business processes of customers in Atrapalo dataset, with the page classification from the previous section. For a full description on process mining algorithms please refer to [26]. As mentioned in Section 3.1.3, only a small fraction of visits to the site ended buying a product. The conversion rate for the site is less than 2% of the total number of visits. Having such a small percentage is a problem for most mining algorithms, as these low-frequency traces (Web sessions) will be filtered out by most implementations producing an incomplete model. In our study we present three different approaches to this problem creating three new different datasets: saturating the data set (*saturated*), clustering (*clustered*), and *biasing* toward a previously set model with the knowledge-based miner. We call the original dataset the *normal* dataset.

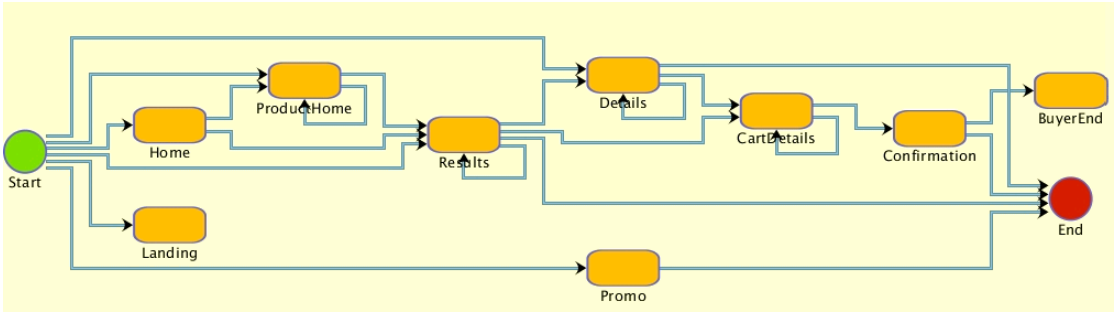
3.3.1 Saturating the dataset with customers

The first strategy to mine customer models was saturating the dataset. This entailed producing a new dataset where the percentage of buying customers is higher by removing sessions that did not purchase. We have chosen the ratio 1/3 of customers to just visitors. This ratio is chosen as customer sessions are longer in average, leaving us with and even dataset of about half of the entries belonging to customer sessions. With this ratio, we have created a new dataset including the entire customer sessions present in the normal dataset, and 2/3 more sessions from regular visits from the top of the dataset. This dataset having about 8% of the total entries of the normal dataset, but including all the purchasing sessions.

Figure 3.2 shows the resulting models by applying the knowledge based miner [26] with default noise and window parameters to the *normal* (Figure 3.2(a)) and *saturated* (Figure 3.2(b)) datasets. The general workflow of events can be seen from the figures, with the main distinction being that the *normal* dataset does not contain the *Confirmation* and *BuyerEnd* events and edges. The *CartDetails* event is present in both. This means that while there are many users that add a product to the shopping cart and see its details, few ultimately purchase the product. In these cases the buying events are being discarded as noise, while on the *saturated* dataset they are being kept. Loops can also



(a) Process model for the normal dataset



(b) Process model for the buyers saturated dataset

FIGURE 3.2: Process models for the *normal* and *saturated* datasets

be seen in both models, but the loops are from the same originating event to itself, such as users iterating over the *Results* event.

Another insight from the models is that the *Promo* event is not linked to any other event; almost all users that get to the site through a promotional page leave the site without any further navigation. On the *normal* dataset, some users from the *Landing* event get to the results. In the *saturated* dataset, however, the landing page event does not have any outbound links. The same can be observed with the *Search* event in the *normal* dataset: its only link is a self-loop. The *Search* event is not present in the *saturated* model, because it is a low frequency event and not used by most customers. We have verified that most results pages were directly reached from each product home pages. *Search* events represent the general site search feature that searches all products at the same time, and results show they are not very effective and were reported back for optimization.

FIGURE 3.3: Process model of a customer cluster

3.3.2 Clustering sessions

The next tested approach to mine for customer sessions was clustering, using a string distance algorithm to cluster imported traces. By clustering similar sessions, we can run the process mining directly on individual clusters. This feature is very helpful as clustering can help remove noise and allows the ability to mine specific customer clusters or target groups without the need to saturate the dataset.

Figure 3.3 shows the model produced by the miner to a specific small cluster of customers, representative of the most common buying process. It shows the critical path (the most important pages) for buyers on the website, and thus, the most important pages to keep optimized. It also shows that the most typical buying process consists of three main pages: *Details*, specific product information; *CartDetails*, final costs details and payment options; and *Confirmation*, the reservation confirmation page. This would mean that most buying sessions go straight to purchasing without much searching, probably performed at a previous time and different session.

The disadvantage of clustering, besides not having the complete process in the output model, is that models cannot be combined directly without manual work. The knowledge-based miner allows us to use prior knowledge, such as the model produced by clustering as shown in Figure 3.3, to assign more weight for these events and edges. This particular feature is detailed in the next subsection as a different strategy. Further details on clustering sessions can be found in [98].

3.3.3 Prior knowledge

The knowledge-based miner [26], besides being able to keep longer paths and be parameterized by the amount of noise (fitting) and window size, can use another model as prior knowledge with a tunable confidence. This feature can be used not only to mine for customer models without saturating the dataset, but also to include certain clusters or behavior, such as the effect of improving the promotional page, or a marketing campaign targeting a certain product.

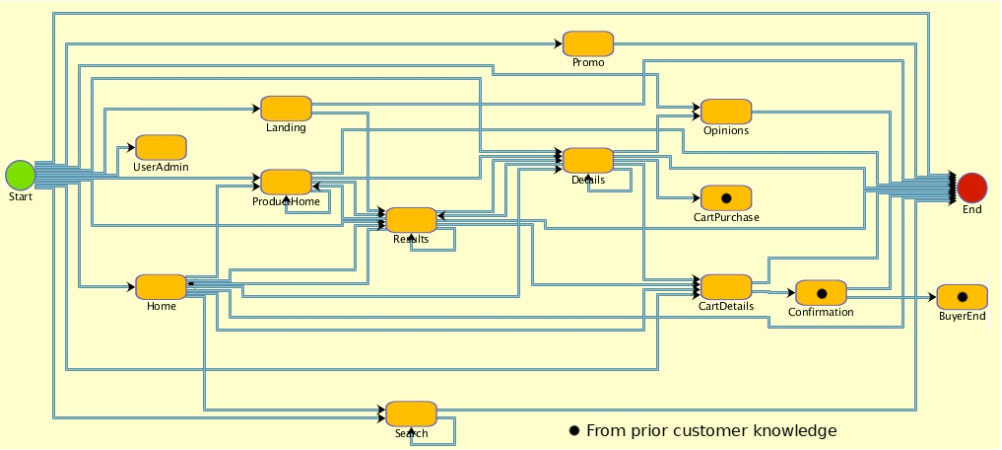


FIGURE 3.4: Knowledge-based miner process models on the normal dataset

Figure 3.4 shows both the model produced by the knowledge-based miner on the *normal* dataset, and the output when the model from Figure 3.3 is applied to the knowledge-based miner. Results are the same in both, except that when the prior knowledge is applied, the output includes the *CartPurchase*, *Confirmation*, and *BuyerEnd* events.

Figure 3.4 also shows the use of the knowledge miner parameters. Compared to Figure 3.2 it shows the *UserAdmin* event and more edges and loops between events. The reason is that both figures were executed with lower *window* and *noise* parameters. This shows how models can be abstracted and fitted using these parameters in the knowledge-based miner algorithm.

The above classification of URLs allowed Web logs to be mined for processes that represent the navigation behavior of users. Process mining algorithms are designed to extract the dominant behavior observed and filter out noise to keep the resulting mined process manageable. However, in our case study the interesting behavior—those that result in a user buying a product—seldom occur. We expect this to be the case in many Web applications. To avoid losing this behavior, we took the approach of saturating the dataset with more traces that result in the outcome of interest. This simple strategy worked well in producing a complete process model that includes both the most common behavior on the site, and also includes the behavior of users that buy a product. Since our first results and feedback, the company redesigned the general site search, improving the conversion rate of visitors using the search feature by up to 46%, and lowering the bounce rate by 22% for a particular product.

3.4 Workload characterization

This section presents the general characteristics of the performance dataset. This characterization was performed as preliminary work for the thesis, on a one-week version of the dataset. From it, many insights have taken for the realization of the contributions. It can also serve the reader as reference for the performance dataset. From the dataset we characterize user sessions, their patterns and how response time is affected as load on Web servers increases. We provide a fine grain analysis by performing experiments differentiating: types of request, time of the day, products, and resource requirements for each.

The characterization of the workload is approached from two different perspectives: firstly, the client workload pattern is studied, considering the request arrival rate, session arrival rate and workload pattern in a representative and generic one week access log. Secondly, the same one week log is studied from the point of view of resource consumption. The outcome is the complete characterization of both user access pattern and non-simulated resource consumption of a Web application. Moreover, the studied dataset presents several features not present in most Web workload characterizations, such as the dependency of external providers, database access and mix of differentiated products (multi-application site). Results from this section can support the building of a workload generator that is able to simulate the real life characteristics of complex workloads such as the one presented here.

3.4.1 Workload decomposition

Figures 3.5 and 3.6 show the traffic pattern for the studied dataset, including number of hits (Figure 3.5) and number of user sessions started (Figure 3.6) over one week, grouped in 30-minute periods. Notice that data has been anonymized through being normalized to the peak load observed for each metric. As it can be observed that a problem with the logging infrastructure caused a short period of no information that can be clearly observed in Figure 3.6.

As it can be observed in Figure 3.5, the traffic decreases over the night, until it starts growing again soon after 7am in the morning. It keeps growing until noon, when it

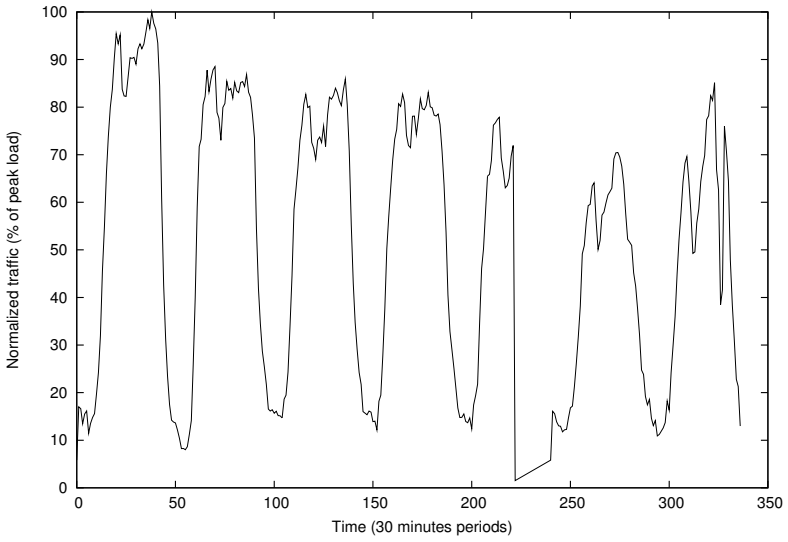


FIGURE 3.5: Traffic volume intensity (relative to peak load). - 1 week

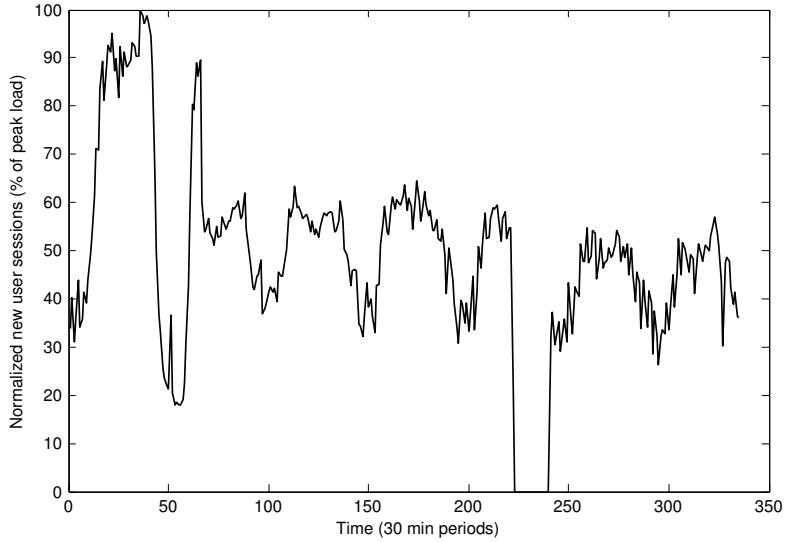


FIGURE 3.6: New user sessions intensity (relative to peak load). - 1 week

slightly decreases. Finally the workload intensity starts increasing again over the afternoon until it reaches its maximum around 9pm. Over the night, the traffic volume decreases until it finally reaches the beginning of the cycle again. Notice that client requests are conditioned by the response time delivered by the Web application (next request in a user session is not issued until the response corresponding to the previous request is not received). For this reason, we made sure that the while logs were collected no severe overload conditions took place in the Web infrastructure, but still capturing the a representative volume of traffic for a normal day in the online travel agency. We followed the same approach to characterize not client requests, but new Web sessions in the system, that is, the number of new clients connecting to the system. The relevance

i	a	b	c
1	8.297	0.002157	1.134
2	8.072	0.002325	4.232
3	0.1572	0.009136	1.542
4	0.04958	0.01732	2.356
5	0.02486	0.02197	2.045
R-Square: 0.9701			

TABLE 3.3: Variables of the Normalized Request Rate Function

of this measure, when taken in non-overloaded conditions, is that reveals the rate at which new customers enter the system. We also grouped data into 1 minute period, which can be seen in Figure 3.6. As expected, per-session data follows the same trends observed for the per-request study, but with a smoother shape.

The mean pageview for the whole week is 6.0 pages per session, with 6:48 minutes spent on the site, an average of 3.0s response time for dynamic page generation, and 8MB of RAM memory consumption. Recall that the highest traffic is on Mondays and decreases to the weekend. The opposite effect is observed on average *pageviews* as well as the time spent on the site; they both increase during the week, peaking at the weekend, from: 5.82 and 6:40 on Mondays to 6.27 and 7:31 on Sundays, pageviews and time spent respectively.

The characterization of the normalized shape of the mean request rate for a 24h period, in 1 minute groups can be done following the Sum of Sines expression found in Equation 3.1, with the parameters described in Table 3.3.

$$f(x) = \sum_{i=1}^{5} a_i * sin(b_i * x + c_i) \tag{3.1}$$

3.4.2 Workload mix and intensity

The workload is composed of several different request types, and for each pageview that the user finally sees on his browser, several dynamic requests may have been executed. In the studied dataset we have identified the following request categories: Regular user page 46.8%, AJAX 19.8%, dynamically generated Javascript 16.6%, HTTP redirect page 9.1%, Administrative 4.5%, internal scheduled batch 3.1%, API Web Service 0.04%, and

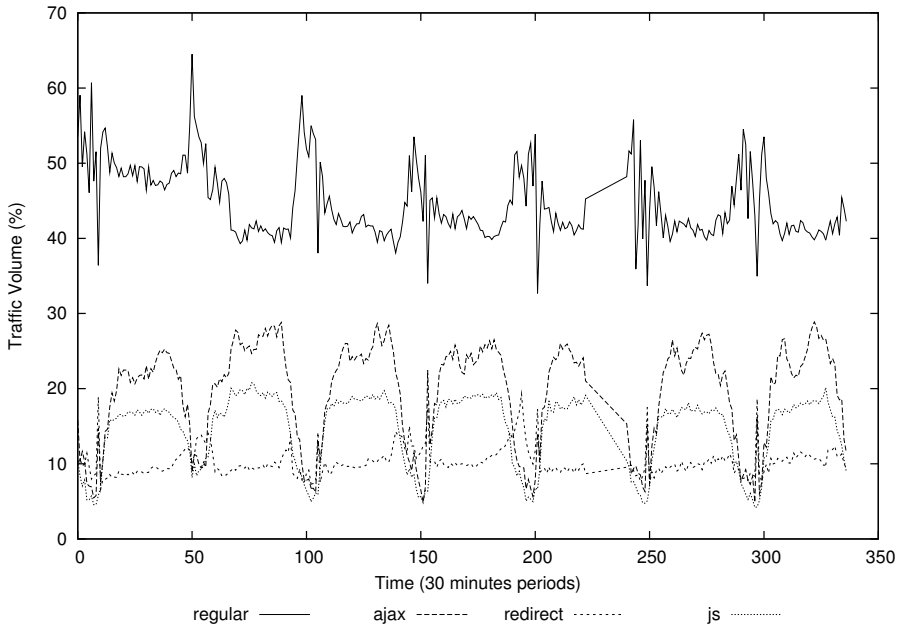


FIGURE 3.7: Traffic mix over time (transaction type) - 1 week

Extranet 0.02%. It is an important feature of this dataset (and probably other real-life logs) that less than 50% of the total dynamic requests correspond to user clicks on their browsers.

Figure 3.7 shows the fraction of dynamic traffic volume that corresponds to different types of request categories, focusing on most relevant ones: Regular, AJAX, Redirects and JavaScript contents. As it can be observed, AJAX content fraction is mainly correlated to site's load, as such traffic is usually generated by human-generated actions (*e.g.*, auto-completion of forms when indicating flight origin and destination airports during a search). During low traffic periods, basically overnight, most of the traffic is identified as regular pages. Night traffic also involved most of the internal batch and crawler activities.

A brief analysis on the number of crawler requests and meta-crawlers by analyzing the *agent* field (the reported browser type) identifying themselves as such; our findings indicate that the number of *bot* requests is about 15% of the total traffic. This is consistent with previous work on a similar dataset 3 years before [28], which identified between 10% and 15% total bot content. Even more traffic may correspond to crawlers and meta-crawlers assuming that some might simulate being a real user when accessing the site, that would show a growing trend in the proportion of automated bot traffic.

Application	Percentage
App 1	27%
App 2	17%
App 3	15%
App 5	6%
App 6	5%
Other	23%

TABLE 3.4: Percentage of the number of requests per application

Table 3.4 shows traffic distribution across anonymized products (applications) offered by the OTA. As it can be observed, almost 60% of the overall traffic come from only three applications, representing the most popular products of the company. Although each application is implemented independently, they share a common code base (*e.g.*, user logging and shopping cart). Such *common* activity is not included in the specific per-application traffic volume, and is considered as a separate application by itself, corresponding to App 3, 15% of the total requests; this distribution is site specific.

3.4.3 Session characteristics

Next step in the workload characterization is to study the per-session characteristics of the OTA visitors. Each session is started when a new visitor comes into the system, and is identified through a single identifier in the workload trace. We will look at four different session-specific characteristics: number of different products visited during the session, number of different pages visited per session, number of hits per session (notice that a hit can be initiated by a user click or by Javascript events such as auto-complete controls), and the session length. For each one of these characteristics, we construct a CDF chart as shown in Figure 3.8. Each CDF is built from the information collected during the lifetime of all the sessions started within a 30 minutes period. Recall that the completion time of a session can be much later than the end of the 30 minutes period.

We have explored 4 different time ranges for each property, selecting time ranges corresponding to 4 points of time with different traffic characteristics, including night, morning, afternoon and evening traffic. The selected time ranges are 5:00am to 5:30am, 11am to 11:30am, 4:00pm to 4:30pm, and 10:00pm to 10:30pm. It can be seen from the

Figures that all properties remain unchanged for all time ranges except for the night one.

Session characteristics are approximately the same for morning, afternoon and evening time ranges, but a clear difference can be seen for the night (5am) traffic. Notice that the OTA is international, most of the traffic comes from European countries located within the time zones with a maximum of 2h of difference. Obviously, the different characteristics of the nightly traffic comes from the fact that the many sessions are initiated by non-human visitors (bots), including crawlers and meta-crawlers. This result supports the results presented before in Figure 3.7. Daytime (10pm) CDFs can be approximated using the probability distributions and parameters listed in Table 3.5.

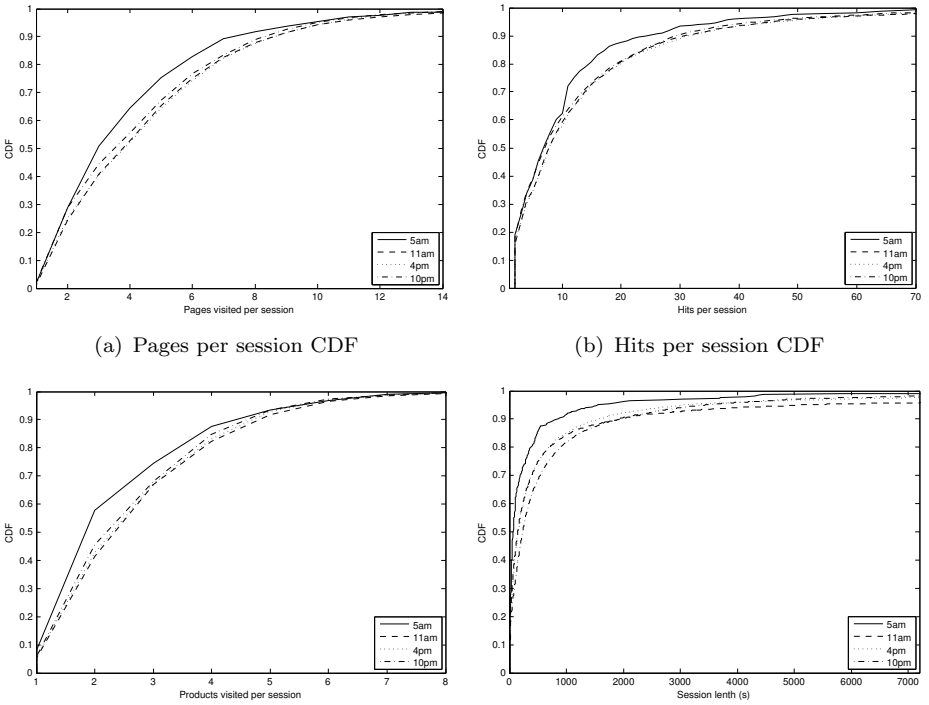
Our study concluded that 75.03% of the sessions only contained 1 hit, that is, the user only accessed 1 page, and then just quit the OTA site. This is mainly due to many visitors reaching the site through external banners that redirect them to especial *landing pages*, and many of these users do not continue browsing the OTA site after this initial page. In the building of the CDFs, 1-click sessions were excluded as we want to study customer's characteristics; 1-click sessions are included in the rest of the experiments.

Figure 3.8(a) shows number of different pages visited per session (notice that a page is a unique URL here). Most users visit few pages during a session, and they may remain in one single page running searches or browsing the OTA catalog. Some users visit up to 14 pages in one single session, but that is the least of them.

Figure 3.8(b) shows number of hits per session, with half of the visitors producing 10 or less requests to the OTA site. Notice that a request can be initiated by a user click, or by an AJAX action, such as field auto-completion in search forms. A significant percentage of visitors produce many more requests, reaching a few tenths in many cases.

Figure 3.8(c) shows number of products visited per session. As the OTA site contains several different products, each one associated to a particular Web application, we were interested in studying how many different products were visited by each individual session. It can be seen that around 50% of the customers are interested in only two different products, but in some cases 8 or even more products may be visited in one single session.

Figure 3.8(d) shows session length CDF, showing that while most visitors sessions last only a few minutes, some of them may be active for several hours. That may be explained



(a) Pages per session CDF

(b) Hits per session CDF

(c) Products per session CDF

(d) Session length CDF

FIGURE 3.8: Session properties - grouped according to session start time in 30-minute bins

Pages per Session	
Log-normal	$\mu = 1.37536; \sigma = 0.60544$
Hits per Session	
Log-normal	$\mu = 2.05272; \sigma = 1.00659$
Products per Session	
Log-normal	$\mu = 1.01541; \sigma = 0.457558$

TABLE 3.5: Per session CDF Fits

by users coming back to the OTA site quite often over a long period of time, or by the presence of crawlers that periodically poll the OTA site contents.

Finally, we look at the burstiness properties of the workload, paying special attention to the session arrival rate and its changes over time. For such purpose, we have characterized the Index of Dispersion for Counts (IDC) of the entire workload as well as for a shorter time period which presents stationary properties. The IDC was used for arrival process characterization in [72], and has been leveraged to reproduce burstiness properties in workload generators in [73]. IDC was calculated by counting sessions started in 1 minute periods In a first step, we characterized the IDC for session arrival rate for the full dataset, covering one week period. The result for this step is shown in Figure 3.9(a).

Metric	Model	Parameters
DB Time	Weibull	$a = 30141.4; b = 0.251286$
DB Queries	Generalized Pareto	$k = 0.61979; \sigma = 4.65092; \mu = -0.1$
Ext. Provider Time	Logistic	$\mu = 6.17049e + 07, \sigma = -191940$

TABLE 3.6: DB and External time CDF Fits for most popular product

In a second step we picked the stationary period shown in Figure 3.9(c), corresponding to a 500 minutes high-load period, and characterized its burstiness through its IDC, as shown in Figure 3.9(b). Both figures indicate, given the high value of IDC observed, that the workload shows a high degree of burstiness as it is expected for any Web workloads. And it remains true at both scales, including one week of processed data and a short and clearly stationary period of logged data.

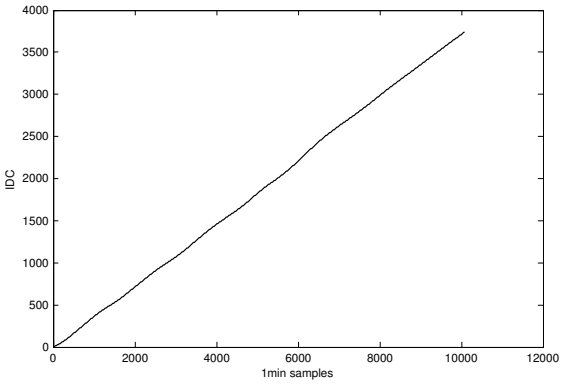
3.4.4 Resource consumption

Figure 3.10 shows resource consumption distribution across anonymized applications.

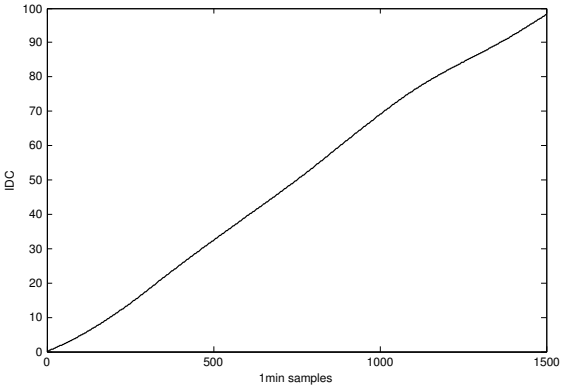
When modeling sessions and requests they also have different characteristics and resource requirements. Figure 3.11 shows the different resource percentage used by each type of requests.

In Figure 3.12 we pick the most popular product of the OTA company and characterize the interaction of its code with both the database tier and external providers of information. The characterization is done by building the CDF of each metric, what can be approximated using the functions seen in Table 3.6.

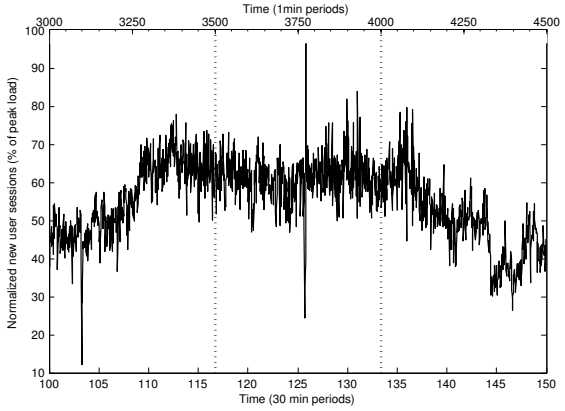
All the services related to this product require accessing at least once at the DB tier. Figures 3.12(a) and 3.12(b) show the CDF of the number of DB queries per request and the time spent per request waiting for the result of DB queries. Recall that this information corresponds only to the most popular product of the OTA. As it can be observed, 50% of the requests issue 1 or 2 queries to the DB tier, and around 80% of the requests require less than 10 queries to complete. But a significant fraction of the requests produce complex results and require a large number of DB queries to be completed, reaching more than one hundred DB requests in some cases. Looking at the time spent waiting for data from the DB tier, most of the requests exhibit just a couple



(a) 1 week period - 1 min bins



(b) Stationary period - 1500 minutes - 1 min bins



(c) New user sessions intensity (relative to peak load). - 1 week

FIGURE 3.9: Workload Burstiness: Index of Dispersion for Counts (IDC) of Initiated User Sessions

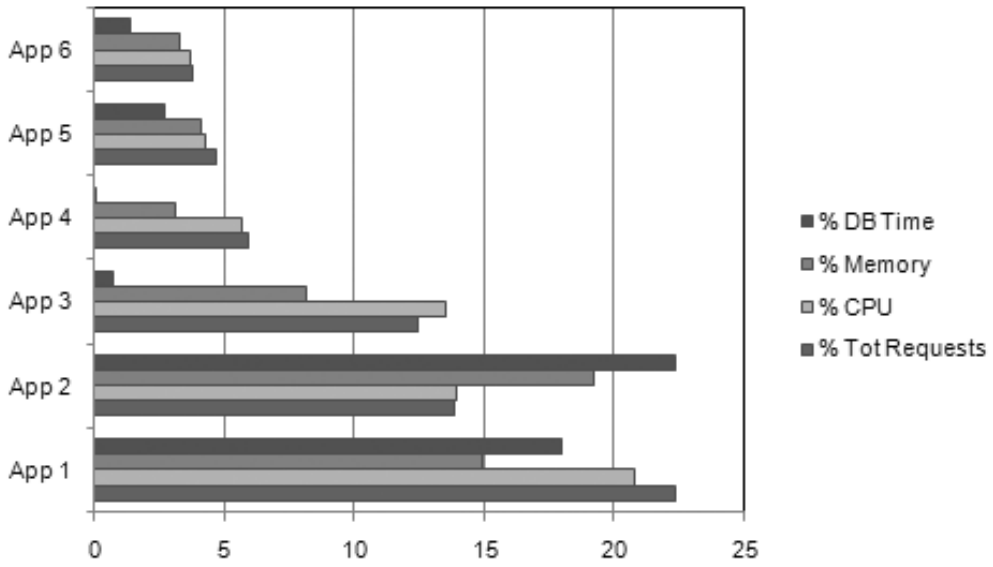


FIGURE 3.10: Percentage of Resources by Application

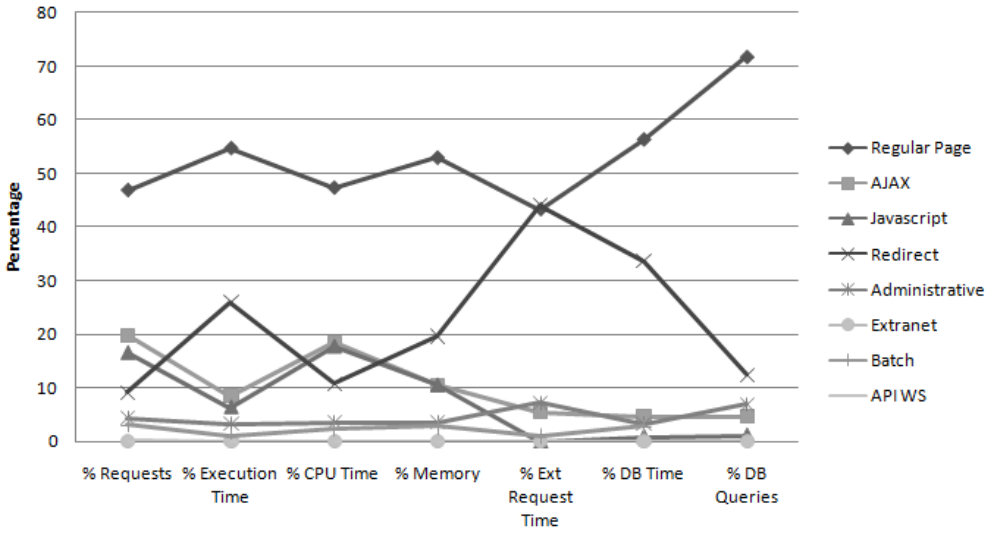
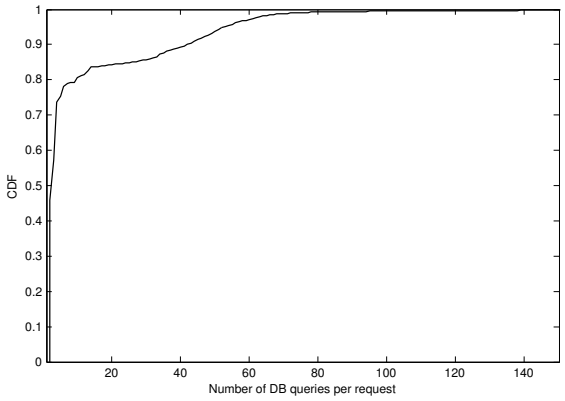


FIGURE 3.11: Percentage of resource usage by request type

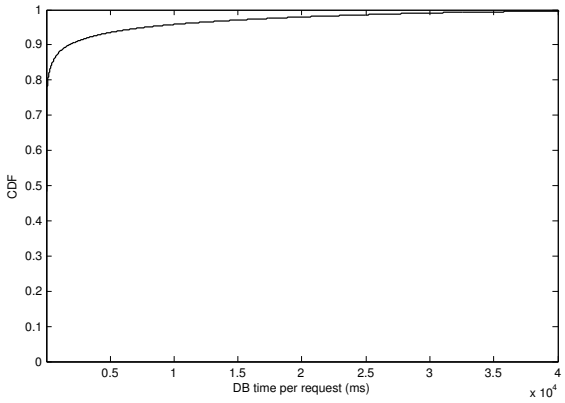
of seconds of DB query waiting time, but some cases can reach up to nearly 40s of DB time. Notice that some OTA operations may require complex optimization operations, as well as may provide a long list of results based on user search parameters.

3.4.5 Characterization results

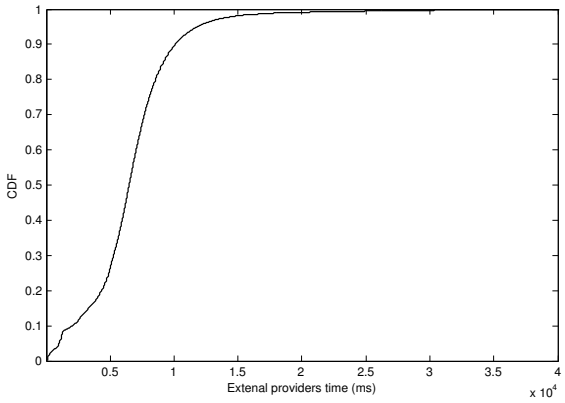
Results from the characterization have been grouped into two categories: workload characterization, including transaction mix, intensity and burstiness; and resource consumption, distinguishing between applications, putting emphasis to databases and external



(a) DB queries



(b) DB time



(c) External Providers Time

FIGURE 3.12: CDF of resource consumption for most popular product during an stationary, high load, 500 minutes period

providers. Results show that the workload is bursty, as expected, that exhibit different properties between day and night traffic in terms of request type mix, that user session length cover a wide range of durations, that response time grows proportionally to server load, that response time of external data providers also increase on peak hours, and that automated crawler traffic is increasing and can represent more than 15% of total traffic, amongst other results that have influenced the development of the thesis.

Chapter 4

Predicting Web session revenue

4.1 Introduction

In this Chapter we propose a new approach to the system overload problem, which is learning from past data, a model for anonymous Web user behavior in a real, complex website that does experience the overload situations. The model is then used to support decisions regarding the allocation of the available resources, based on utility-related metrics. The learning phase captures in a model the selected features according to a utility goal. As a proof of concept, we have selected the features 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.

We describe the architecture of the prototype we have developed, and the simulation experiments we have performed on dataset in Section 3.2. The experiments indicate that using our prototype 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 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 in a small number of servers, possibly shutting down those other servers that would produce little or no revenue.

4.1.1 Methodology

In this section 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.

We have developed a prototype which includes: a script that preprocesses access logs to remove non-user generated actions and rewrites the log in a more convenient format; a program that generates two higher-order Markov chains: one for purchasing users and another for non-purchasing users; an offline learning module that produces a Naïve Bayes classifier or predictor given the log with both static and the dynamic information from the Markov chains; and a real-time module which passes each user click through both Markov models and asks the classifier for a purchase probability. In this way, for every incoming request on the server, the prototype outputs its purchase probability, so that the session manager can prioritize it according to its current load and business rules.

4.2 Progress beyond the State-of-Art

In the context of Web workload analysis, there are few published studies based on real Ecommerce 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 [62]; and file popularity following Zipf's Law [62, 63]. Also, works such as [64] 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.

Related works on Web user behavior prediction and profiling [43–46] have focused on Web caching or prefetching of Web documents, to reduce latency of served pages and improve the cache hit rate. 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.

Path analysis [47, 48] and Customer Behavior Model Graphs (CBMG) such as [50] are similar to the dynamic part of our approach — we use the closely related Markov chain model. Menascé *et al.* [50] 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 chapter 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 [23, 37, 38]; the work presented here is an extension to these aproaches. Related works on resource management, i.e. by Littman et al. [59] uses Naïve-Bayes for cost classification and a Markov chain feedback approach for failure remediation. Other works such as [60] also take into account costs and resource allocation; in contrast with previous approaches, in this chapter we are focusing on the actual revenue that is lost by denying access to purchasing users, and not resource allocation costs.

In the topic of session revenue, in [15] authors present MyQoS, a framework for exploiting session prioritization by comparing known profitable user behavior against individual queries issued by sessions and retrieved results. While[51] presents a similar and later approach making prediction from Customer Behavior Model Graphs. Our session evaluation technique is previous to the presented work [28], it is based on Machine Learning techniques to predict if a user is going to buy or not a particular session. In particular, we predict session profitability from the first click, this way being able to discriminate sessions, and improve precision for each user click.

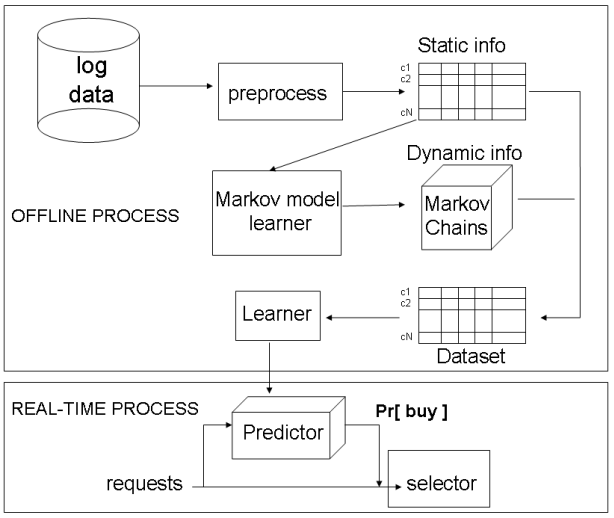


FIGURE 4.1: Prototype architecture

4.3 Prototype implementation

In this section we describe the architecture of the prototype we have implemented to perform the experiments. The prototype currently has two subsystems: an offline component (the *learner*) that takes the historical log file and produces a predictive model or *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 4.1.

The input to the offline component is the log file 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 thesis) to prioritize and even block some of the sessions according to the current load and policies. We remark that the firewall is *not* a part of the prototype: it is often a complex, and very sensitive, part of the infrastructure so we do not aim at replacing it. The prototype, 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.

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 non-buying 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. Related works on resource management, i.e. by Littman et al. [59] uses Naïve-Bayes for cost classification and a Markov chain feedback approach for failure remediation. Other works such as [60] also take into account costs and resource allocation; in contrast with previous approaches, in this thesis we are focusing on the actual revenue that is lost by denying access to purchasing users, and not resource allocation costs. 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. We have extended the experiments described in this chapter, to also detect automatically 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.

4.3.1 Generating static information

The goal of the preprocessor is two-fold: First, it should clean the log file 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 log file 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 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 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.

4.3.2 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 the prototype we propose to use high-order Markov chains to address this issue.

A Markov chain describes (is) 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 $\Pr[s_{i+1} \mid s_1 \dots s_i]$ 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 ono our attemps. 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 patterns which are rare on 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.

4.3.3 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, 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.

We recall very briefly how the Naïve Bayes predictor works. In the training phase, the dataset is used to estimate all the conditional probabilities of the form `Pr[var = x | class = c]`, for all predictive variables `var`, all their possible values `x`, and all class values `c`. These probabilities are stored and constitute the predictor. At prediction time, the probability `Pr[class = c | variables]` can be estimated, using Bayes rule, by multiplying out a few of the probabilities above. This process takes time linear in the number of variables, *i.e.*,, very low.

4.4 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.

4.4.1 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 chapter 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.

	j48 classifier	NB classifier	Logistic
%accuracy	76.5	78.1	72.7
%admitted	25.7	22.9	29.8
%recall	66.9	57.5	68.9
%precision	17.2	16.6	15.3

FIGURE 4.2: Models built by different classifiers with static information only

4.4.2 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/non-buying 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 transactions 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 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

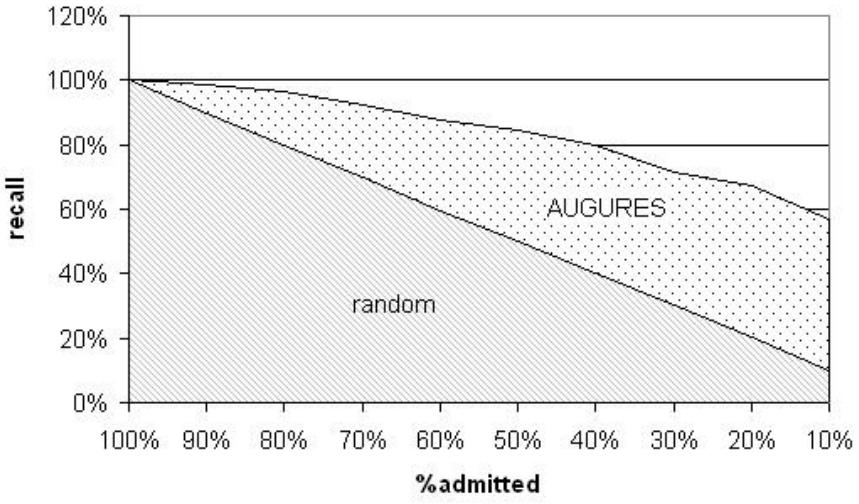


FIGURE 4.3: %admitted vs. recall

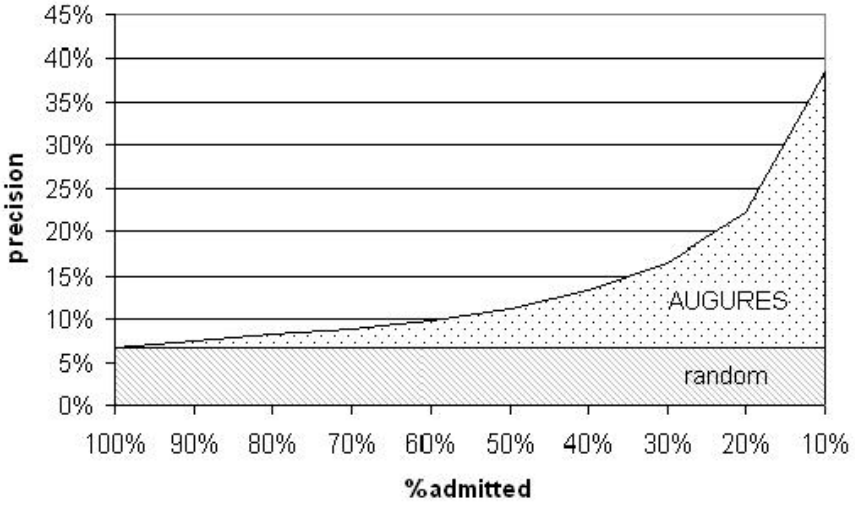


FIGURE 4.4: %admitted vs. precision

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.

4.4.3 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 4.3 and 4.4 present the evolution of recall and precision as we vary the percentage 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 lose 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, the prototype 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 prototype 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, the prototype 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 4.1 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 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.

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

TABLE 4.1: Recall and precision for clicks 1 to 3.

4.4.4 Performance in real-time prediction

Our next experiments test the 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 4.5 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.

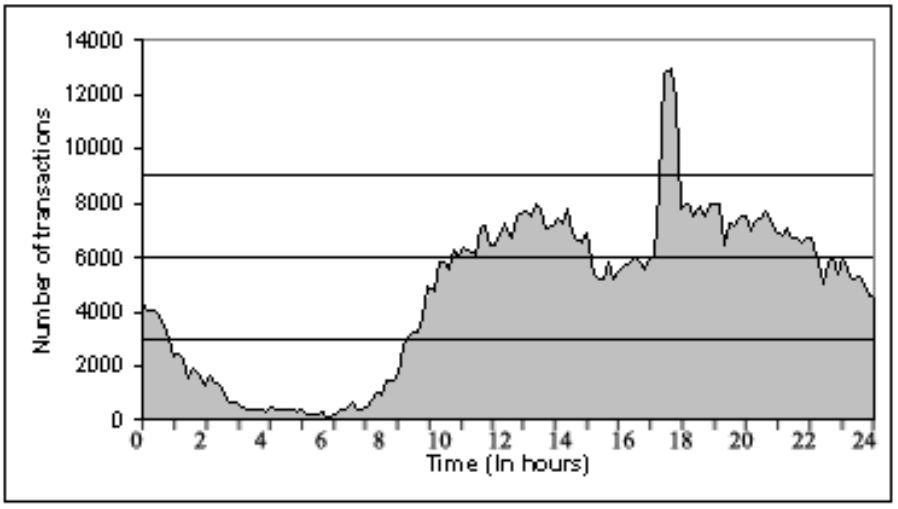


FIGURE 4.5: Transactions/hour rate in the workload

More precisely, we compared the number of transactions that would end up in purchase if admitted with the 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 the prototype with the workload corresponding to the reference day, sequentially. Every T minutes, the prototype computes the rate transactions/hour from the current load and, with this figure, it recomputes ton to these aproaches. Related works on resource management, i.e. by Littman et al. [59] uses Naïve-Bayes for cost classification and a

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 4.2: Results of simulation on real workload

Markov chain feedback approach for failure remediation. Other works such as [60] also take into account costs and resource allocation; in contrast with previous approaches, in this thesis we are focusing on the actual revenue that is lost by denying access to purchasing users, and not resource allocation costs. The threshold of the classifier 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 4.2. 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 the prototype 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.

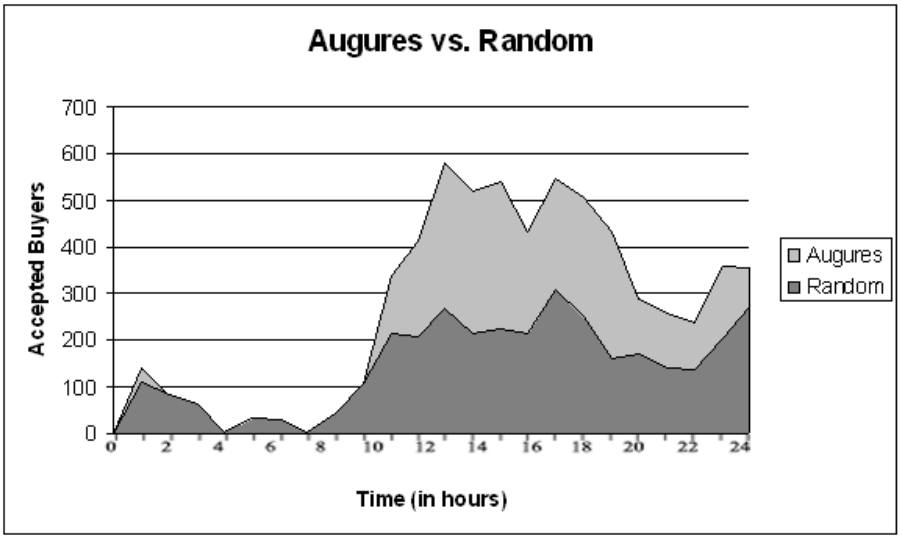


FIGURE 4.6: Admitted buys by the prototype and random selection

Another view of the process is presented in Figure 4.6, which presents the absolute numbers of buying transactions accepted by the prototype 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, the prototype did not act, so the curves for both strategies coincide. On the other hand, when the workload exceeds MAX, the prototype chose better and admitted a substantially higher number of buyers. In other words, the area between both curves is the %improvement column of Table 4.2.

4.5 Automatic detection of content stealing Bots

As an example, we take the online travel sales industry. Online travel agents contract services from Global Distribution Systems (GDS) (see Section 3.1) under specific SLAs and *look-to-book* ratios (number of searches per reservation). When a user makes a flight search, the request is sent via Web services to the GDS, which in most cases forwards petitions to airline companies to produce the final flight availability result.

Recently, *flight comparison* sites are appearing that operate by scraping in real-time several travel sites and combining results in their own page. Although they might be useful for users, they are becoming a problem for real travel agencies and the rest of the supply chain, as each search is resource-intensive and costly. *Flight comparison* sites also increase the *look-to-book* ratio, therefore the derived costs of travel agents,

while operating basically at zero cost. Detection and banning of such sites is generally performed manually by administrators by blocking their IP address; nevertheless, as in the case of spammers, Web scrapers are continuously introducing new techniques to bypass detection, such as IP proxies or IP pools [96].

As the prototype presented in this chapter prioritizes users according to their expected revenue, and most Web scrapers never purchase, we have detected that they were systematically being assigned a very low priority, and thus the firewall could discontinue these sessions in case system resources where scarce. As a proof of concept, we have expanded the mechanism from only detecting purchasing users, to also detect *Web bots*, in a travel agency dataset presented in 3.2 that we use again here.

4.5.1 Experiments Bots

We wanted to test whether it was possible to identify with reasonable certainty bots accessing a Web site for automated banning. For this we have classified each session in the training dataset, as either a *buying* (human) user, a *non-buyer* user, or a *bot*. To classify a session as content stealing bot we have used the following criteria: the total number of searches, time average between searches, and the number of origins/destinations in all searches, for each Web session or IP addresses. The approach has been manually sampled to validate bot classifications to minimize false positives; once training models are generated further classification can be performed automatically. In this way, about 15% of the traffic in the training set ended up being marked as due to content stealing bots.

In particular, 74% of bot accesses were predicted as such, and among all accesses predicted as bots, 81% truly corresponded to bots. We regard these figures as quite good. In particular, they indicate that the system could potentially be used to filter out at least 10% of total traffic by banning bots.

4.5.2 Summary Bots

Content stealing on the Web is proliferating rapidly as it becomes more profitable, affecting not only content sites, but also Ecommerce sites that rely on paid transactions for product availability. We have extend the prototype to show promising results on the

applicability of machine learning techniques for automatic detection and banning of bots. Results obtained can free administrators from manually performing these operations while leveraging revenue loss from the spurious transactions.

4.6 The effect of dynamic server provisioning

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 prototype implementation makes a big difference between both situations: using the predictions from the prototype, the most promising customers can be allocated to server 1, the next most 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 non-buying 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

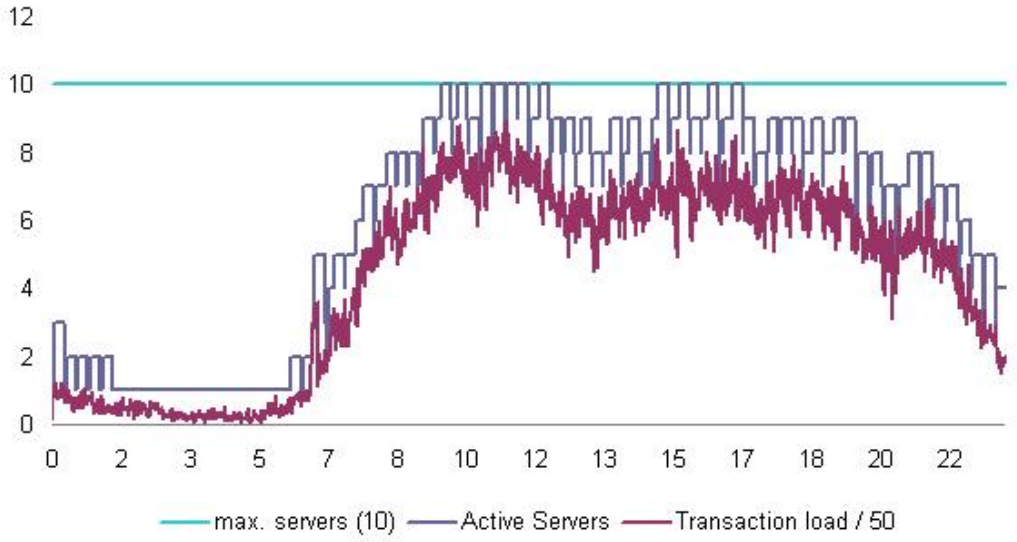


FIGURE 4.7: Evolution of load and number of dynamically provisioned servers

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 4.7 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 over provisions 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.

So far, this is independent of the usage of the prototype. 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 4.3 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

#servers	Buyers Static & Dynamic	Buyers Prototype	Benefit Static	Benefit Dynamic	Benefit Prototype
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 4.3: Buyers served and benefits of different strategies

as the load requires, and sessions are assigned to servers as above; and the combination of the dynamic strategy which the prototype, 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 prototype 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 over provisioning 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 prototype 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, the prototype's

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), the prototype's 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 losing not only buyer, but also non-buyer 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.

4.7 Other potential applications

This section describes other potential applications to the techniques described in the chapter. 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 chapter, 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 do not sell products, yet 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 that 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 fewer resources and penalize resource-consuming ones. 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* [28] 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 [44–46]. Predicting user intentions can be used in the field of *dynamic content adaptation* [43, 49], where the page content, layout, links, and banners change according to the predictions about the user.

4.8 Summary

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 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.

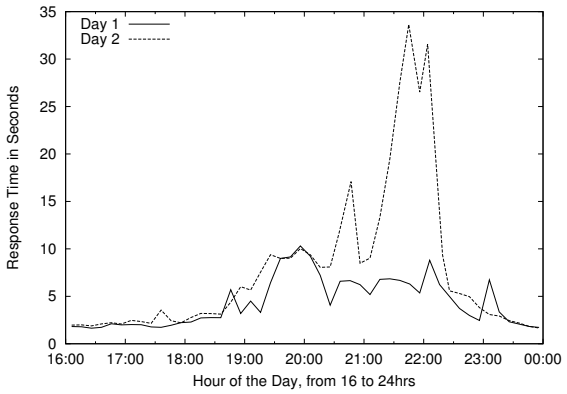
Chapter 5

Response time effect on sales

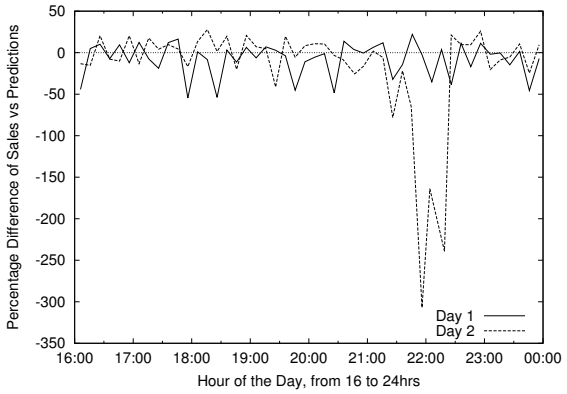
5.1 Introduction

Capacity planning techniques and service management for online Web sites are living a period of changes: the cloud computing paradigm and the appearance of advanced service management products that dynamically adapt provisioned resources pose new challenges at deployment time. System administrators need to make important decisions about the different parameters that seriously affect the business results of any online Web site such as: what is the best performance goal for each online application, usually presented in the form of a response time objective? What is the highest number of servers that the company may afford on peak hours and other workload surges? What would be the effect, in business terms, of limiting the resources for an application and degrading its performance slightly to reduce the bill of the hosting? In this chapter we propose a methodology that provides answers to these questions without the need to manipulate the systems in production, as it only requires offline information usually collected by most online Web sites.

Determining the impact of high response time on sales and business volume is something that can be studied injecting delay on pages and using A/B testing methodology to measure, but this approach is not always feasible. Very large companies can intentionally degrade the performance delivered by a fraction of their servers with minimal impact for their business to study the effects of performance degradation on their business balance. The same process may have a strong impact for small companies and thus, they hardly



(a) Observed response time



(b) Relative error between actual sales and predicted sales

FIGURE 5.1: Comparison of Response Time and Predictions in Normal and Overload Operation for Two Days

decide to use such approach. Therefore, new methods must be developed to carry on such studies with minimal interference on the systems.

In this chapter we introduce a novel methodology for studying what is the total volume of sales lost for an online retailer during performance degradation periods. We use Machine Learning techniques to predict the expected sales volume over time and look for deviations over the expected values during overload periods that introduce performance degradation. Using such technique, we can estimate the total impact of high *response time* in the business activities of an online service in a non-intrusive way. The proposed methodology starts with a study and characterization of the sales dataset presented in Section 3.2, that leveraging Machine Learning techniques constructs a model of sales. Such model allows for an accurate prediction of expected sales in short time frames. The model is then used to contrast actual sales with expected sales over time, and determine the impact in sales of overload periods that caused degraded QoS —measured in the response time— by the online applications.

5.1.0.0.1 High Response Time Example For the sake of clarity, we include here a simple example that shows the use of the methodology presented in this chapter to a real Web application. Figure 5.1(a) shows response time for the same application over a 8h period for two different days, where *Day2* corresponds to a day in which some overload was observed, and as a consequence, performance degradation resulted in a response time surge; on the other hand, *Day1* corresponds to a regular day. A question one may want to answer after observing such performance degradation is: what is the volume of sales that was lost due to response time surge? The result of applying the proposed technique to the previous example can be seen in Figure 5.1(b), where relative error between real sales and the predicted sales volume is shown. As it can be seen, the expected sales would be higher than what was observed, and it can be even quantified by what margin. In the following sections we will further elaborate on how to systematically build the sales model that helps estimating the loss of sales during the overload period. Where the model needs to capture conversion rate variability of every time and date in small time frames, as QoS on servers changes by the minute and sales are prone to seasonality effects.

5.1.1 Methodology

This chapter first presents the study of conversion rates for a real OTA, that brings results not previously reported in the literature about peak load periods; and second, the use of a sales model built using machine learning technologies with the goal of quantifying sales loses due to overload periods and response time surges. To our knowledge, such application of a sales model has not been previously reported in the literature. We have classified the contribution as a four steps methodology that can be systematically followed to understand the sales characteristics of any online business, and to build and use the sales model mentioned above. In particular, the proposed steps are:

- Step 1: Workload and performance characterization of the supplied datasets to understand user behavior and the causes high response time. (Section 3.2).
- Step 2: Study the conversion rate variation during the day for the OTA based on sales datasets (Section 5.5).

- Step 3: The construction of a model for sales prediction through Machine Learning techniques and its validation (Section 5.7).
- Step 4: Characterize response time thresholds of user satisfaction (satisfied, tolerating, frustrated) for the OTAs applications (Section 5.8).

The output of Step 4 is suitable to be used for building autonomic resource managers that dynamically adjust the resources allocated to each different online application in cloud-like environments while observing conversion rates, performance, and energy constraints. Such use of the proposed methodology is our current subject of research.

5.2 Progress beyond the State-of-Art

Response time effect on user behavior has been studied as long as 1968, where Miller [30] describes the "Threshold Levels of Human Interaction and Attention with Computers". In 1989, Nielsen [81] revalidated Miller's guidelines and stated that the thresholds are not likely to change with future technology. These thresholds being: 0.1 to 0.2 seconds, instantaneous; 1-5 seconds the user notices the delay but system is working; and 10 seconds as the threshold for user attention. Other authors [79, 80] adhere to what they call the 8 seconds rule, where no page should take longer than 8 seconds to reply. While the APDEX standard [82] sets the threshold for frustrated users at 4 seconds. The are several industry reports stating that users expect faster response times than previous years, specially the younger generation [1, 4].

To prevent loss in sales due to overloads several techniques have been presented such as *session-based admission* control systems [22, 23] used to keep a high throughput in terms of properly finished sessions and QoS for limited number of sessions. However, by denying access to excess users, the Web site loses potential revenue from customers. Later works include service differentiation to prioritize classes of customers, in [99, 100] authors propose the use of utility functions to optimize SLAs for gold, silver and bronze clients. In Chapter 4 we propose the use of Machine Learning to identify most valuable customers and prioritize their sessions.

Menasce *et al.* [11], perform an assessment on how QoS of Ecommerce sites plays a crucial role in attracting and retaining customers, where they propose a workload manager

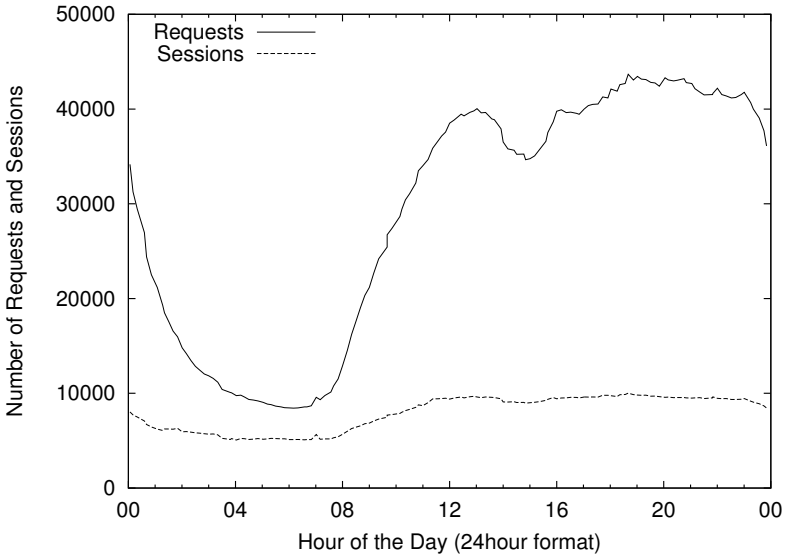


FIGURE 5.2: Number of requests and sessions for 24hr period

using predictive queuing models of computer systems based on a *hill-climbing* method. Authors perform admission control based on the response time of the system. However, their work is based on a popular but synthetic workload, the TPC-W, and does not have sales into account in contrast to this thesis. In [7], Mazzucco proposes the use of utility functions to optimize auto-provisioning of web servers. None of this works however, explore the effects of higher conversion rates a peak loads, and by ignoring this fact, QoS of service is no optimized and potential sales are lost. Nowadays, Cloud computing enables enable 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, enabling autonomic resource managers to auto-configure server numbers [7, 101]. In Chapter 6, we show how the methodology presented in this chapter can be extended to produce a QoS-to-sales model to be applied to Cloud environments.

5.3 Workload characteristics

For the dataset, the mean pageview for the whole dataset is 5.1 pages per session, with 6:45 minutes spent on the site, an average of 1.5s response time for dynamic page generation, and 10MB of RAM memory consumption. Recall that the highest traffic is on Mondays and decreases to the weekend. The opposite effect is observed on average *pageviews* as well as the time spent on the site; they both increase during the week, peaking at the weekend, from: 5.82 and 6:40 on Mondays to 6.27 and 7:31 on Sundays,

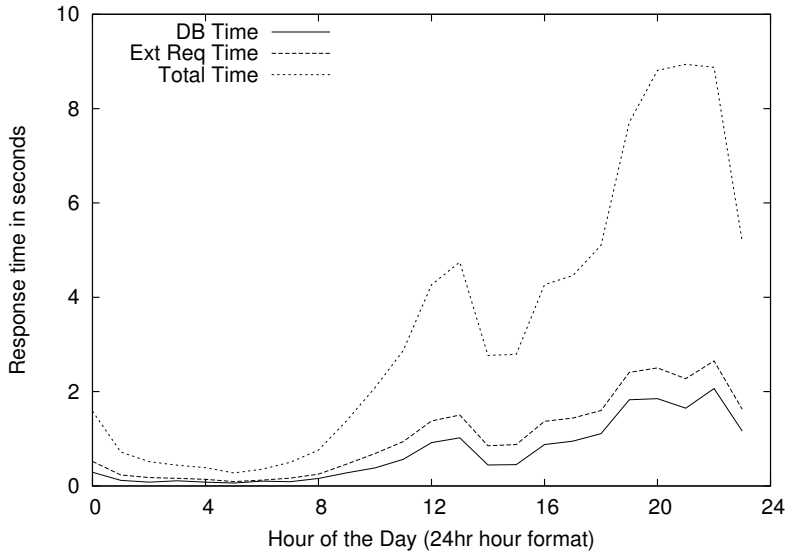


FIGURE 5.3: Variation of DB and External Requests time during a busy day

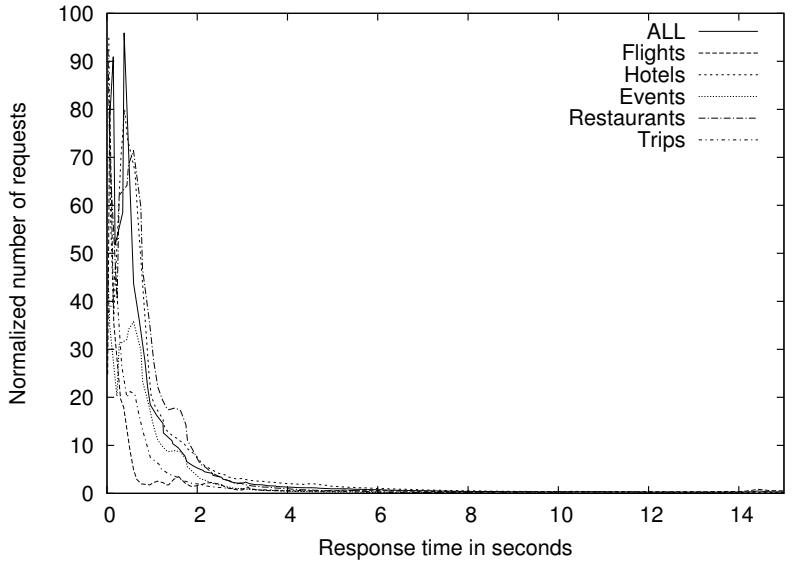


FIGURE 5.4: Normalized percentage number of requests by response time for different products

pageviews and time spent respectively. An interesting feature of the dataset is that mobile devices represent over 6% of the visits at the end of the dataset, while the previous year represented less than 2% of the total visits, showing a growing trend.

Figure 5.2 presents the number of *requests* and *sessions* for an averaged 24hour day from the full dataset. As it can be observed, traffic starts growing from 8 to 14 hours, lunch time in Spain, where most of the traffic comes from, then it decreases until 16, return to office hours, and continues to grow peaking around 22 hours. After midnight, traffic volume decreases until it finally reaches the beginning of the cycle again.

5.4 Response time analysis

In the following section we perform an analysis on response time: how it varies during the day, how it is affected by server load, how it affects the different applications, and finally it effects on *user behavior*. While studying the dataset, we have noticed that there was a great variation of response time during the day for the different applications of the OTA, as shown in Figure 5.3. We have identified for the OTA that *response time increase* was directly related to: the number of concurrent user sessions at a given time at the site, database response time, and external providers response time. Database and external providers response time also increased at peak hours of the OTA as can be seen in Figure 5.3, causing part of the slowdown. Some resource contention is also present at peak load times, from Figure 5.3 there is contention between 18 and 23 hours.

As a note, most of the requests in our dataset have a response time below two seconds as shown in Figure 5.4, so even in peak times periods, the system is generally in a normal state. Total response time for a request is the time it takes Web servers to start sending the reply over the network to the user's browser. It is important to notice that in the OTA's application, *output buffering* is turned *on* for all requests, so no data is sent over the network until the full request is processed and *gzipped*, if supported by the user's browser. There is an additional time for the browser to render the final webpage, but it is not present in our dataset and is not part of this study as it deals with the actual HTML and media content.

5.4.1 Response time decomposition

From the available dataset, response time can be decomposed into: CPU time in system mode, CPU in user mode (including I/O times), database wait time, and external request wait time. Figure 5.3 presents the total response time for the complete dataset grouped by hour of the day. If we contrast it with Figure 3.5, by each daily period it can be seen clearly that response time increases with the total number of requests. Figure 5.3 also divides total time by the different resources, where the database response time also increases at peak hours. External request response time is not affected in the same proportion. CPU in system mode is not plotted on the graph as it was too low in comparison to the rest of the features; however it also presented noticeable higher

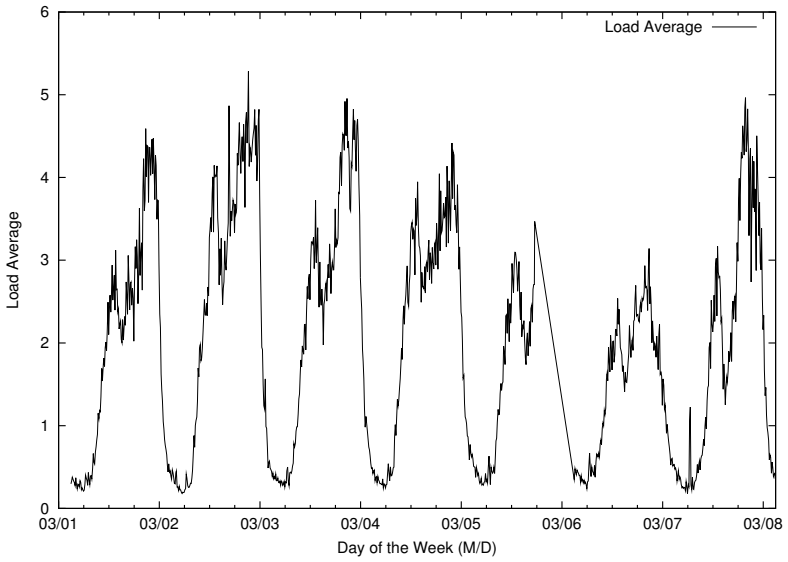


FIGURE 5.5: Load Average values for the week

response times at peak load. At peak time, from 18 to 22hrs, as Web server process more requests, they also present some resource contention due to high *load average* detailed in the next section.

5.4.2 Response time and server load

The next section analyzes how response time is affected as the *load* on the Web servers increases. To measure server load, we take the *load average* system value present in most UNIX systems [102], recall that the value of the *load average* is related to the number of scheduled jobs pending to be processed by the CPU. *Load average* is a very extended, simple, but accurate value for measuring current load on a server; in this study we use load averaged to 1 minute —opposed to 5 or 15 minutes— to have higher detail. To understand how busy is a server by the load average, it is important to notice that each Web server has 2 CPUs with 2 cores each (described in 3.1.6), giving a total of 4 cores per server; as a rule of thumb, after 4 units of *load average* servers are considered overloaded (1 for each core).

Figure 5.5 presents the *load average* of the servers during the one week dataset. If we compare Figures 3.5 and 5.5 we can correlate how *load average* is affected directly by the number of concurrent requests on a given time, and that it follows the daily request pattern.

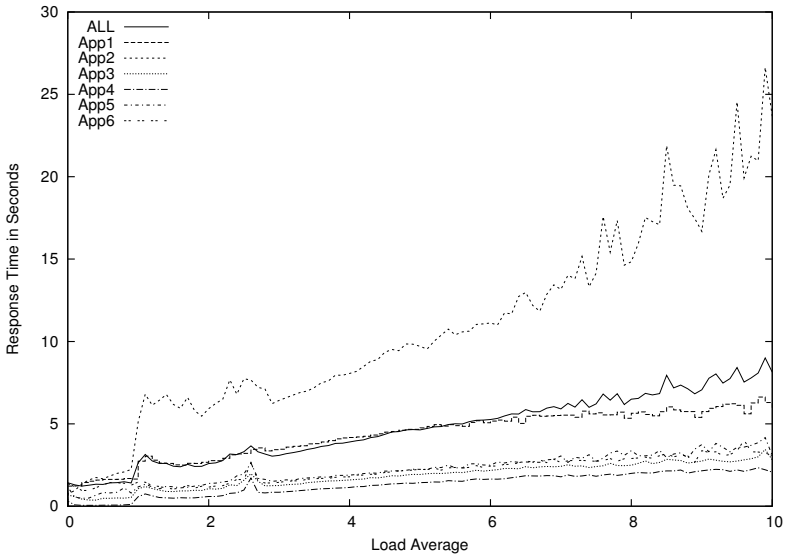


FIGURE 5.6: Response Time by Load for different Apps

In Figure 5.6 we plot response time (Y axis) and *load average* (X axis) for the most popular applications, Apps 1 through 6 and the average *ALL* for all applications. Load average starts from 0, being the least busy value, to 10, the maximum value recorded on our dataset. From Figure 5.6 it can be appreciated that response time increases almost linearly as server load increases. From load 0 to 10, it increases almost to 10x in *ALL* for all applications, and up to 25x for App 2.

Response time increases with server load for three main reasons: server resource starvation (less dedicated system resources for a request), external B2B requests increased response time (low QoS), and contention between resources (jobs waiting for blocked resources).

For server resource starvation, Figure 5.7 shows how the percentage of CPU assigned by the OS to a specific Apache thread (request) reaches a maximum at load 2 (saturation point), and then starts decreasing leading to higher response time values. The same effect happens with the percentage of assigned memory, Figure 5.8, plots how memory percentage to Apache threads decreases very steeply from load 2.

As for external resource QoS, Figures 5.9 and 5.10 shows the response time for database queries and external B2B requests respectively. In Figure 5.9 we can see how the database response time also increases 3x in average for all applications, and up to 8x for App 2, which has a higher database usage. Figure 5.10 shows the average response time to external requests, we can see that App 2 is affected highly by the QoS of the

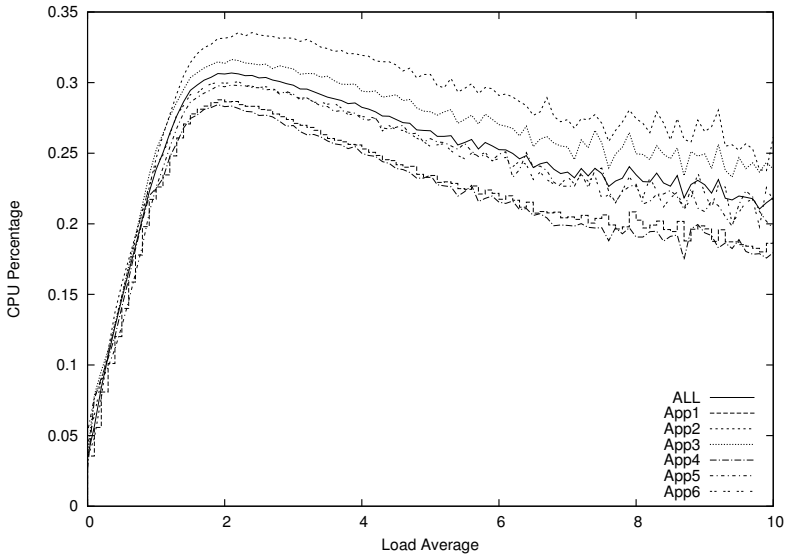


FIGURE 5.7: Percentage of CPU Assignation by Load for Different Apps

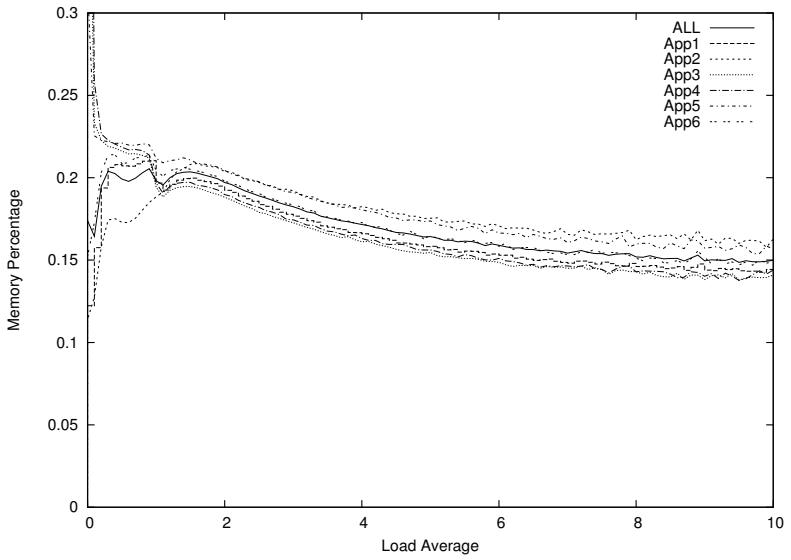


FIGURE 5.8: Percentage of Memory Assignation by Load for Different Apps

external provider(s), while for App 1 it stays constant. The effect on App 2 is caused by the external providers getting overloaded at similar day times, than the analyzed site; while the QoS for App 1 is stable at the sites peak hours.

It is important to notice that the less affected application by server load, is App 3, it is clear from Figures 5.9 and 5.10 that it does not rely on database or external requests, having the lowest response time increase, 2.5x. The other extreme, App 2, is heavily affected by both the database and external request times. An important feature from the last figures, if we zoom into the graph, is that the best response time is not at load 0, but is between load 0 and 1, as at load 0 (mainly at night time) *cache* hit rate is lower

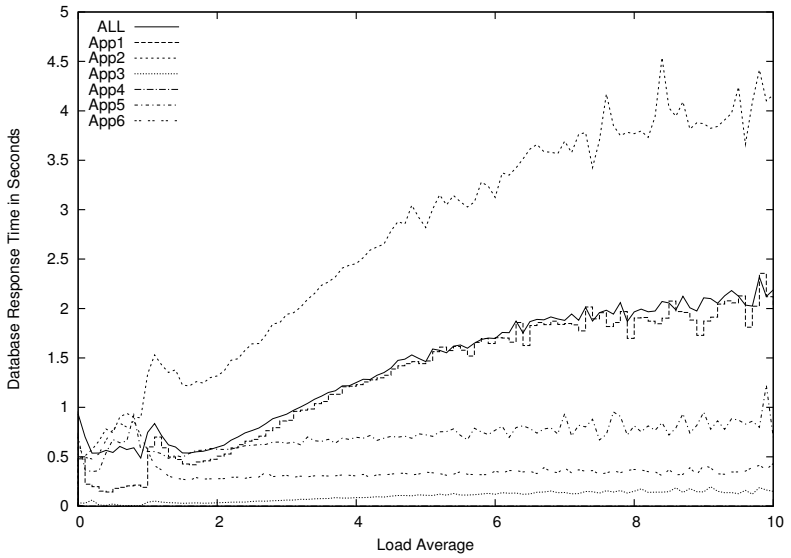


FIGURE 5.9: Database Response Time by Load for Different Applications

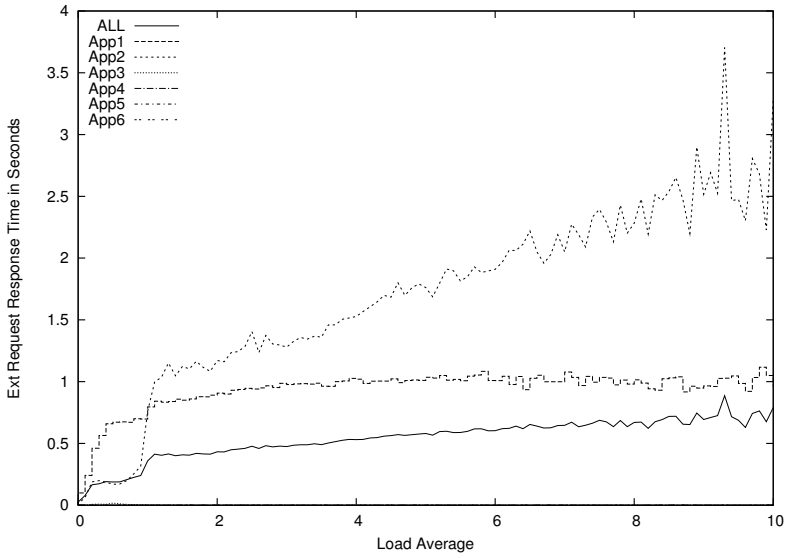
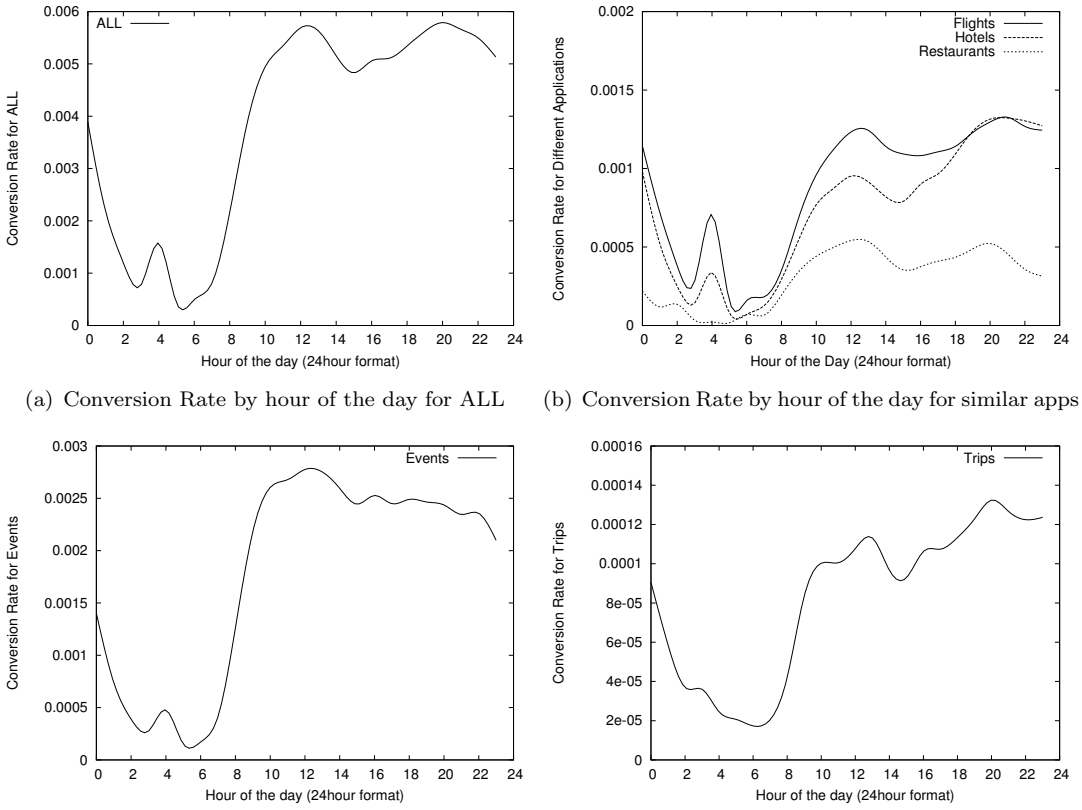


FIGURE 5.10: External Request Time by Load for Different Applications

which leads to slightly higher times, although not comparable to high *load average*.

5.5 Conversion rate study

The main concepts of Conversion Rates, CRs, where briefly introduced in Section 3.1.3. The following section provides and analysis of the CR of each product of the OTA during the 6 month period of the *performance datasets*. CRs are calculated from the number of sessions in the *performance datasets* and the number of sales for each product in the *sales datasets* for the corresponding period of time. The objective of such analysis is to



(a) Conversion Rate by hour of the day for ALL

(b) Conversion Rate by hour of the day for similar apps

(c) Conversion Rate by hour of the day for Events

(d) Conversion Rate by hour of the day for Trips

FIGURE 5.11: Conversion Rate by hour for different applications

understand how selling hotspots are distributed over time for the studied business. Such study is very relevant to measure and quantify the effects of QoS on the sales volume observed for the OTA. Notice that the CR does not necessarily change with oscillations in volume of traffic that can be observed for any Web site (*e.g.*, Figure 5.2), as CR has to do with the fraction of visitors purchasing products, and not with the overall volume of visitors.

5.5.1 Conversion rates by product

Figure 5.11 presents the CR of the different products of the site; averaged and grouped in a 24 hour period. Figure 5.11(a) shows the CR of *ALL* the products combined; Figure 5.11(b) groups products with similar CRs; while Figures 5.11(c) and 5.11(d), *events* and *trips* (vacation packages) respectively.

In average for all products of the site, the conversion rate is higher at the site's peak traffic time in number of requests. The CR of the averaged application follows very

closely the daily patterns of the number of sessions in the system (compare to Figure 5.2) during a 24 hour period: from 1am to 8am there is a low CR, sleep time and also higher percentage of bots and crawler activity; from 8am until 14hrs, lunch time in Spain where most of the traffic comes from, the CR increases progressively during the morning as the number of sessions increases; from 14hrs until 16hrs, the CR lowers during lunch time, then continues to grow in the afternoon along with sessions, until 18hrs where the CR decreases due to end of office hours; it increases again from 20hrs peaking around 22hrs. A remarkable conclusion must be pointed from the observation of these figures: the CR for all the products available from the OTA follow a pattern that is strongly correlated to the traffic pattern observed in Figure 5.2. The interpretation of this fact is that the hours at which the traffic volume is the highest, the fraction of customers that are actually purchasing products is also higher, resulting in still higher sales periods. Recall that such a result indicates that the relation between volume and sales is not constant over time, and leads to a very important increase of sales over peak periods were not only traffic volume grows but also the average CR; most users buy at similar time-frames.

Notice that it is a usual case that many infrastructures are not dimensioned for sustaining QoS at peak hours, as they are considered surges in the traffic and static provisioning of resources to manage punctual very high traffic volumes is unaffordable. Of course, such decision results in worse response time and QoS in general during peak periods. Looking at the charts for Atrapalo.com, it can be observed that these are not only surges in traffic, but also the best-selling periods of the day. Although industry and consumer studies (see Section 5.1) reports that a high response time has a direct effect on sales, as conversion rates are higher at peak times, the total loss in sales might not be apparent in most cases. Figure 5.11(b), shows the CR for products with similar magnitudes: flights, hotels, and restaurants. The *flights* application has similar CR during day, while *hotels* has a higher peak than the other applications between 21 and 23hrs. Restaurants in the other hand have subtle differences, has a morning peak at 13hrs, just before lunch time at 14hrs; the same happens at 18hrs before dinner time.

Figure 5.11(c), show the CR for *events* which has a distinct pattern it grows fast in the morning and peaks at 12hrs then decreases throughout the day. For *events* the most important time frame to provide a high QoS is between 10 and 13hrs, while for *hotels* is between 19 and 22hrs. Another interesting feature of the CR of *events*, is the magnitude

of the CR, which is twice as high compared to applications in Sub-figure 5.11(b). This might be due to the fact that certain events are sold exclusively by the site. This fact makes this product more inflexible to loss in sales due to poor performance.

5.6 Conversion Rates as a function of response time

It has been shown above that, in general terms and for most products, high conversion rate times coincide with the rest of the product peak hours and worst response times. This can be seen by comparing Figures 5.2, 5.3, 5.11(b), and 5.12. Notice that for each application, its CR will inherently exhibit a different behavior in result of changing QoS, which is caused by the nature of the application.

Figure 5.12 explores the *conversion rate* as a function of average response time by combining data fom the long-term sales dataset and data from the short-term performance logs (see Section 3.1) grouped in 10 minutes intervals from the performance dataset. By analyzing the figure there is no clear indication that a high response time yields fewer sales. On the contrary, most applications are able to maintain CR and sales even in the periods of higher response times. For instance, *flights* usually involve complex search it would indicate that a high response time maintains or improves sales. To study this effect further, the next section presents our methodology for forecasting expected sales in short time frames for each application; in order to measure how a low QoS during peak times affects more sales than previously reported.

5.6.1 Seasonality

Vacation products such as flight and hotel reservation vary greatly during the year according to season. During spring there is a high season for reservations before summer; the opposite effect is appreciated during fall, where the number of sessions drops significantly. Holidays and school vacation periods also affect touristic products, *e.g.*, Christmas and New Year's Eve also creates peak time during the first days of December. A similar but more condensed effect is seen in ticket booking and as tickets for certain events tend to be limited, a rush of users might flock the site to get hold of the tickets, also creating peak loads. Moreover, as users are a click away from any site on the Web,

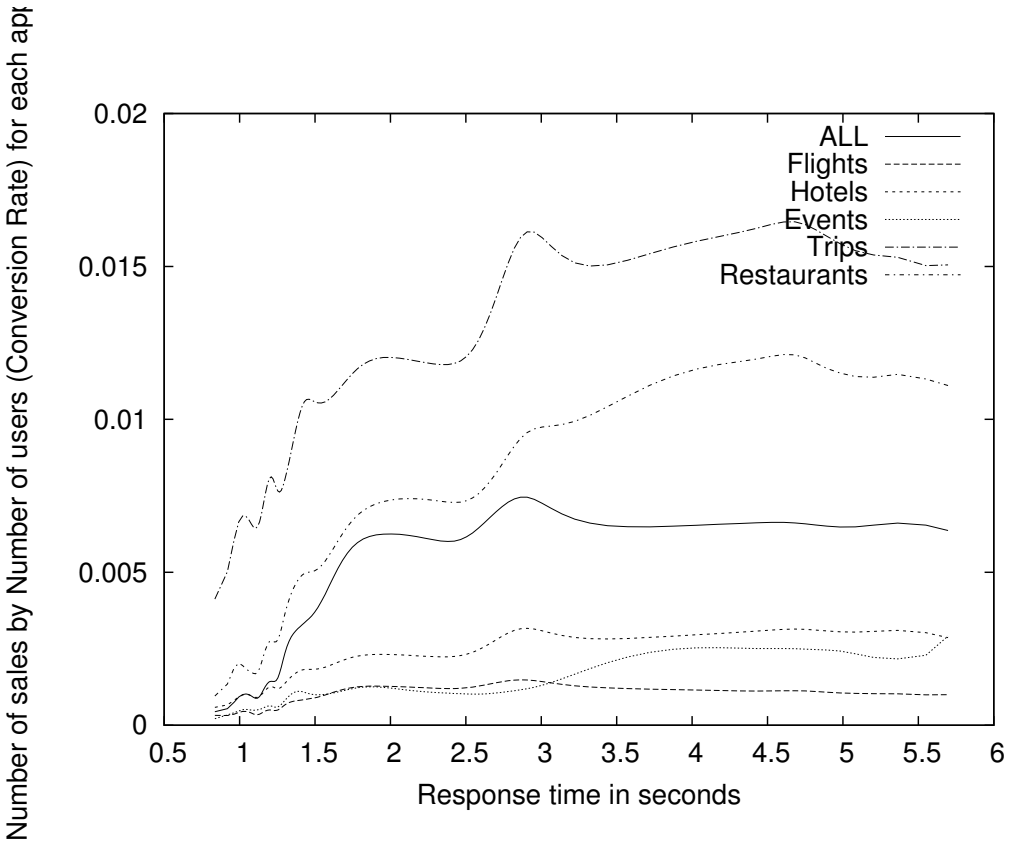


FIGURE 5.12: Conversion Rate as a function of response time

certain events such as mail promotions, high-impact banners, social linking can create peak loads in the workload within minutes.

5.7 Predicting sales with Machine Learning

Most traditional sales forecasting techniques involve some form of linear or multiple *regression analysis*. In our preliminary work we have tried several methods *i.e.*, *linear*, *quadratic*, and *cubic* regressions to predict expected sales for different weeks in short periods of time bins, *e.g.*, 10 minutes, using the *sales* dataset. We found that predictions were too general and not precise enough for the short time frames we wanted to measure response time in. To overcome this situation and improve predictions, we have built a sales forecasting prototype implementing Machine Learning numerical algorithms. The prototype implementation learns from the different CRs during the day and across seasons, with training from the five years of the sales dataset to overcome the effects described in the previous section about peak loads and seasonality effects.

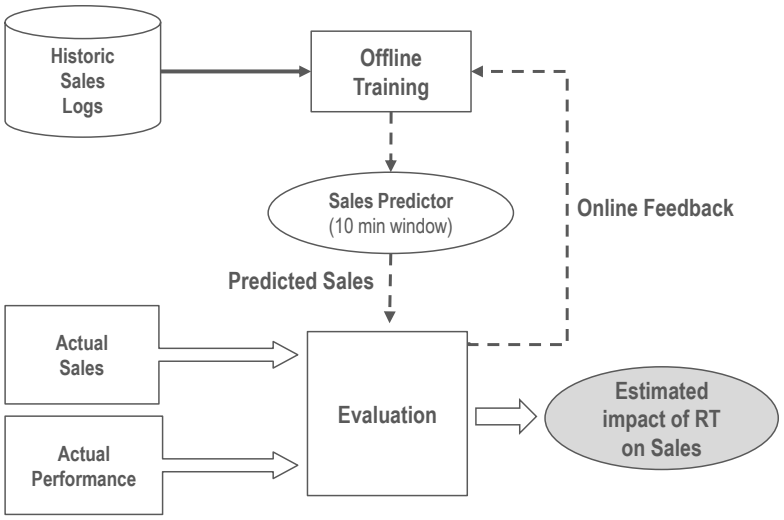


FIGURE 5.13: Sales prediction process

To forecast sales, we have built a prototype based on the WEKA open-source Machine Learning package (see Section 2.4.1), which contains several ready to use classifier algorithms. As we want to predict expected sales for short time bins, representative of the *response time* (QoS) in the system at that moment, while also having precise prediction results, which depend on the volume of sales per time bin in the *sales* dataset. We have tested predictions for 30, 15, 10 and 1 minute bin intervals. One minute bins turned out to be too low to have accurate prediction results, and as Web sessions for buying visits are longer [27], it only partially reflected the response time for the session. Thirty minute proved to be too high, as server status might have changed drastically in this time frame. After some experimentation, we have decided to use ten minutes as the time frame for the rest of the experiments: it is low enough to represent the current QoS on the system and high enough to cover most of the Web session with accurate sales predictions in our sale dataset.

In WEKA, a *predictor* model is trained using a specially formatted *training dataset* that should contain the most relevant available variables to predict sales. After training the *predictor* with the training dataset, a *test dataset* with the same format is used to perform predictions. The predictor reads the test dataset, ignoring the *class* —the unknown variable, in our case the number of sales— if present, and according to the training and the algorithm used, outputs a prediction of sales —the class— for each of the time bins.

Figure 5.13 presents the general process of our prototype implementation. It begins by

preprocessing the historic *sales* dataset provided by the OTA, by aggregating entries into the specified time-bin length —10 minutes in our case— then exporting the aggregated dataset in a format compatible with the sales learner. Following, the sales learner implements the specified machine learning classifier and trains itself to produce the predictor model as an offline process. After the predictor model is trained, for every 10-minute bin from the period corresponding to the *performance* dataset, the prototype performs a sales prediction. From this moment two things happen: first, the data on the period corresponding to the time bin, is fed back to the learner module; second, actual and predicted sales are compared and the difference in number of sales is stored along with the current system response time. The following sections detail the prediction process and prototype implementation.

5.7.1 Prediction methodology

For the training dataset, we have preprocessed the 7 year long sales history dataset (see Section 3.2) into 10 minute bins, each day containing 144 bins; creating a *training* and *test* datasets for each application of the OTA, and one for *ALL* the applications combined. As resulting predictions should be as-close-as-possible to the actual number, but not necessarily predicting the exact number, for this purpose we have implemented in the prototype several *numerical classifiers* available in WEKA, that predict values within an error percentage range.

5.7.1.1 Attribute selection

After some experimentation selecting attributes the training dataset contains the following attributes:

- Number of sales for the 10 minute bin. The *class* being predicted.
- Year of the sale.
- Month of the sale.
- Day of the year.
- Day of the week.

- Hour of the day.
- The 10 minute bin for the day, from 1 to 144.

The goal is that each attribute adds valuable information when building the *predictor* model. For example, the month of the year should add information on whether we are in low or high season. Each day of the week has differentiated sales volume: Mondays have more sales and decreases through the weekend for most applications, except for *events* which is higher on Fridays. The same goes for the time of the day, which has different conversion rates for each application (Figure 5.11(b)). It is important to remark that the *training* dataset only contains data previous to the *test* dataset.

More attributes could be added to improve predictions, especially to cover seasonality effects *e.g.*, of holidays, where the numbers of days to a vacation period could have been added. However the purpose of the prototype is to provide representative predictions and a proof-of-concept of the method. As a note, most numerical algorithms can improve predictions by using *nominal* values instead of numerical ones, *e.g.*, hour of the day as a category not as a number, however resource requirements are higher and default parameters for the algorithms needed to be tuned to improve predictions; we found only negligible improvements using nominal attributes and the combination of numerical and nominal attributes at the same time.

5.7.1.2 Prototype Implementation

For our prototype implementation we have used the 7 year long sale history dataset to create the training dataset, we cut the dataset the day before we want to apply the predictions, in this case at the beginning of the performance dataset so no information of the future known when building the model. Our prototype implementation has as a parameter how long (in days) the test dataset should be and creates several training and test datasets incrementally. As an example, setting as input 7 days to the prototype, the first training dataset would contain data from the beginning of the sales dataset, to the date where the performance dataset starts —the first week of February 2011. The second training dataset would contain same data as the previous training dataset plus the first week of February, the second test dataset would be for the second week of February, and so on.

	LinearR	**M5P**	**REPTree**	**Bag(M5P)**	**Bag(REPTree)**
Correlation coefficient	0.7215	0.9515	0.9465	0.9571	0.9556
Mean absolute error	16.8619	5.892	5.8393	5.6465	5.6517
Root mean squared error	19.9969	8.1515	8.067	7.8087	7.8263
Relative absolute error	64.3319%	24.6382%	24.418%	23.6116%	23.6333%
Root relative squared error	65.8654%	29.9866%	29.6757%	28.7257%	28.7904%

TABLE 5.1: Classifier evaluation

We have tested predictions with different lengths, as the shorter the time frame, the better predictions should be, as behavior from previous days is learned by the model and should improve predictions. However, we have noticed that since the sales dataset covers a great time span, for the tested classifiers previous days do not influence or improve results significantly. Moreover, if on the previous day there was a surge, we do not want the model to predict lower values for the next day, but what would be the expected sales taking into account the conversion rate for this product. In a real-time online implementation, the model will be up to date to the previous time bin. 7 days (1 week) was used as test dataset time span for the presented results in the next sub-section.

5.7.2 Prediction results

The first training dataset used by the prototype to build the *predictor* contained 214,849 instances, while the first test dataset contained 1008 instances, for a total of 24,192 tested instances for the full length of the performance dataset. We present results for the following numerical classifiers found in WEKA: LinearRegression, REPTree, Bagging(M5P), and Bagging(REPTree). More complex classifiers could have been used *e.g.*, *neural networks*, but the processing time required for training seems too high to consider them in a real-time applications and were discarded for the time being.

Table 5.1 presents accuracy for the different selected classifiers: *LinearRegresion* is the least performing classifier with a Relative Absolute Error of 64.33%, while *M5P* and *REPTree* have 24.63% and 24.41% Relative Absolute Errors respectively, *REPTree* being faster to train. The *Bagging* meta-classifier was also tested implementing both *M5P* and *REPTree*, improving precision for both algorithms to 23.6% for both *Bagging(M5P)* and *Bagging(REPTree)*. Meta-classifiers split the training dataset in instances, testing different parameters of the selected classifier and selecting the most precise ones, they

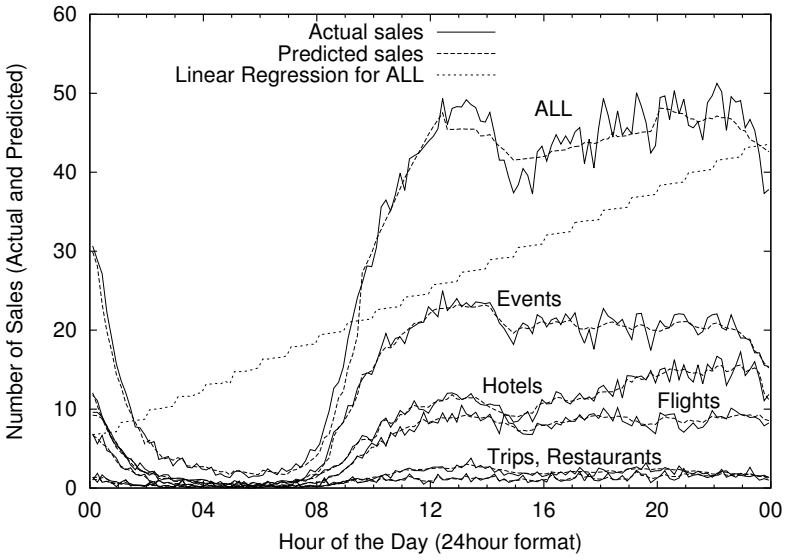


FIGURE 5.14: Classifier Precision by App for an Averaged 24hour Day

perform several iterations of the regular classifiers with different attributes and selecting the best for each case, but take longer to train. As a note, all experiments were performed using default values in WEKA. Linear regression has essentially no parameters. RepTree and M5P, like all decision tree methods, have a parameter controlling the size of the tree; we verified that, in our case, optimizing over that parameter gave only negligible advantage over the default value. Figure 5.14 presents the averaged class results for a 24hr day of actual sales vs. predictions.

Although *LinearRegression* was the least performing algorithm, the model is simple enough to illustrate how classifiers work:

$$10minute\_sales = 1.48 \times year + 0.17 \times month + -0.03 \times day +$$
$$0.43 \times day\_of\_week + 1.08 \times hour + 0.08 \times 10minute + k$$

Where k is a constant not disclosed for confidentiality limitations. Parameters for each variable are dependent on the training dataset values, and in this case specific for the OTA and presented as an example.

Apart from linear regression, the rest of the classifiers have less than $\pm$ 2% error difference compared to actual sales when average response time for the 10 minute time bin falls

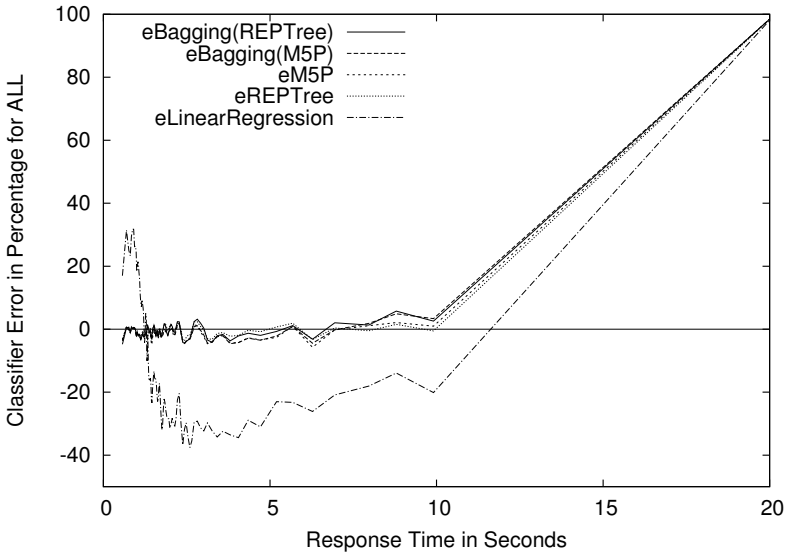


FIGURE 5.15: Classifier errors by response time

within 2 seconds. Between 2 and 4 seconds, the classifiers are less precise and underperform up to -4%, this might indicate that the site sells more than expected at this QoS. After 4 seconds, classifier error rate starts increasing positively, over predicting results, with milestones at 6, 8, and 10 seconds. From 10 seconds, classifier error starts to grow steeply from an average of +3% errors to 40% at 14 seconds. When response time is above 10 seconds, classifiers over predict indicating that sales should have been higher, and deviate from actual sales results.

In this section we have presented our methodology for predicting future sales for short, 10 minutes, time bins by implementing and testing different Machine Learning classifiers on the sales dataset. Tree classifiers —M5P and REPTree— as well *Bagging* meta-classifier implementing both algorithms, offer high accuracy predicting sales for normal Web site operation. In the next section we perform an evaluation of a high response time effect on predictions and user satisfaction.

5.8 Response time effect on users

In Section 5.6 we have shown that the workload peak times coincide with high conversion rate for most applications of the OTA. Therefore, a loss in sales due to high response time is not apparent (Figure 5.12) as the fraction of buyers in the workload is also higher at these times. To counter-measure this effect, on the previous section, we have presented our prototype implementation to forecast expected sales for short time bins of

the workload with great accuracy results. However, as response time increases, classifiers over-predict sales; this section analyses this effect further.

In section 3.2, we have seen how Load on the Web servers increases with the concurrent number of requests and sessions. High load on servers usually translates into an increase in response time, which increases for three main reasons: server resource starvation, less dedicated system resources for a request; external B2B requests and database increased response time, low QoS on dependencies; and contention between resources, jobs waiting for blocked resources. Details of these reasons are further developed in [10]. Furthermore, not all applications response time increases in the same proportion with server load, as each application has differentiated resource requirements, especially for external resources. Going back to Figure 5.3, we can appreciate how *Hotels*, is the most affected application by server load, as it has a higher number of database queries compared to other applications, and its peak time coincides with external providers worst QoS. While *events* is the least affected application as it has no external dependencies besides the database.

Notice that, as it was shown in Section 5.6, peak load times coincide with high conversion rate periods for most applications of the OTA, where we have shown that all of the analyzed applications have a corresponding high CR when there are more users on the system and the QoS is worst. Therefore, a loss in sales due to high response time may not be apparent as the fraction of buyers in the workload is also higher at these times. Figure 5.12, exemplifies this situation where a high response time seems to maintain or even improve sales for most applications and the loss in sales might be undetected by system administrators, and most importantly, by management on high level reports. Comparing the actual sales with predicted sales (based on mostly non-overloaded system state) will highlight the net loss of sales due to surges in response time.

5.9 Response time thresholds and user satisfaction

A drop in sales might not be evident due to low QoS, especially at peak times, as it might coincide with a high conversion rate (CR) moment or the product might have slow search pages, increasing averages to buying customers. To overcome this situation without modifying the OTAs infrastructure (Section 5.7), and intentionally adding delay

to users like in [5, 7], we have proposed the use of Machine Learning classifiers to predict expected sales for short time bins, in our case 10 minute bins.

Figure 5.14 plots the relative percentage difference between actual sales and predicted sales as response time increases for each product of the OTA. It can be read as follows: the *Y-axis* is the percentage difference from 0% to -100% representing the loss in sales; and the *X-axis* plots the response time from 0 to 30 seconds. Notice that some applications do not start with values at 0 seconds, as their response might not be that low.

To produce the predicted data for this experiment, we have selected the *M5P* classifier, as it has less performance requirements than *bagging* meta-classifiers and the output tree model for predictions is more complete than REPTree from a human perspective. The output model is used for validation and to understand what features in the training dataset are the most relevant for predicting sales (See Section 5.7.1.1). Recall that each application has differentiated response time averages and also different conversion rates for each time bin.

For *ALL* products, actual sales start to deviate from expected values at about 3 to 7 seconds, and from 11 seconds have a huge drop compared to the expected sales. Next is the *flights* product, sales start to drop significantly between 4 and 8 seconds, and a huge drop after 14 seconds. For *hotels*, the first drop is between 3 and 7 seconds and the second drop after 10 seconds. The *events* product registers only one drop, after 7 seconds; recall that this product is more inflexible due to exclusivity and also has its peak CR during the morning contrary to the rest of the products. Restaurants on the other hand, has a very low first drop, between 2 and 4 seconds, and then also a low second drop after 7 seconds.

Table 5.2 summarizes inflection times for each application, where we have separated inflection points in two thresholds in accordance to the APDEX standard [80]: *tolerating* and *frustration*. At the *tolerating*, 1^{st} threshold, some sales are lost, but most users remain on the site until the *frustration*, 2^{nd} threshold, where a high percentage of sales is lost and more users abandon the site. Also, an average of sales loss is presented for each range. The rest of the users with response time lower than the thresholds are considered to be *satisfied*. For the analyzed OTA, there is a *tolerating* response time threshold between 3 to 11 seconds, where some sales are lost. In average the *frustration*

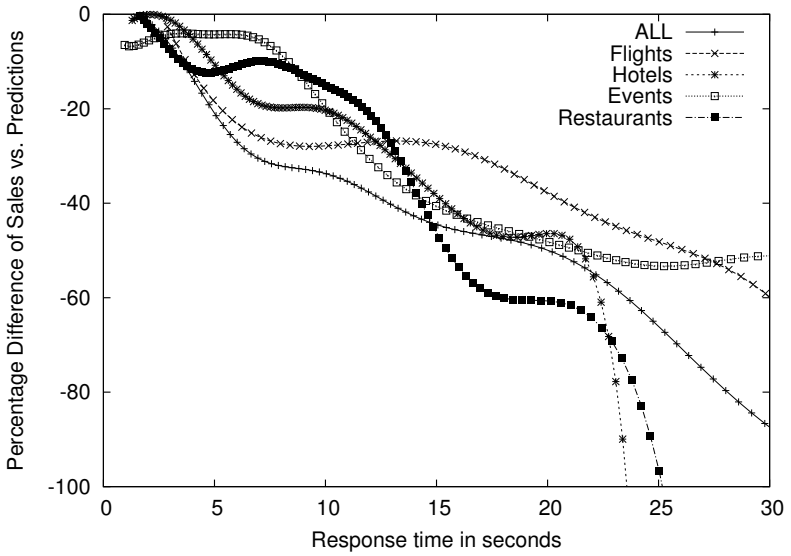


FIGURE 5.16: Percentage Difference of Sales and Predictions by Response Time

threshold for the analyzed OTA is at 11 seconds, where each increase in 1 seconds increases total sale loss by 3%. Even within the same Web site, each application has differentiated thresholds, and as products exhibit different flexibility, they also show different *tolerating* and *frustration* times and each should be treated separately from one another.

5.10 Experiments Validation

To validate the model and the methodology we have compared results for different days in the dataset where overload was not present as shown in Figure 5.1. Machine learning predictions are needed as sales are prone to seasonality effects and volumes for different times of the day cannot be fully predicted by linear tendencies. Results indicate that the prediction technique is valid to countermeasure conversion rate effects on peak times as models capture conversion rate variability by hour of the day, day of the week, and season.

5.10.1 Response time effect on user clicks

User clicks and session length are also affected by response time. Figure 5.17 plots the average number of clicks per session as response time increases for each product. As it can be seen for most products, there is a huge drop in the average session number of

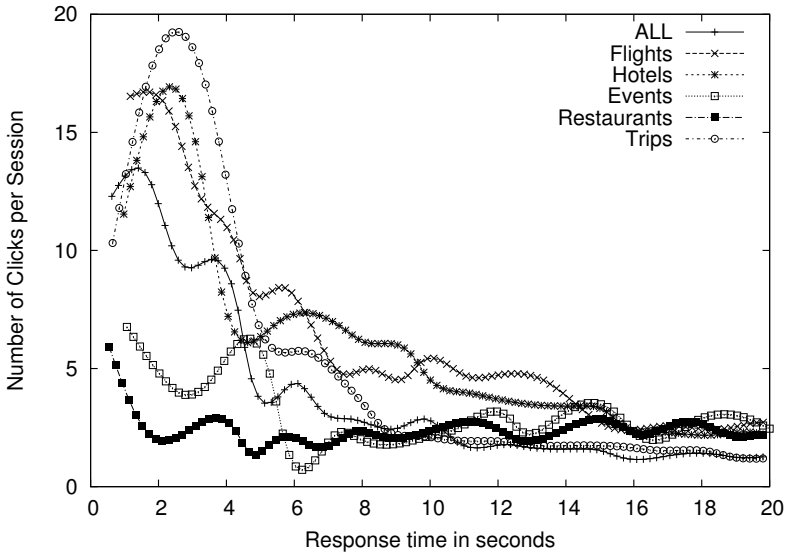


FIGURE 5.17: Average number of clicks decrease as response time increases

clicks after 2 seconds. For sites that do not sell products, the average session number of clicks or session length can be used to determine abandon times for users. However, as we have explained in Section 5.3, average session length is different by day of the week, a learning and prediction approach would be recommended to capture these features more precisely, but it is out of the scope of this study. These results are consistent with previous works on user behavior [79, 80]. The next sub-section continues with the discussion of results.

5.11 Discussion of Results

Results from this section show that the user's tolerating *response time* thresholds are higher for most applications of the OTA from previous literature, especially industry reports. Where our response time numbers are in line with Miller's [30] work on "Threshold Levels of Human Interaction and Attention with Computers". We believe that most of industry reports take static pages as a base measurement, which might not be representing the reality of many Ecommerce sites. Some Ecommerce sites such as the OTA presented here, have pages that usually take a long time to be generated, *e.g.*, *flight availability search*, that have a especial waiting page from which users assume a high complexity of the search, and thus are more patient on the results. The same can be observed for pages that involve external transactions, such as booking a restaurant, where

Appl	1^{st}thresh.	1^{st}drop	2^{nd}thresh.	2^{nd}drop
ALL	3-7s	5%	11-22s	55%
Flights	4-8s	28%	14-30s	55%
Events	-	-	7-25s	50%
Hotels	3-7	20%	10-18s	45%
Restaurants	2-4s	10%	7-18s	60%

TABLE 5.2: Percentage of sale loss by increasing response time: two incremental response time thresholds, and corresponding drop of sales in percentage

many checks need to be performed *i.e.*, credit card fraud, availability, re-check rates, Web Service calls, before the booking is made.

Furthermore, *tolerating* and *frustration* times are different for each application. For example the *events* application has exclusive content that cannot be purchased in online competitors, making it more inflexible than other applications such as *flights*, that has multiple competitors. Having different conversion rates and thresholds per application poses new challenges for differentiated and dynamic per-application QoS management during the day. Considering the current trend in Web ecosystem to observe lower conversion rates due to different factors (*e.g.*, rising competition, affiliation, meta-crawling, and changes in user habits such as multi-tabs[28]), online retailers will progressively support more traffic for less direct revenue by visit, increasing the importance of optimizing the number of servers without sacrificing sales.

The presented methodology enables online retailers to determine inflection points where sales start to be affected by the current application *response time*. Where resulting values can be applied on autonomic resource managers to optimize the number of servers and reduce infrastructure costs in cloud computing environments. Most importantly, optimizations should not be made to accommodate all load, but to provide the best QoS when conversion rates are higher, generally at peak loads. Our model could have benefited from more overload periods in the dataset to improve precision, however, even at the low number of samples of high response time for the less popular products, main inflection points and loss of sale tendencies can be obtained from it. As an additional contribution, results from this study had led the presented OTA to make important changes in their infrastructure to avoid high response times, especially at peak times with positive results.

5.12 Summary

We have argued that the effect of response time degradation can be hidden by the fact that peak load times can coincide with high conversion rates, *i.e.*, when higher fraction of visitors have intention to purchase. To overcome this effect we have introduced a novel methodology for studying what is the volume of sales lost in an online retailer due to performance degradation without modifying its application. We use machine learning techniques to predict the expected sales volume over time and look for deviations over the expected values during overload periods that may introduce performance degradation. Using such technique, we can quantify the impact of response time in the business activities of an online service without modifying production system.

We have tested the approach on logs from a top Online Travel Agency, using a 5 year long sales dataset, HTTP access log, and resource consumption logs for 6 months of 2011. From the obtained results we are able to identify inflection points where sales start to drop for different applications when response time is high. For the OTA, there is a *tolerating* response time threshold from 3 to 11 seconds, where some sales are lost, and a *frustration* threshold at 11 seconds, where each increase in 1 second interval increases total sale loss by 3%.

Chapter 6

Policies for Profit-Aware resource management

6.1 Introduction

During the last years, the Cloud Computing paradigm has been rapidly adopted to host Web applications due to its inherent cost effectiveness [19, 20]. It has also proven effective in scaling dynamically the number of servers according to simple performance metrics and the incoming workload. However, while some applications are able to scale horizontally [21], hosting costs and user satisfaction are currently not optimized.

A problem that arises with dynamic server provisioning or *elastic scaling* as it is referred by Amazon Web Services (AWS) [32], is deciding when to scale in number of servers and to what number. This is especially a problem as the only usable metric by default in AWS is CPU utilization [33]. For Web applications, CPU usage is not a good indicator of QoS or server load, as the CPU is rarely a bottleneck *e.g.*, for the datasets that we perform the experimentation, when the servers are overloaded, less than 45% of the CPU is being utilized [10].

The question of how many servers to keep at any given time it is currently an open question [32], as Web applications can be served at different response times depending on the number of concurrent users by Web server. However, this decision has an impact on user satisfaction and is directly linked with sales [17]. Not understanding how an

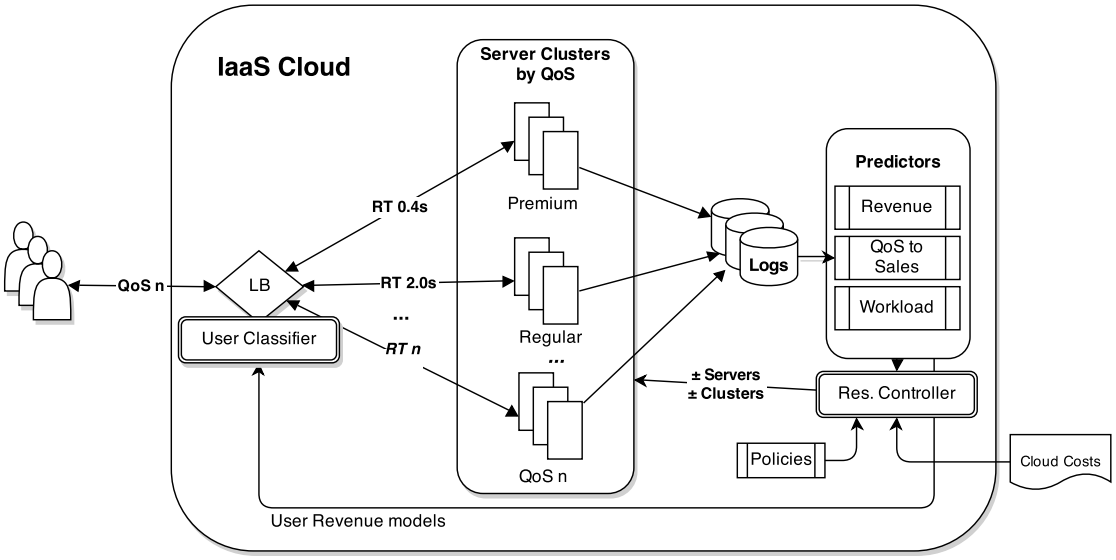


FIGURE 6.1: Execution environment of AUGURES

application behaves under load, or what the true limiting factors of the application are, may result in an ineffective or even destructive auto scaling configuration [33] *e.g.*, when malicious or spurious Bot traffic creates the load.

The following chapter proposes a new technique for dynamic resource provisioning based on a *profit-aware* policy. The policy leverages revenue and cost metrics, produced by Machine Learning techniques, with the intentions to optimize profits for consumers of Infrastructure-as-a-Service (IaaS) platforms. We base our proposal on user behavior models that relates Quality-of-Service (QoS), to the intention of users to buying a product on an Ecommerce site. In addition to the QoS-to-sales model, we use as inputs the current Cloud costs and a server capacity model to determine the maximum concurrent sessions per server to offer a specific response time. We also show that by classifying the types of users, and grouping them into clusters with differentiated QoS, the site can obtain further reduction in server costs, while keeping a high number of sales.

6.1.1 Methodology

Figure 6.1 presents an overview of the target Web execution scenario for the proposal, also presenting the main components of the system. This chapter focuses on the *Resource Controller (RC)* module, as work on the predictor modules has been presented on previous chapters.

As an offline process, the predictors are first trained from log data: the *Revenue* predictor is trained from the user navigational log as described in Chapter 4; the *QoS-to-Sales* model is trained from the sales and performance datasets presented in Section 3.2, and the methodology to produce the model can be find in Chapter 5; the *Workload* predictor is briefly described in Section 5.7. The resulting models are fed to the *RC*, that for every control period, it evaluates the predictions, and if profitable according to the predictions, decides to add or remove servers to the infrastructure.

In the case user prediction is enabled —in the high-level policies, the *RC* not only adds or removes servers, but can decide to create clusters that would return differentiated QoS to user requests. Figure 6.1 show three clusters: *Premium*, that have an average response time of 0.4 seconds; *Regular*, with an average response time of 2.0 seconds; and *QoS n*, representing another cluster with the target response time automatically set by the *RC*. When a new request from a user arrives to the system, the *Load Balancer (LB)*, first classifies the user to one of the defined classes, it forwards then the request to an available cluster with the appropriate QoS for the user, according to the profit policy set by the *RC*.

Experiments are performed on custom, real-life datasets presented in Section 6.4 . The datasets contain over two years of access, performance, and sales data from popular travel Web applications. Results from our prototype implementation can enable *profit-aware* resource management allowing the *self-configuration* of IaaS to an optimal number of servers. In this way, potentially reducing hosting costs for an incoming workload following high-level policies of business expectations.

6.2 Progress beyond the State-of-Art

Cloud computing is primarily driven by economics [19, 20] and it has been rapidly adopted due to its inherent cost effectiveness. For this reason, there are a number of studies on the topic of market-based resource allocation for Grid and Cloud computing. Most noticeable on scheduling mechanisms including: FirstPrice [87], FirstProfit [88], and proportional-share [89]. However, as Cloud computing has first evolved out of Grid computing, where jobs to be executed where mostly batch —rather than having real-time requirement. Most of these works therefore targeted *supercomputing* workloads

with a fixed number of resources and Service Level Agreements (SLA). Currently, Cloud platforms are used to host almost any type of application, and in particular it has become the most popular platform for new Web applications. Web traffic presents a very different, transactional based, QoS dependent workload.

Work related to improving the performance of Ecommerce servers has largely focused on providing better response times and higher throughput [22]. However, the most important performance metric from a business perspective is *profitability* [15, 28, 51]. management, i.e. by Littman et al. [59] uses Naïve-Bayes for cost classification and a Markov chain feedback approach for failure remediation. Other works such as [60] also take into account costs and resource allocation; in contrast with previous approaches, in this thesis we are focusing on the actual revenue that is lost by denying access to purchasing users, and not resource allocation costs. Authors perform admission control based on the response time of the system. Profitability of clouds has recently got attention; Liu *et al.* [92] propose a cost-aware approach to maximize the net profits that service providers may achieve when operating distributed cloud data centers in multi-electricity market environments. By capturing the relationship between SLA, cost on energy consumption, service request dispatching and resource allocation. Choon Lee *et al.* [19] introduce a pricing model to Cloud scheduling algorithms where profits are bound to a customer SLA specification. While most works [11, 19, 87, 92, 93] have focused on SLAs as a measure of optimization, in practice, from the IaaS consumer and business perspective, it is the final user satisfaction what finally leads to profits in online businesses.

Chen *et al.* [93] propose a profit-driven provisioning technique to maximize profits in the Cloud, by using auto-regressive performance model to predict the expected demand curve and determine when and how much resource to allocate and to maximize profits based on SLA specifications and costs generated by leased resources. While similar to our approach, there are also main differences: first, the workload employed for the study is the popular FIFA '98 dataset, composed of static requests, without customers in the workload; and second, it also uses static SLAs to calculate profits, while have shown in a previous work [17], that the QoS that Web users expect varies during the day and is different by application, one of the main techniques used in our approach to reduce costs in servers.

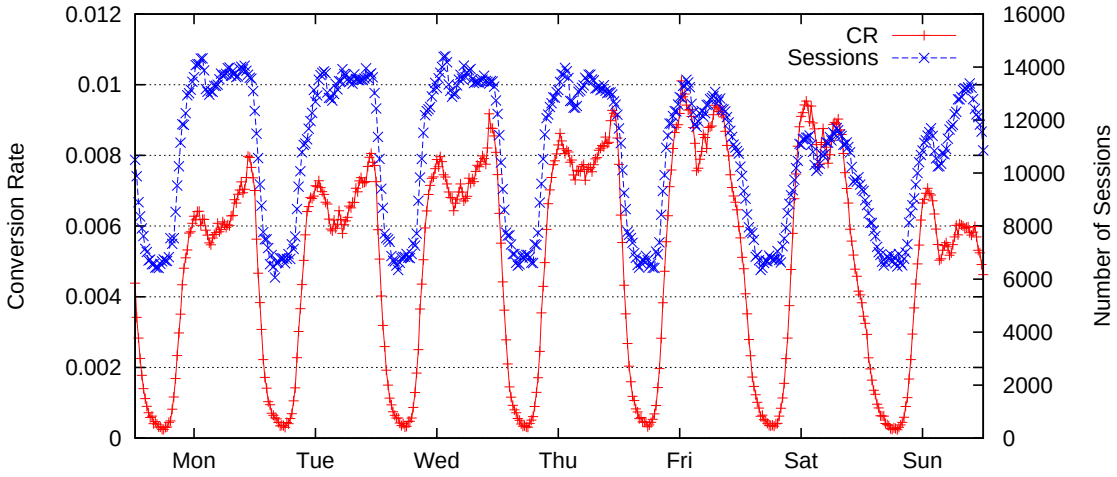


FIGURE 6.2: Number of sessions and Conversion Rates over a week period

Chen *et al.* [90] also argues that existing SLA based approaches are not sufficient to address performance variation and customer satisfaction. They present a model of customer satisfaction, leveraged by a utility-based SLA to balance performance and the cost of running services and two scheduling to make tradeoffs between profit and customer satisfaction. In previous works [29, 91], we have also evaluated the use of utility functions to set SLA targets, however this work differentiates in two ways: first, we base our approach on real user satisfaction, while [90] based their approach on an synthetic satisfaction metric based in utility; second, our experimentation is done to a production Ecommerce workload, while [90] is on video encoding workloads.

6.3 Conversion Rates as a function of Response Time

Conversion Rates (CR) are not static; CRs vary according to time of the day, application, day of the week, or season. Figure 6.2 presents both the CR and number of sessions for the site during a week period on separate Y-axis respectively. It can be seen how while the CR follows the daily session traffic patterns, it is also different per day. While the number of sessions is greater on Mondays, decreasing until Saturday, the CR has the opposite effect, and increases throughout the week. Saturday is the day with the overall higher ratio of buyers, while Sunday is a special case, it has the lower ratio of buyers compared to the number of sessions. There are also other CR peaks, Monday morning, just before lunch time and after office hours. The ratio of buyers is higher in general when there are more users in systems, as users buy at similar times of the day.

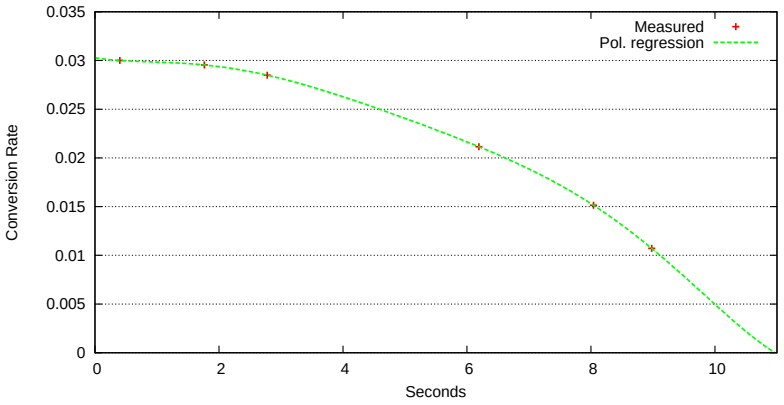


FIGURE 6.3: Decreasing Conversion Rate as Response Time increases

In addition, CRs are particularly sensitive to fluctuations in service response times. Figure 6.3 presents a static sample of how CRs are affected as response time increases. The CR at 0.4 seconds —the minimum recorded by the application— is 3% and decreases to 0.5% at 10 seconds. More than 25% of sales would be lost at 5 seconds of response time, while at 10 seconds, 80% of sales would be lost. In our prototype implementation, presented in Section 6.5, this model is calculated for each time of the day the prototype runs in. Since CRs vary during the day, so does the effect of response time. What it is important to notice from the figure, is that there is not a linear relation of sales loss *e.g.*, between 0.4 and 1 seconds, very little CR is lost; this allows our resource controller to save in servers. Further details of how the model is calculated are given in [17].

In Chapter 5 we have shown that CRs are usually higher when there are more users on the system, thus, when the incidence of high response times is higher. This situation makes particularly important, to give good QoS to incoming sessions at specific times of the day. We have also shown that this is application dependent, *e.g.*, for the restaurant reservation application, this time is just before lunch hours. Conversion rates being higher at peak times also hides the total impact of poor QoS during peaks. For this reason we had presented a methodology using Machine Learning for predicting the impact on sales as QoS (RT) degrades. In the present study, we have applied the methodology on a larger and newer dataset detailed in the next sub-section.

6.4 Datasets and Server Capacity

Evaluation of Web application resource consumption requires realistic workload simulations to obtain accurate results and conclusions. This section presents the different datasets provided by Atrapalo.com from their production environment. The datasets are produced by special probes in the application, logging not only HTTP access data, but detailed performance data about the resources consumed by the request and the current state of the server.

The datasets used for experimentation consists of two years of user HTTP requests to Atrapalo site, from February 2011 to February 2013. The data is composed of over 2.1 billion requests, representing around 480 million sessions. Where the average server response time is 1.36 seconds, the average database service time is 0.13 seconds, and for requests that involved external Web services the average is 0.64 seconds per request. A similar, but shorter dataset has been characterized in Chapter 5 .

We have also been given access to over 7 years of sales history. As vacation products suffer from great variation of sales and visits according to high and low seasons, day of the week, or the proximity of a holiday; while ticket sales varies according to the availability of the event on competing sites and the limited availability, sometimes causing rush periods. We use this dataset for our sales predictor, that captures the variability of conversion rate presented in Section 6.3 Sale volumes for these datasets are not reported due to confidentiality limitations.

6.4.1 Server Capacity

Not understanding how an application behaves under load, or what the true limiting factors of the application are, may result in an ineffective or even destructive auto scaling configuration [33]. As Web applications can be served at different QoS, depending mainly on the number of users that consume resources concurrently per server, to be able to set response times to a desired level, first the servers need to be benchmarked. From the performance datasets available, we are able to estimate with great accuracy the capacity of the application under production load. We use this information as input for our prototype implementation in later sections.

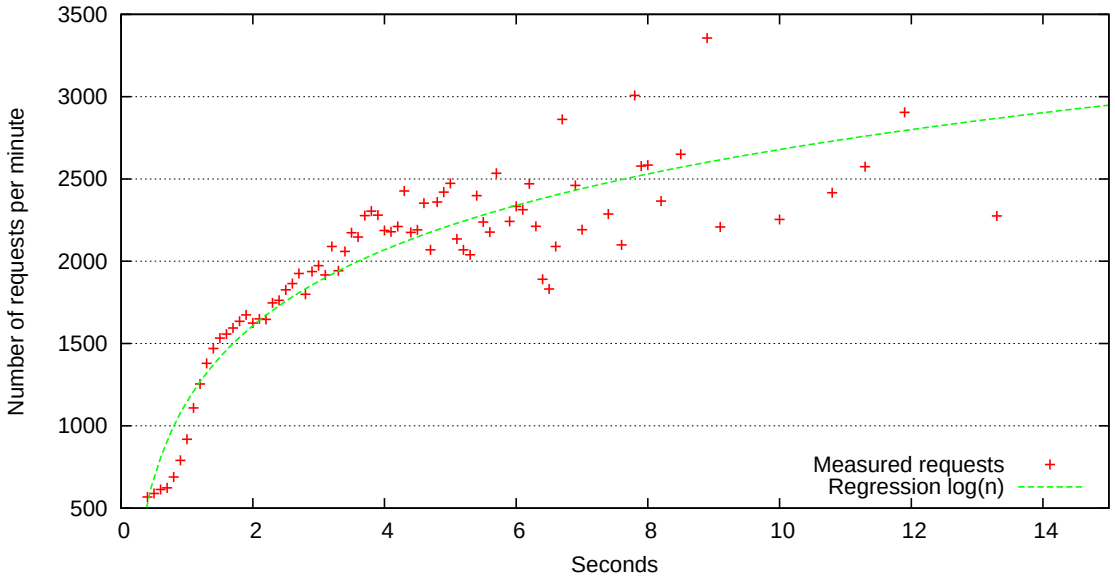


FIGURE 6.4: Average measured and modeled server capacity

Figure 6.4 shows the measured and modeled sever capacity of a single server in Atrapalo infrastructure. It shows how the Response Time to serve Web requests goes from 0.4 —the lowest the application can serve— to 15 seconds, as the number of requests per minute grow from 500 to 3000. It is important to remark that Atrapalo.com host their own servers without the use of any Cloud provider, as the company has over 12 years of online presence, and their services planed before Cloud offerings were available at their location. However it is currently being considered a migration to the Cloud.

The capacity of the servers with this dataset used as input in the prototype implementation follows the logarithmic regression: $y = 664.96ln(x) + 1147.7$

6.4.1.1 Server specification

Since Atrapalo application runs on physical servers, this sub-section gives an overview of the server characteristics. Such information is important along with the server capacity model from the previous section, to be able to compare them to current offerings at AWS [32] to set prices accordingly for our prototype.

The servers are composed of Intel XEON processors with 4 cores, with hyper-threading enabled and 16GB of RAM, using conventional SAS hard drives. Using [103] to compare the processing power of Atrapalo physical server to to Amazon's offering, each of Atrapalo servers have at least 2x processing power to Amazon's current largest offering at

time of writing. We found that the main performance difference due to Atrapalo servers not being virtualized; also that resources are exclusive, not shared with other tenants.

6.5 The AUGURES prototype

This section presents AUGURES, our prototype implementation of a profit-aware resource provisioner and load balancer. The overview of the implementation architecture can be seen in Figure 6.1.

Algorithm 1 Main AUGURES algorithm

```
1: while T do                                              ▷ For each time interval
2:     WL ← predictExpectedWorkload(T)
3:     profitByRT ← calculateProfitByRT(t, WL)
4:     optimalServers ← maximizeProfitWithPolicies(t, profitByRT)
5:     needServers ← optimalServers − currentServers
6:     setServers(neededServers)
7: end while                                               ▷ Until system stops
```

Algorithm 1 presents the main process of AUGURES, where, t represents the current time of the day, while T is the control cycle period in number of minutes. T provides the granularity in which resource re-provisioning will take place. AUGURES runs every T minutes to predict the most optimal server configuration according to the time of the day and the predicted workload.

The first step in AUGURES is predicting the expected number of requests and user sessions for the next T minutes *e.g.*, 20 minutes. The workload prediction mechanism employed is explained in further detail in a previous work [29] and it is not the objective of this study. Second, an array with all the possible server configuration options by response time (RT) is calculated and returned. The array *profitByRT* contains: the number of servers (cost), conversion rate (sales), and profit for each possible RT for the T period. The algorithm of the procedure *calculateProrfitByRT*, is the core of AUGURES, is explained later, in the next subsection.

After obtaining the different profits for each response time combination, the array *profitByRT* is used as input to the *maximizeProfitsWithPolicies* procedure, to select the optimal server configuration according to high level policies. Policies can include constraints such as minimum and maximum number of servers (cost), maximum and minimum response times (QoS), target conversion rates, date and times, and a combination of these

features. After constraints are checked, the response time policy with the maximum profit is selected to scale out or down the current number of servers.

6.5.1 Calculating profits by response time

Algorithm 2 Calculating profits by response time procedure

```
1: procedure calculateProfitByRT(t, WL)
2:     for RT ← MIN_RT, MAX_RT do                    ▷ For each possible response time
3:         reqsPerServerRT ← maxRequestsToAchieveRT(t, RT)
4:         neededServersRT ← ceil(WL/reqsPerServerRT)
5:         serversCostRT ←
                  (neededServersRT ∗ CostPerServer) + staticServersCosts
6:         CR_RT_t ← getConversionRateForThisRTandDate(t, RT[, userClass, product])
7:         revenueRT ← WL ∗ CR_RT_t
8:         profitRT ← revenueRT − serversCostRT
9:         [satisfactionRT ← getUserSatisfactionRT(RT)]                ▷ Optional
10:        [profitSatisfactionRT ← profitRT − satisfactionRT]          ▷ Optional
11:        profitByRT[RT][ ] ← {                                  ▷ Response time array
12:            neededServersRT,
13:            serversCostRT,
14:            CR_RT_t,
15:            revenueRT,
16:            profitRT,
17:            [satisfactionRT]
18:            [profitSatisfactionRT]
19:        }
20:        RT ← RT + STEP_RT
21:    end for
22:    return profitByRT
23: end procedure
```

The core of the AUGURES algorithm relies on the profit procedure *calculateProfitbyRT* shown in Algorithm 2. The profit function iterates from the minimum response time, *MIN_RT*, that the application can provide to the maximum response time defined in the system or server capacity limitations, denoted by *MAX_RT*. From the server capacity model in Section 6.4.1 to Atrapalo Web servers, *MIN_RT* is 0.4s, while we have set *MAX_RT* to 15 seconds, when the server is handling about 3000 requests per minute. After the 3000 request limit, the server is considered overloaded and starts giving negative throughput compared to the maximum —the server starts trashing. Server overload for a similar dataset has been explored in [10]. For our experimentation we have set the precision *STEP_RT* to 0.1 seconds.

The first part of the profit procedure consists in evaluating how many sessions we can fit per server to achieve the current required *RT*. The total number of servers needed is the *ceiling* of *WL* by request per server, again using the server capacity model from Section 6.4.1. The total server cost in this case, is the product of the number of servers needed to achieve the current *RT* and the cost per server at the given time of the selectedCcloud provider. To the per server cost, a fixed cost is added representing the cost of fixed supporting servers *i.e.*, DNS, load balancers, firewalls, and database servers. Additionally, server costs can be dynamic depending on the cloud provider and the desired type of server instance, *i.e.*, spot instances, bidding, and second market instances [94]. For this work we assume server prices to be fixed, but they can easily be extended for prices to be updated every time *T* that the prototype runs.

The second part of the profit procedure consists in evaluating how the currently selected *RT* affects the Conversion Rate (*CR*). The *CR* as explained in Section 3.1.3, is the ratio of buying sessions in the workload, and the higher the *RT*, the lower it will be. The model for predicting how the *CR* will be affected by the given *RT* is described in Section 6.3. After the CR by RT (*CR_RT*) is found, the revenue is calculated by multiplying *CR_RT* by the predicted number of sessions for period *T*, times the value per sale. The value per sale is dependent on the application and types of product offered. The value per sale can be either the gross value or directly the gain margins per sale if available.

The third part of the algorithm would include user satisfaction for the given *RT* if provided. The user satisfaction can be a value attributed by the user to the given *RT* or the value of the reputation of the company, or the probability that the user will return to the site given the current *RT*. [17] gives more insight on how user satisfaction can be calculated from the users' perspective, not only by the sales from the Ecommerce retailer. After obtained the potential profit for this *RT* in time *T*, the total profit can be calculated and added to the array. The algorithm continues by looping each *STEP_RT* until *MAX_RT* is reached, and returns the *profitByRT* array.

The next section describes the values for our implementation of the algorithm presented in this section and the results.

6.6 Experimentation

To test the applicability of the algorithms presented in the previous subsections, a prototype has been implemented that is able to reply the dataset described in Section 6.4. The dataset is grouped and splitted in T minute intervals, in our case 20 minutes. 20 minutes is chosen as the interval, as it is high enough to provide accurate predictions by our Machine Learning predictors, represents the QoS of the server, while it is also high enough for resource provisioners to act upon to apply changes in number of servers. The server cost per hour is calculated from the specs of the servers of the current application compared to the offering of Amazon EC2 servers[103] at time of writing. To the servers cost, it was added the bandwidth cost, manually calculated from values of Atrapalo.com. For the following experiments, the server cost was manually calculated at a fixed at 5 USD per hour. This number, for the time being is divided by T; however Cloud providers such as AWS [32] charge by the entire hour, not the fraction. Taking into account this limitation could be an improvement to our proposal.

For the following experiments we do not include fixed servers costs *i.e.*, database, cache, and supporting servers; as we want to evaluate just the costs of dynamically provisioned servers. As sale value, in the next experiments, we assume that the net revenue for each sale is of 1 USD, the revenue that can be expected of a concert ticket sale for an online retailer, such as the one presented in Section 3.1.1.

6.6.1 Metrics for dynamic provisioning

Cloud resource provisioners can currently use system performance metrics in order to scale out or down in number of servers. A problem that arises with dynamic server provisioning or *elastic scaling* as it is referred by Amazon Web Services (AWS), is deciding when to scale in number of servers. This is especially a problem as the only usable metric by default in AWS is CPU utilization [33], which for Web applications is not a good indicator of QoS. The CPU utilization can tell how loaded is a servers if the main bottleneck is the CPU, but for Web applications this is hardly the case [10, 33]. Furthermore, there is no direct relation between the utilization of a resource and the QoS of the service. For Websites, response time can be a better indicator of QoS of the current infrastructure and relate to user satisfaction [17].

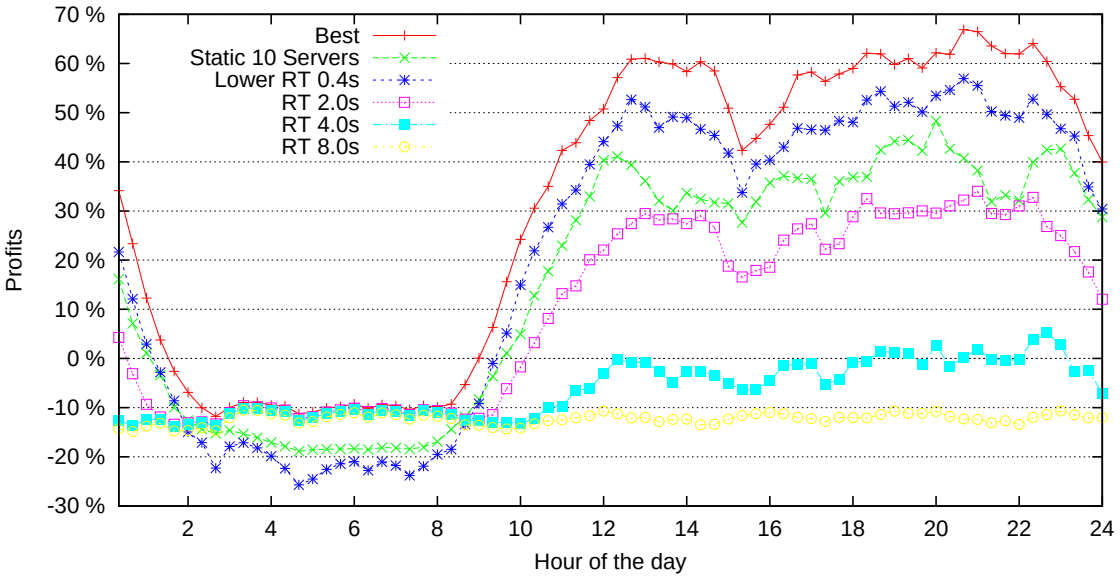


FIGURE 6.5: Profits obtained by different response time strategies over a 24hr period

As to what maximum response time to serve requests some authors adhere to what is called the *8-second rule* [79, 80] to determine the maximum threshold of user attention. While the APDEX standard, establishes 4 seconds as the threshold for unsatisfied users [80]. Industry leaders such as Google advocates a 2 second limit for page load times [17]. The next subsection compares the profits that can be obtained with AUGURES to different response time thresholds, and our proposal, a *profit-oriented* metric based on the models obtained from the provided production datasets.

6.6.2 Profit-aware policy vs. fixed Response Times

Figure 6.5 presents the averaged profits for every T period in a day when running AUGURES over the Atrapalo dataset over a 24-hour period with the cost and revenue variables from previous sections. It can be noted that the profits follows the pattern of the conversion rates (CR) and number of session variation throughout the day as in Figure 6.2 as expected. From the figure, t can be seen that profits are close to zero or even negative during the early morning as CR is very low at this time, as well as the number of visits to the site. Profits grow very steeply during the early morning, have a dip during lunch time and continue to grow and peak to the evening.

The first curve in Figure 6.5 represents the profit-aware strategy of AUGURES, *Best*, while *Lower* presents the best possible RT for the application, 0.4s. The rest represent the strategies of maintaining fixed 2, 4, and 8 seconds. From the figure, it can be

seen that the most expensive, less profitable strategy is the *Lower* strategy during early morning due to requiring more servers. While *Lower* is the second best during day time, when CRs are higher and there are more users in the system.

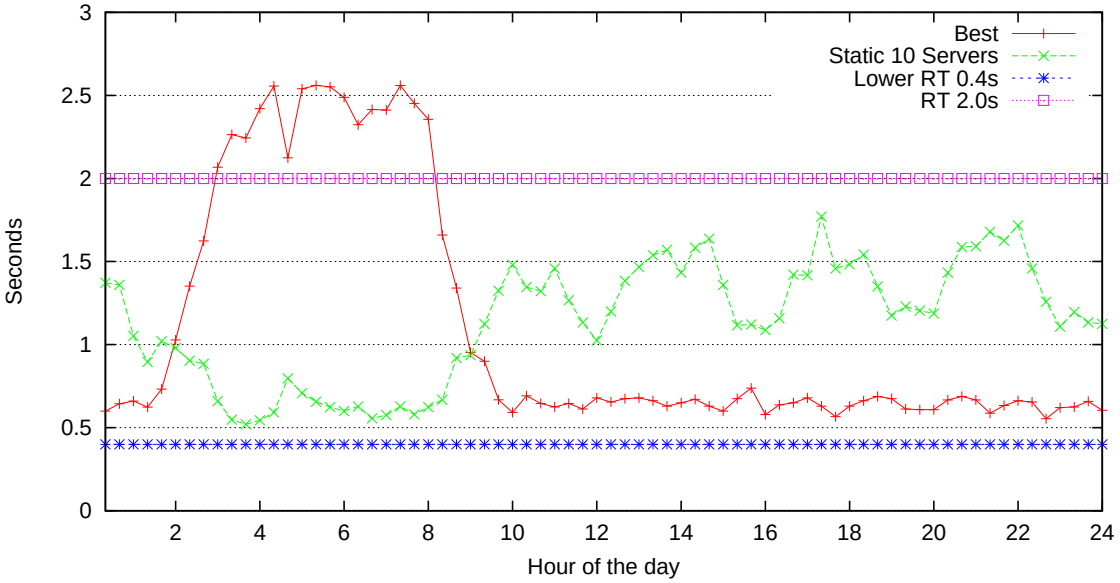


FIGURE 6.6: Response time in seconds for Best, Static 10 servers, Lower, and 2 seconds strategies

The *Best* strategy benefits by saving costs in servers when the conversion rate (CR) is low and it does not compensate to give a better response time (RT) to users. It does so by varying the RT and number of servers needed for every T period as can be seen in Figures 6.6 and 6.7 respectively. During the night and early morning, it offers up to 2.5s in response time as conversion rates are low during these times. This is due to most of the traffic during these hours corresponds to crawlers and other automated bots, and the CR is very low [28] at this time. During daytime, the chosen RTs are between 0.6 and 0.8 seconds, it does not offer the best RT —0.4— as the ratio of user sessions per server (see Figure 6.4) and the CR improvement is not the most profitable. On the contrary it lowers the number of required servers closer to the rest of the fixed strategies.

Figures 6.6 and 6.7 also show the RT under a fixed number of servers —10 servers— as in a static infrastructures without dynamic provisioning. Under a static configuration, the RT is dependent on the number of sessions arriving to the system. It can be seen that during night time, the response time is low, while during the day the response time is at 1.5 seconds in average, losing a small percentage of sales. As a consequence of increasing traffic or a traffic spike, a static server configuration will degrade response times —losing sales—, but keeping fixed server cost prices. A dynamic policy would increase hosting

prices temporarily according to maximum defined values to accommodate load, and keep the desired *RT*.

The rest of the strategies provide a fixed QoS (*RT*), it can be seen the difference in number of servers needed from 2 to 8 seconds is minimal compared to the Lower strategy. While for this application is profitable to give low *RT*s, the number of servers needed to serve the site at 2 seconds is 8 servers during daytime. About one third of required servers for the *Lower* strategy, where up to 25 servers are needed. The response time of a static provisioned infrastructure, *i.e.*, 10 servers all day, will vary during the day. Resulting in the worst response time when there are more users on the system and the CRs higher [17].

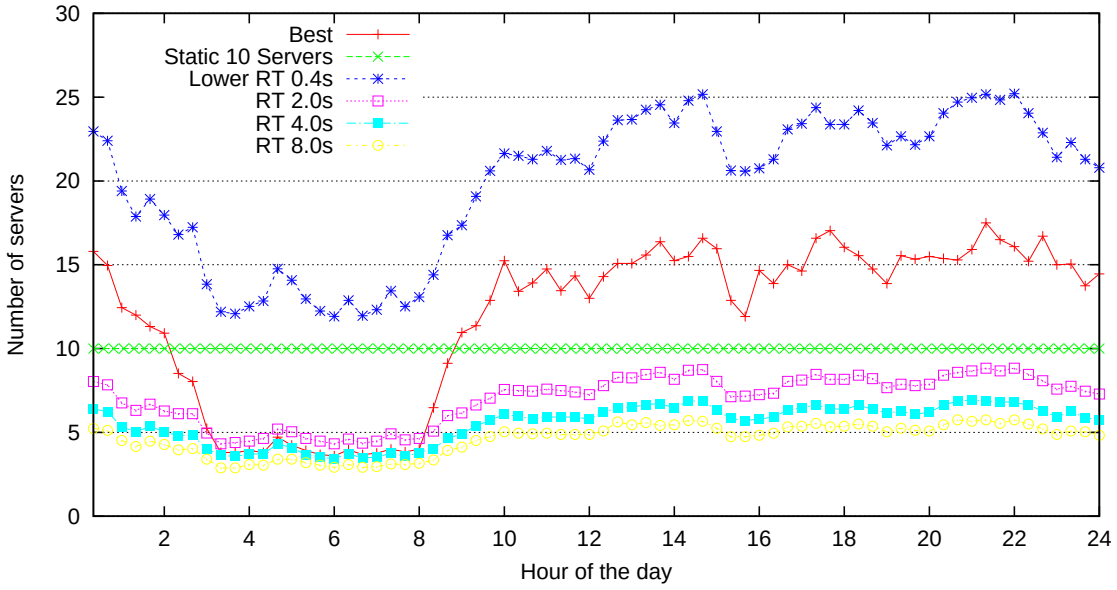


FIGURE 6.7: Number of servers needed by each strategy during a 24hr period

Figure 6.8 shows the *CR* by *RT* of the different strategies, it can be seen that the *Lower* strategy has the best *CR*. The *Best* strategy on the other hand, while it does not achieve all of the sales, it is very close to the best possible with the *Lower* strategy. This small difference in lost *CR*, allows the *Best* strategy to save up to one third in number of servers, reducing infrastructure costs. For this reduction in number of servers to be a significant gain in economic terms, it depends of course in the cost per server, the sale value, conversion rates, time of the day, and the application.

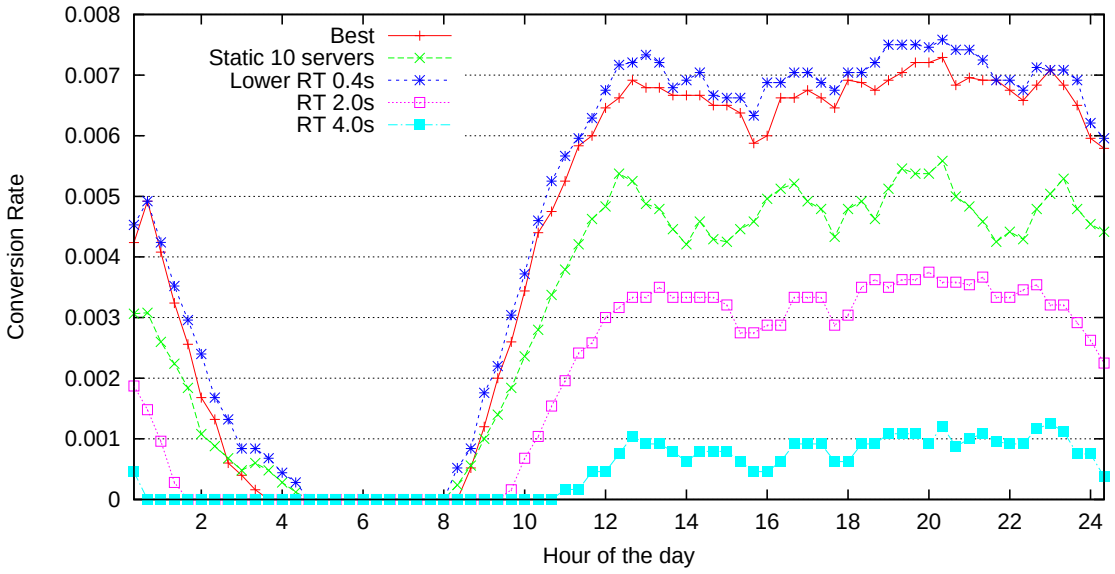


FIGURE 6.8: Obtained Conversion Rates for the different strategies during a 24hr period

6.7 Classifying users

The previous section has presented the potential benefits in business profits of having a profit-aware resource provisioner. While the Best strategy can provide improvements by adjusting the target response time (RT), thus the number of servers needed, all of the incoming requests are considered from the same class. The next section describes our extension to AUGURES to dynamically classify requests into classes of users and the potential benefit of combining our resource provisioner with session classification.

6.7.1 Classes of users

The first step to classify users is to select the classes of users. While this task can be performed by clustering techniques, the company that provided the dataset already has users classified into: regular and buyers visitors, API users, internal requests, intranet, and crawler traffic. Crawler traffic can be divided into: search engine crawlers, which are beneficial; and malicious bots or Web scrapers. Web scrapers, simulate being regular users and fetch content from the target site to later brand it as their own, and benefit for not having to produce their own content, paying contracts ie. Hotel Chains or Flights Reservation systems [28]. API users in the dataset are clearly identified as such, as they need login credentials and access specific sub-domains and URLs. For this work we are

interested in differentiating buyers to regular visits, and content-stealing bots, while we discard API requests. The next sections describes buyer classification.

6.7.2 Predicting for Buying sessions

In Chapter 4 we have built a prototype implementing both Machine Learning algorithms and Markov chains in order to predict the intentions and final outcomes of Web sessions. As with this chapter, our sample Web application is an Ecommerce site, thus the outcome of interest for our study is whether a session will buy a product or not in its current visit. Results from Chapter 4 demonstrated that the user intention for visiting the site such as buying a product on the current session, can be predicted from available navigational information and time features, from the first clicks to a certain precision.

We have used the previous work to test session buying prediction for session admission control. The main idea was that if there were traffic spikes on a static infrastructure, and not all sessions could be served, to select them by the probability of the session buying a product until the maximum capacity was reached, redirecting remaining sessions to a wait page. This resulted in sustained profits during traffic spikes. On the current study, we apply user classification to group users and apply the algorithm presented in Section 6.5 to each group.

Algorithm 3 AUGURES with user prediction

```
1:  while T do                                              ▷ For each time interval
2:      for all userClass ∈ definedUserClasses do
3:          WL[userClass] ← PREDICTEXPECTEDWORKLOAD(T, WL)
4:          profitByRT[userClass] ←
                CALCULATEPROFITBYRT(T, WL, userClass)
5:      end for
6:      optimalServers ← MAXIMIZEPROFITWITHPOLICIES(T, profitByRT)
7:      for all userClass ∈ definedUserClasses do
8:          needServers[userClass] ←
                optimalServers[userClass] − currentServers[userClass]
9:      end for
10:     SETSERVERS(neededServers)
11: end while                                               ▷ Until system stops
```

Algorithm 3 presents the changes to the main algorithm to work with different classes of users. Basically, it is same algorithm as Algorithm 1, except that it loops over the different defined classes. This change requires that the functions in algorithm 3 to handle

the *userClass* parameter, and for data structures to be divided/indexed by the *userClass*. The *predictWLClass* procedure will predict the number of requests expected for this user class from the predicted total (*WL*). The prediction is handled by the Machine Learning module described in Chapter 4

Predicting the user class at each click and forwarding them to the appropriate class server cluster is a different and separate process. The process of predicting the class of an incoming session, is a separate, real-time process described in detail in Chapter 4

6.7.2.1 Prediction results

In average, 9.7% of the sessions are classified as buyers and 74.3% of the requests in this group correspond to buyers. Increasing the CR of this group in average from 0.7% to 6%. The *CR* for each group can be seen in Figure 6.9. The figure shows that buyers are effectively being grouped together in the same cluster, while in the *NonBuyer* group; there are some buyers, but fewer that with the *Best* strategy.

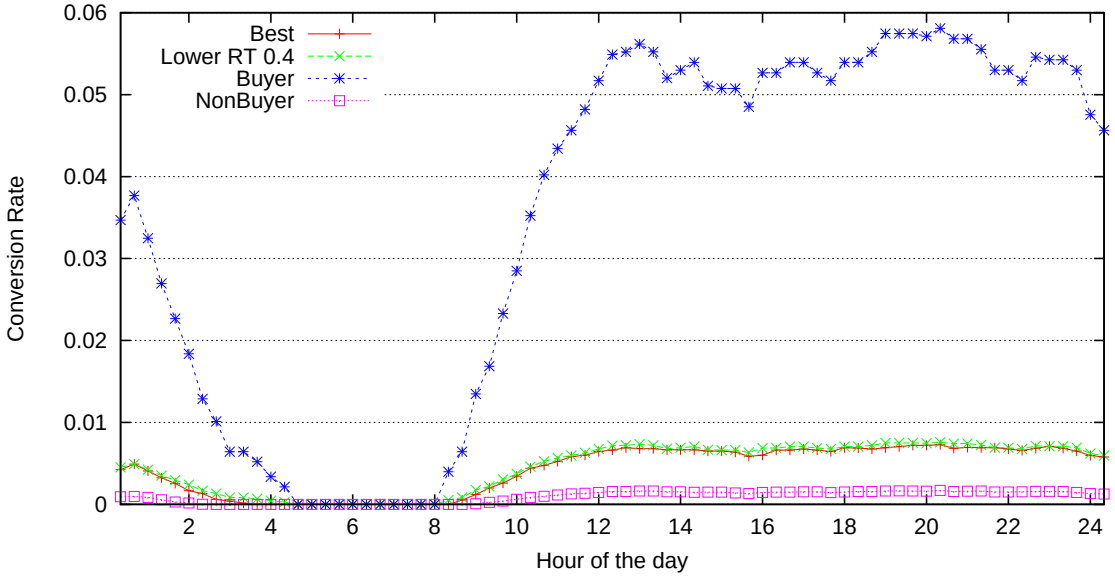


FIGURE 6.9: Conversion rates per group of server for *Buyers*, *NonBuyers* compared to *Best* strategy

6.7.3 AUGURES with *Buyers* and *NonBuyers* groups

Figure 6.10 presents the profit results for each group user classes and the combined group and the *Best* strategy from the previous Section. It can be seen that most of the profit is made on the Buyers group, while the *NonBuyers* group is close to zero in profits.

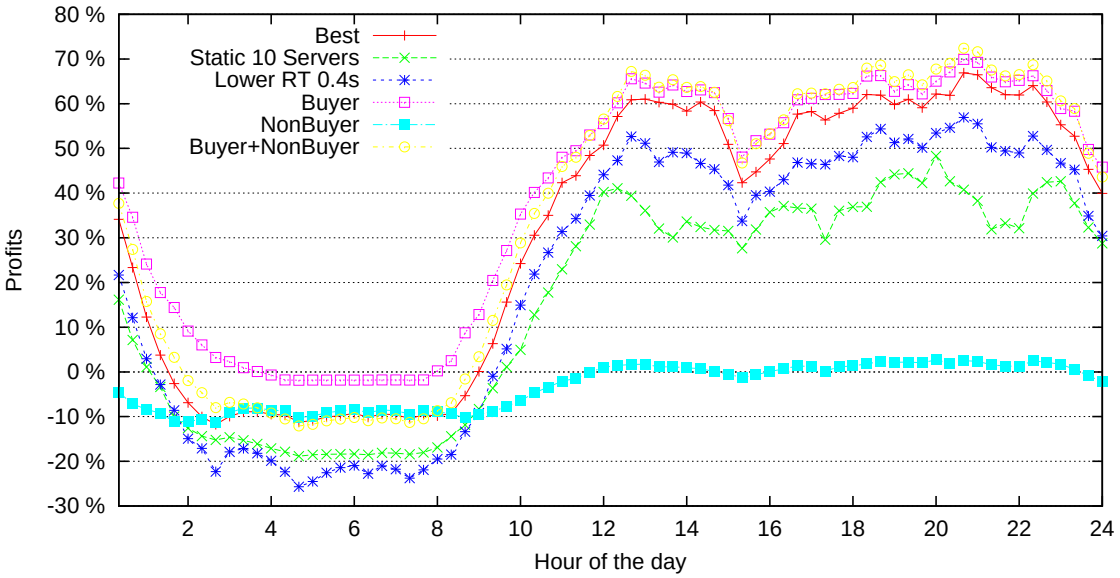


FIGURE 6.10: Profit obtained by *Buyers* and *NonBuyers* compared to *Best* policy

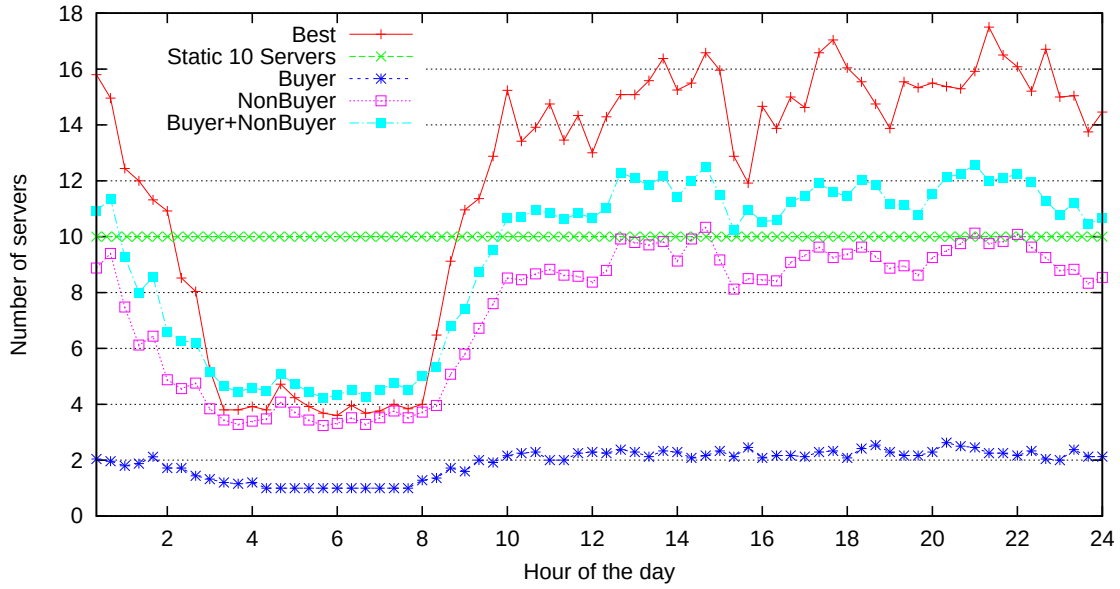


FIGURE 6.11: Number of servers for *Buyers* and *NonBuyers* compared to *Best* policy

Classifying users into groups and allowing different QoS for each group and server resources can be optimized further. Figure 6.11 shows the number of servers need: the *Buyers* group, 2 servers; the *NonBuyers* groups, 10 servers; and the *Best* strategy 16 server in average during daytime. Combined *Buyers* and *NonBuyers* used about 12 servers, giving a saving of 25% over the *Best* strategy.

Separating groups into user classes also enables to give differentiated QoS to each group. While the *best* strategy tries to balance more server costs and profits, the *buyers* group, as needs less servers, can provide a lower response time, and improve the likelihood of

Policy	Session Percentage	Buyers Percentage
Policy 0: Best strategy	no user classification	
Policy 1: Winner strategy	9.7%	74.3%
Policy 2	20.2%	91.9%
Policy 3	13.6%	83.0%
Policy 4	5.2%	53.5%

TABLE 6.1: Different number of sessions and buyers for different prediction policies

purchases further in this group.

6.7.4 Multiple predictions

Since predictions in the implemented numerical classifiers do not give hard, buy/do-not-buy predictions but a probability for the session belonging to each class, different classifications can be performed.

This change requires the procedure *predictWLClass* in Algorithm 3 to return the probabilities of *WLclass* belonging to the different *userClass* groups. The profit for each of the multiple predictions is stored, and then the *maximizeProfitWithPolicies* procedure will select the most profitable one according to policies.

We have selected to use 5 different policies. The first policy, *Policy 0*, corresponds with the *Best* strategy from Section 6.6.2. The reason for this is that if the profit is not improved by user classification, the *Best* strategy should be used. Furthermore in cases where few servers are needed, the *Best Policy 0* will reduce server costs as there is no need to split them in groups, also reducing IT management complexity, so it is favored if not considerable savings can be achieved by the classification. *Policy 1* corresponds to the *winner* strategy. The *winner* strategy is the default selection by classifiers, it select the class with the higher probability. *Policy 2*, where the probability of being a buyer is greater than zero. *Policy 3*, any click with *NonBuyers* probability less than 90%. *Policy 4*, any *Buyer* probability greater than zero and less than 90% probability of being *NonBuyers*. Table 6.1 summarizes results for each class.

Figures 6.12, 6.13 shows how the different prediction policies/strategies can be used to optimize resources to a finer granularity according to the expected *CR* and *WL*.

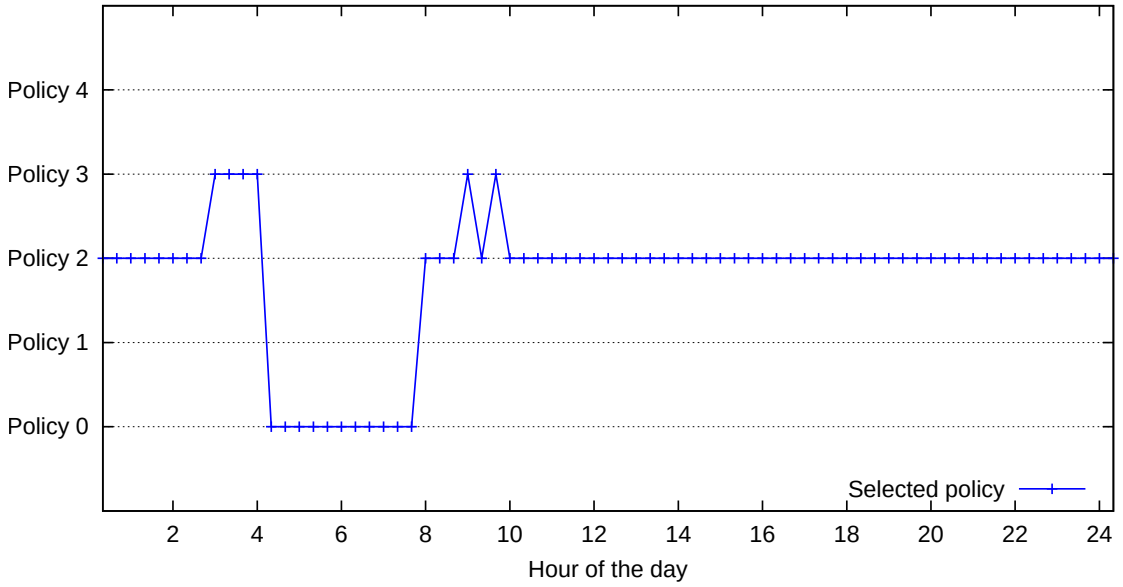


FIGURE 6.12: Different policies selection for every T a sample 24-hour period

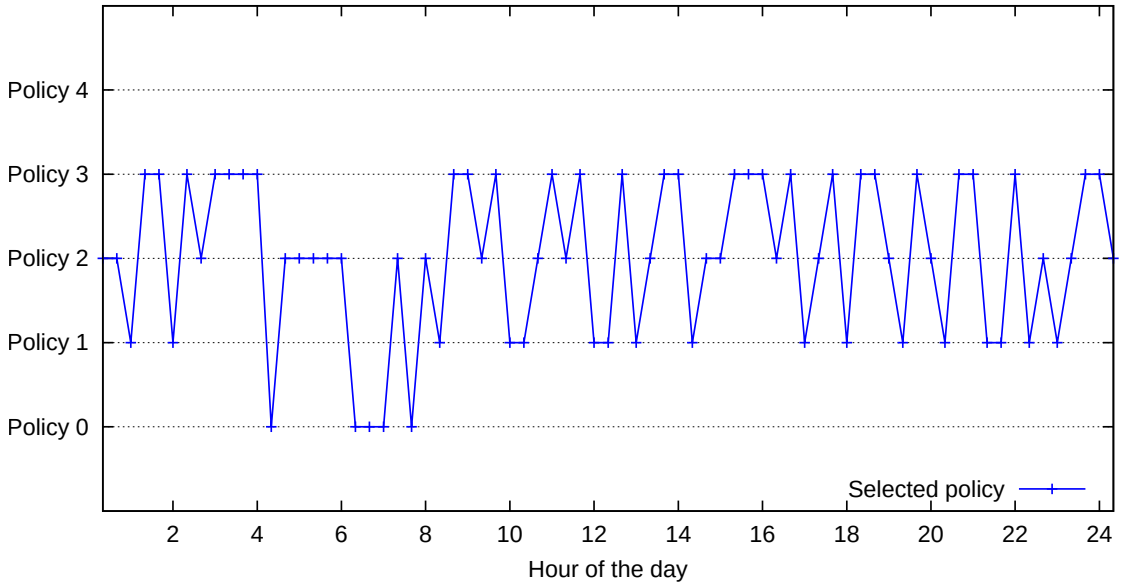


FIGURE 6.13: Different policies selection for every T a sample 24-hour period

6.7.5 Classifying Bot traffic and Admission Control

From domain knowledge of the OTA and their scenario (See Section 3.1.1), in Section 4.5 we have also made some tests to classify in real-time content stealing bots to the site. These are not to be confused with search engine crawlers that are beneficial to the company search rankings and require a high QoS. For the dataset presented in Section 6.4, 17% of the total requests have been classified as belonging to unwanted *Bot* traffic and manually verified. Further details on Bot detection with Machine Learning can be found in Section 4.5. In the case of the presented dataset the prediction technique

could eliminate 17% of total requests. As an extension to the AUGURES strategy, the *AUGURES + Bots* strategy, besides classifying users, it also predicts if the session might come from an automated bot to ban such traffic.

Figure 6.14 presents the number of servers required by classifying users and banning *bots*. Figure 6.15 presents the profits of classifying users into *buyer* and *not-buyer* while banning *bots*.

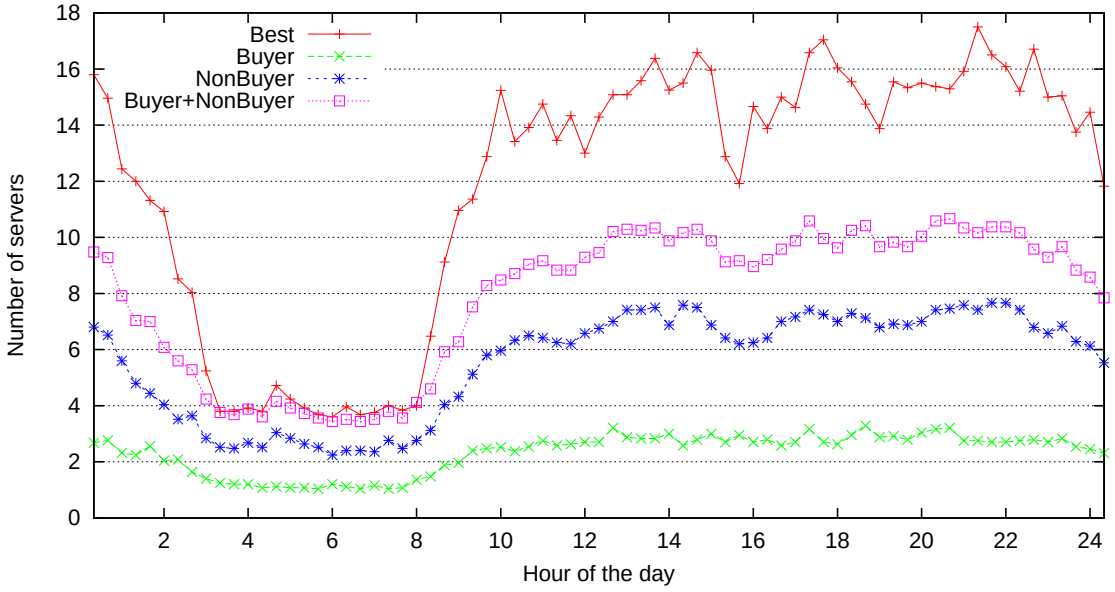


FIGURE 6.14: Number of servers needed by *Buyers*, *NonBuyers*, and banning Bots classification

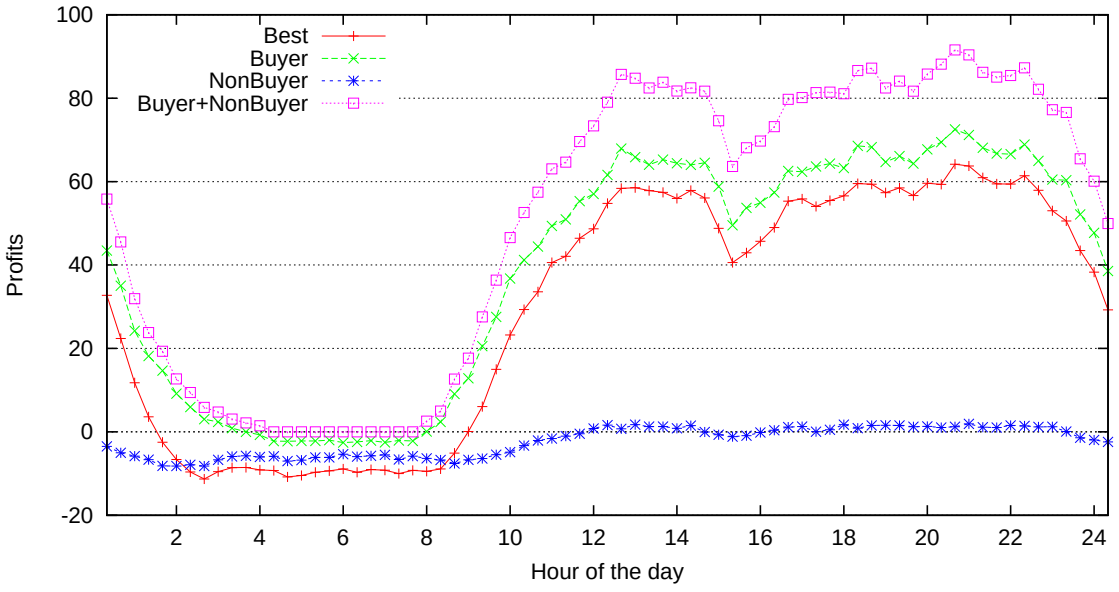


FIGURE 6.15: Profits for *Buyers*, *NonBuyers*, and banning Bots classification

6.8 Discussion of results

This section presents a comparison of the different fixed and dynamic response time strategies as well as a comparison with static servers at different costs. Table 6.2 presents the aggregate percentages of the total possible profits obtained. It was generated by running AUGURES with the last month in the dataset, January 2013. 100% profits under this scenario, will be equivalent to no hosting costs, and all the possible revenue from user sales. Sales values are obtained from the supplied sales dataset.

The *Lower* strategy keeps the lowest possible response of the application at all times —0.4 seconds, since it is the best response time possible, it obtains all sales. However, at this response time, hosting costs corresponds to 73% of all possible profits, lowering profits to 37%. Keeping a policy of maintaining 2 seconds in the infrastructure leave us with only 17% of total profits, as many sales are lost at this response time and inccurs in some server costs. Server use under the different strategies can be seen by going back to Figures 6.7 and 6.11. Keeping 4 seconds for response time, is even worse, yielding negative results. The main reason for this, is that for the prices use for this experiment, selling has higher benefits than server costs, so it always preferable to give a good QoS and promote sales, as normally occurs with real-life Ecommerce retailers.

Next, for contrasting profits with dynamic server provisioning, the use of a fix number of servers was studied. 10 servers was selected as the baseline as it is the number Web servers used by Atrapalo.com. With this number of servers, they are able to serve all their clients —unless there is an important traffic spike. For fixed servers, different cost have been explored: We can see that at 5 USD —the same price as the dynamic strategies— the total profits are only 28% of the total possible revenue. Since fixed number of servers is generally cheaper, both in a Cloud infrastructure as well as in collocated scenarios. If reserved instances had a cost of 3 USD, profits increase up to 41%. In the case of servers being collocated in a traditional datacenter as Atrapalo currently has, the cost of running servers is much lower —without counting the actual server prices and the cost of managing the hardware. For this reason, a lower price of 1 USD was selected, yielding 53% of total profits.

The last 4 strategies from our own proposal, re-evaluate the response time at each control period (T) —20 minutes— in this case. The *Best* strategy outperforms the

Lower static response time strategy by 18% in profits. The main reason for this gain in profits is that it offers a low CR, around 0.7 seconds during the day, and obtains almost all possible sales. This way in average saving 40% the number of servers needed compared to *Lower*. This number of course is dependent on the server capacity model, but our model presented in Section 6.4.1 has been validated in different benchmarks from Atrapalo application using the 2-years dataset. The other reason is that it obtains more profits is because during early-morning time it decides to degrade slightly QoS, giving a higher response time, close to 2 seconds. The CR during the early-morning is very low, most of the traffic comes from automated requests and crawlers.

The next best performing strategy is AUGURES. The AUGURES strategy classifies users into *Buyers* and *NonBuyers* groups, obtaining 7% more profits than the *Best* strategy. By doing such classification, it is able to group most buyers in a cluster and give a better response time, while degrading slightly the response time of the *NonBuyer* group. In this way, saving in servers, as buyers represent a small portion of the total workload. The classifiers selects about 10% of the total user sessions as belonging to the *Buyer* group, thus sets the QoS higher for this cluster.

The effect of session admission control was also studied; Section 6.7.5 presented the classification of unwanted *Bot* traffic. In the case of the presented dataset the prediction technique could eliminate 17% of total requests. As an extension to the AUGURES strategy, the *AUGURES + Bots* strategy, besides classifying users, it also predicts if the session might come from an automated bot. By classifying Bots and discarding such traffic, an extra 5% of profits can be obtained compared to AUGURES, and 12% compared to the *Best* strategy, as less servers are required. Next, the total profits for using multiple buyer prediction policies from Section 6.7.4 are evaluated: by being able to change the target response time, and the size of cluster, an extra 5% of profits could be obtained *e.g.*, at night time using only one cluster, but during the select the most profitable one. Adding *Bot* selection to multiple policies improves profits, but by less than 2% to the multiple policies strategy.

Strategy	Price per hour	Percentage of profits
Lower RT 0.4	5 USD	37.2%
RT 2.0	5 USD	17.9%
RT 4.0	5 USD	-11.6%
Static 10 Servers	5 USD	27.6%
Static 10 Servers	3 USD	40.6%
Static 10 Servers	1 USD	53.2%
Best	5 USD	55.4%
AUGURES	5 USD	62.0%
AUGURES + Bots	5 USD	66.7%
Multiple policies	5 USD	67.3%
Multiple policies + Bots	5 USD	68.6%

TABLE 6.2: Percentage of profits obtained by each policy

6.9 Summary

Improving cost information enables organizations to understand how demand relates to supply, and how cost behaves through consumption. By improving demand, supply, and consumption accounting, the better the planning, budgeting and decision making. We have presented a methodology to provide economical metrics relating costs and revenues to answer the questions of when to scale in number of server and to what number, as well as setting the most convenient QoS at each time of the day. We do so by relating service response times to user satisfaction that leads to sales. As a proof-of-concept, we have presented a prototype implementation, AUGURES, which predicts the expected workload, and the potential profits to be obtained from the workload at low time intervals *e.g.*, 20 minutes, according to the expected conversion rates for that date and time of the day.

In our results a dynamic policy of changing the response time during the day outperforms the baseline policy of maintaining two-second response time as performance target by 52% in profits. Furthermore, if the company had a policy of keeping the lowest possible response time for users, our *Best* strategy would outperform it in profits by 18% by saving 40% in servers, degrading slightly the response time. In the case of user classification, grouping users into different clusters, we show that profits can be improved at least 7% more. The effect of session admission control was also studied, and applied to the classification of unwanted scrapper traffic, which in the case of the presented dataset and prediction technique eliminated 17% of total requests. We also compared having a static infrastructure to a dynamic server infrastructure. At the same server per hour price, the

AUGURES technique outperforms a static infrastructure by 34% in profits. If the static infrastructure or reserved instances had a cost of only one fifth, profits still improve by 11%, with the additional benefit of supporting traffic spikes. Session admission control can improve profits by banning automated crawlers, saving the costs of 20% extra servers from the supplied datasets. The presented technique can enable *profit-aware* resource management as shown in our experiments; allowing the *self-configuration* of IaaS to an optimal number of servers. This way, potentially reducing hosting costs for an incoming workload. While also keeping a high throughput in sales, following high-level policies of business expectations.

Chapter 7

Conclusions

This thesit has presented three different and complementary techniques to contribute in the area of dynamic service provisioning and Web server resource management in general, based on real-life production datasets.

In the first contribution, 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. Using the proposed technique 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; these should be mostly non-buying and crawler sessions, while it admits most buying ones. The second scenario is that in which keeping extra servers running has a quantifiable cost; in this case, one could try to group buying sessions a small number of servers, possibly shutting down those other servers that would produce little or no revenue *e.g.*, for crawler and bot traffic, or to provide differentiated QoS per server. We can conclude that admission control, and resource management in general, is a promising application field for automatically learned user models.

In the second contribution, we have argued that the effect of response time degradation can be hidden by the fact that peak load times can coincide with high conversion rates, *i.e.*, when higher fraction of visitors have intention to purchase. To overcome this effect we have introduced a novel methodology for studying what is the volume of sales lost in an online retailer due to performance degradation without modifying its application. We use machine learning techniques to predict the expected sales volume over time and look for deviations over the expected values during overload periods that may introduce performance degradation. Using such technique, we can quantify the impact of response time in the business activities of an online service without modifying production system. From the obtained results we are able to identify inflection points where sales start to drop for different applications when response time is high. For the application in the supplied dataset, there is a *tolerating* response time threshold from 3 to 11 seconds, where some sales are lost, and a *frustration* threshold at 11 seconds, where each increase in 1 second interval increases total sale loss by 6%. The presented methodology enables online retailers to determine inflection points where sales start to be affected by the current application's *response time*. Where resulting values can be applied on autonomic resource managers to optimize the number of servers and reduce infrastructure costs in cloud computing environments. Most importantly, optimizations should not be made to accommodate all load, but to provide the best QoS when conversion rates are higher, generally at peak loads. As an additional contribution, results from this study had led the presented OTA to make important changes in their infrastructure to avoid high response times, especially at peak times with positive results.

As third and final contribution we have presented a resource management policy applied on custom economical metrics relating costs and revenues to answer the questions of when to scale and to what number. We do so by leveraging the user revenue models from the first contribution and relation of service times to sales from the second contribution, while adding a server capacity model and a prototype to perform the experimentation. In our results AUGURES outperforms the baseline policy of maintaining two-second response time as performance target by 34% in profits. Furthermore, if the company had a policy of keeping the lowest possible response time for users, our *Best* strategy would outperform it by requiring 40% less servers. At the same server per hour price, the AUGURES technique outperforms a static infrastructure of keeping a fix number of servers by 34% in profits. If the static infrastructure or reserved instances had a cost

of only one fifth, profits will improve by 11%. The effect of session admission control was also studied, and applied to the classification of unwanted scraper bot traffic, which in the case of the presented dataset and prediction technique eliminated 17% of total requests. Session admission control can improve profits, as it saves the costs of 20% extra servers in our experiments.

This thesit has presented how by understanding user behahavior —the demand, the relation of Quality-of-Service to sales —resource consumption, and scaling Cloud infrastructures —the supply, an online business can improve planning, decision making, and improve profits by keeping the most optimal infrastructure. In particular scalability decisions can be automated, freeing system administrators from operational details, while the infrastructure runs at top performance following high-level policies of business expectations.

7.1 Future work

As future work we plan to test the presented approach in different number of Cloud scenarios and workloads. As first extension, we will like to evaluate profits optimizations having into consideration different server sizes (vertical scalability), this way, when scaling, deciding to consolidate a group of smaller servers into a single larger server. This way, optimizations can be made to more parameters *i.e.*, revenue, server costs, improved QoS or savings in energy. As second extension, market based instances *i.e.*, *spot instances* can also be considered for profit calculations, reducing costs in servers at particular times of the day when the price offering is lower. A similar, but opposite approach can be explored for Cloud reserved server instances, which have a lower cost than on-demand instances; having the penalty of reserving them for a year in current offerings. Time constrains such as having an on-demand server for a full hour —rather than fractions of hours— need to be explored in more depth, as current Cloud offerings charge by the hour. For large or globally distributed Web sites, the effect of multi-datacenters and migration among them should be explored, as in this work we have centered the experiments in the market of the studied site and applications.

We have noticed that mobile access to the site has increased from being practically inexistent at the beginning of the datasets, to representing more than 10% of the total

traffic. It is expetected that mobile users will demand a diffent experience in terms of UX and site peformance, as current studies indicate. An interesting extension of this work would be to differentiate users by type *i.e.*, device type, connectivity, work or leasure, and demographics.

We also plan to extend the prototype and experimentation in a per-application base, as our findings show in Chapter 5 that applications have differentiated resource requirements and QoS thresholds. Where we have identified particularities of certain applications *i.e.*, the restaurants application has very defined purchasing hours —just before lunch or dinner— or the *event-ticket* application, that is less affected by response times as it can have the exclusivity of the event.

We have briefly explored different session valuation techniques, however we have centered the experiments in this thesis in user sales. Our prototype should be able to adapt to other metrics, specifically for sites that do not sell products. Value can be set to different actions that the user performs in a site, *i.e.*, submitting content to a social network, in that case rewarding and giving higher priority for this type of user sessions.

Appendix A

Thesis scientific production

A.1 Main publications

- Nicolas Poggi, Toni Moreno, Josep Lluis Berral, Ricard Gavaldà, and Jordi Torres. **Web Customer Modeling for Automated Session Prioritization on High Traffic Sites**. *Proceedings of the 11th International Conference on User Modeling*, pages 450–454, June 25-29, 2007

- Nicolas Poggi, Toni Moreno, Josep Lluis Berral, Ricard Gavaldà, and Jordi Torres. **Automatic Detection and Banning of Content Stealing Bots for E-commerce**. *NIPS 2007 Workshop on Machine Learning in Adversarial Environments for Computer Security*, December 8, 2007

- Nicolas Poggi, Toni Moreno, Josep Lluis Berral, Ricard Gavaldà, and Jordi Torres. **Self-Adaptive Utility-Based Web Session Management**. *Computer Networks Journal*, 53(10):1712–1721, 2009. ISSN 1389-1286

- Nicolas Poggi, David Carrera, Ricard Gavaldà, Jordi Torres, and Eduard Ayguadé. **Characterization of Workload and Resource Consumption for an Online Travel and Booking Site**. *IISWC - 2010 IEEE International Symposium on Workload Characterization*, December 2 -4, 2010

- Nicolas Poggi, David Carrera, Ricard Gavaldà, and Eduard Ayguadé. **Non-intrusive Estimation of QoS Degradation Impact on E-Commerce User**

Satisfaction. In *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, pages 179–186, 2011. doi: 10.1109/NCA.2011.31

- Nicolas Poggi, David Carrera, Ricard Gavaldà, Eduard Ayguadé, and Jordi Torres. **A methodology for the evaluation of high response time on E-commerce users and sales**. *Information Systems Frontiers*, pages 1–19, 2012. ISSN 1387-3326

- Nicolas Poggi, David Carrera, Eduard Ayguadé, and Jordi Torres. **Profit Oriented Fine-Grain Web Server Management**. *Technical Report: UPC-DAC-RR-2013-60*, November, 2013

- Nicolas Poggi, David Carrera, Eduard Ayguadé, and Jordi Torres. **Profit-Aware Cloud Resource Provisioner for Ecommerce**. *Sent for consideration*, 2013

A.2 Extensions and collaborations

- Toni Moreno, Nicolas Poggi, Josep Lluis Berral, Ricard Gavaldà, and Jordi Torres. **Policy-based autonomous bidding for overload management in eCommerce websites**. *Group Decision and Negotiation Meeting - Group Decision and Negotiation, INFORMS*, pages 162–166, 2007

- J. Torres, D. Carrera, K. Hogan, R. Gavaldà, V. Beltran, and N. Poggi. **Reducing wasted resources to help achieve green data centers**. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, 2008. doi: 10.1109/IPDPS.2008.4536219

- J. Torres, D. Carrera, V. Beltran, N. Poggi, K. Hogan, J.L. Berral, R. Gavaldà, E. Ayguadé, T. Moreno, and J. Guitart. **Tailoring Resources: The Energy Efficient Consolidation Strategy Goes Beyond Virtualization**. In *Autonomic Computing, 2008. ICAC '08. International Conference on*, pages 197–198, 2008. doi: 10.1109/ICAC.2008.11

- Nicolas Poggi, Vinod Muthusamy, David Carrera, and Rania Khalaf. **Business Process Mining from E-Commerce Web Logs**. In *Business Process Management*, volume 8094 of *Lecture Notes in Computer Science*, pages 65–80. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-40175-6

List of Figures

List of Tables

Bibliography

[1] E-commerce web site performance today. an updated look at consumer reaction to a poor online shopping experience. *Forrester Consulting*, August 17, 2009.

[2] Oecd report on broadband penetration. available at: http://www.oecd.org/sti/ict/broadband. *OECD*, Dec, 2009.

[3] International Telecommunication Union. Global ict developments. http://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx, February 2013.

[4] Carroll Rheem. Consumer response to travel site performance. *A PhoCusWright WHITEPAPER*, April 2010.

[5] Marissa Mayer. In search of a better, faster, stronger web. *Velocity 2009, Web Performance Operations Conference*, June 2009.

[6] David Artz. The secret weapons of the aol optimization team. *Velocity 2009, Web Performance Operations Conference*, June 2009.

[7] Michele Mazzucco. Towards autonomic service provisioning systems. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:273–282, 2010. doi: http://doi.ieeecomputersociety.org/10.1109/CCGRID.2010.125.

[8] Using site speed in web search ranking. webpage: http://googlewebmastercentral.blogspot.com/2010/04/using-site-speed-in-web-search-ranking.html. *Google*, April 2010.

[9] Internet World Stats. World internet users and population stats. http://www.internetworldstats.com/stats.htm, June 2012.

[10] Nicolas Poggi, David Carrera, Ricard Gavaldà, Jordi Torres, and Eduard Ayguadé. **Characterization of Workload and Resource Consumption for an Online**

Travel and Booking Site. *IISWC - 2010 IEEE International Symposium on Workload Characterization*, December 2 -4, 2010.

[11] Daniel A. Menascé, Daniel Barbará, and Ronald Dodge. Preserving qos of e-commerce sites through self-tuning: a performance model approach. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, EC '01, pages 224–234, 2001. ISBN 1-58113-387-1.

[12] J.O. Kephart and R. Das. Achieving self-management via utility functions. *Internet Computing, IEEE*, 11(1):40–48, 2007. ISSN 1089-7801.

[13] Sean Power. Metrics 101: What to measure on your website. *Velocity 2010, Web Performance Operations Conference*, June 2010.

[14] Symantec. State of the data center global data. http://www.symantec.com/content/en/us/about/media/pdfs/Symantec_DataCenter10_Report_Global.pdf, January 2012.

[15] Ahmed Ataullah. In *Web Information Systems Engineering - WISE 2007*, volume 4831 of *Lecture Notes in Computer Science*, pages 533–542. 2007. ISBN 978-3-540-76992-7.

[16] Ronny Kohavi. Mining e-commerce data: The good, the bad, and the ugly. In *Advances in Knowledge Discovery and Data Mining*, Lecture Notes in Computer Science, pages 2–2. Springer Berlin Heidelberg, 2001. ISBN 978-3-540-41910-5.

[17] Nicolas Poggi, David Carrera, Ricard Gavaldà, Eduard Ayguadé, and Jordi Torres. **A methodology for the evaluation of high response time on E-commerce users and sales**. *Information Systems Frontiers*, pages 1–19, 2012. ISSN 1387-3326.

[18] Gary Cokins. Evaluating the costing journey: A costing levels continuum maturity model. *Professional Accountants in Business Committee, International Federation of Accountants, IFAC*, July, 2009.

[19] Young Choon Lee, Chen Wang, A.Y. Zomaya, and Bing Bing Zhou. Profit-driven service request scheduling in clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 15–24, 2010.

[20] J. Torres, D. Carrera, V. Beltran, N. Poggi, K. Hogan, J.L. Berral, R. Gavaldà, E. Ayguadé, T. Moreno, and J. Guitart. **Tailoring Resources: The Energy Efficient Consolidation Strategy Goes Beyond Virtualization**. In *Autonomic Computing, 2008. ICAC '08. International Conference on*, pages 197–198, 2008. doi: 10.1109/ICAC.2008.11.

[21] Mika Majakorpi. Theory and practice of rapid elasticity in cloud applications, April 2013.

[22] Jordi Guitart, David Carrera, Vincent Beltran, Jordi Torres, and Eduard Ayguadé. Session-based adaptive overload control for secure dynamic web applications. *Parallel Processing, International Conference on*, 0:341–349, 2005. ISSN 0190-3918. doi: http://doi.ieeecomputersociety.org/10.1109/ICPP.2005.72.

[23] Ludmila Cherkasova and Peter Phaal. Session-based admission control: A mechanism for peak load management of commercial web sites. *IEEE Transactions on Computers*, 51:669–685, 2002. ISSN 0018-9340. doi: http://doi.ieeecomputersociety.org/10.1109/TC.2002.1009151.

[24] Toni Moreno, Nicolas Poggi, Josep Lluis Berral, Ricard Gavaldà, and Jordi Torres. **Policy-based autonomous bidding for overload management in eCommerce websites**. *Group Decision and Negotiation Meeting - Group Decision and Negotiation, INFORMS*, pages 162–166, 2007.

[25] J. Torres, D. Carrera, K. Hogan, R. Gavaldà, V. Beltran, and N. Poggi. **Reducing wasted resources to help achieve green data centers**. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, 2008. doi: 10.1109/IPDPS.2008.4536219.

[26] Nicolas Poggi, Vinod Muthusamy, David Carrera, and Rania Khalaf. **Business Process Mining from E-Commerce Web Logs**. In *Business Process Management*, volume 8094 of *Lecture Notes in Computer Science*, pages 65–80. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-40175-6.

[27] Nicolas Poggi, Toni Moreno, Josep Lluis Berral, Ricard Gavaldà, and Jordi Torres. **Web Customer Modeling for Automated Session Prioritization on High Traffic Sites**. *Proceedings of the 11th International Conference on User Modeling*, pages 450–454, June 25-29, 2007.

[28] Nicolas Poggi, Toni Moreno, Josep Lluis Berral, Ricard Gavaldà, and Jordi Torres. **Automatic Detection and Banning of Content Stealing Bots for E-commerce**. *NIPS 2007 Workshop on Machine Learning in Adversarial Environments for Computer Security*, December 8, 2007.

[29] Nicolas Poggi, Toni Moreno, Josep Lluis Berral, Ricard Gavaldà, and Jordi Torres. **Self-Adaptive Utility-Based Web Session Management**. *Computer Networks Journal*, 53(10):1712–1721, 2009. ISSN 1389-1286.

[30] Robert B. Miller. Response time in man-computer conversational transactions. In *AFIPS '68 (Fall, part I): Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277, New York, NY, USA, 1968. ACM. doi: http://doi.acm.org/10.1145/1476589.1476628.

[31] Nicolas Poggi, David Carrera, Ricard Gavaldà, and Eduard Ayguadé. **Non-intrusive Estimation of QoS Degradation Impact on E-Commerce User Satisfaction**. In *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, pages 179–186, 2011. doi: 10.1109/NCA.2011.31.

[32] Steven Rose Geoffrey Hoffman. Auto scaling your website with amazon web services (aws) - part 2. http://www.cardinalpath.com/autoscaling-your-website-with-amazon-web-services-part-2/, May 2012.

[33] Justin Becker Greg Orzell. Auto scaling in the amazon cloud. http://techblog.netflix.com/2012/01/auto-scaling-in-amazon-cloud.html, January 2012.

[34] Nicolas Poggi, David Carrera, Eduard Ayguadé, and Jordi Torres. **Profit Oriented Fine-Grain Web Server Management**. *Technical Report: UPC-DAC-RR-2013-60*, November, 2013.

[35] Nicolas Poggi, David Carrera, Eduard Ayguadé, and Jordi Torres. **Profit-Aware Cloud Resource Provisioner for Ecommerce**. *Sent for consideration*, 2013.

[36] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003. ISSN 0018-9162.

[37] Jordi Guitart, David Carrera, Vicenç Beltran, Jordi Torres, and Eduard Ayguadé. Designing an overload control strategy for secure e-commerce applications. *Comput. Netw.*, 51(15):4492–4510, October 2007. ISSN 1389-1286.

[38] Yaya Wei, Chuang Lin, Fengyuan Ren, E. Dutkiewicz, and R. Raad. Session based differentiated quality of service admission control for web servers. In *Computer Networks and Mobile Computing, 2003. ICCNMC 2003. 2003 International Conference on*, pages 112–116, 2003.

[39] Tarek Abdelzaher, Kang G. Shin, and Nina Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13:2002, 2001.

[40] K. Sharma, G. Shrivastava, and V. Kumar. Web mining: Today and tomorrow. In *ICECT*, volume 1, 2011. doi: 10.1109/ICECTECH.2011.5941631.

[41] R. Bhushan and R. Nath. Automatic recommendation of web pages for online users using web usage mining. In *ICCS*, 2012. doi: 10.1109/ICCS.2012.17.

[42] Myra Spiliopoulou, Carsten Pohle, and LukasC. Faulstich. Improving the effectiveness of a web site with web usage mining. In *Web Usage Analysis and User Profiling*. 2000. ISBN 978-3-540-67818-2. doi: 10.1007/3-540-44934-5_9.

[43] Dario Bonino, Fulvio Corno, and Giovanni Squillero. A real-time evolutionary algorithm for web prediction. In *Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence*, WI '03, pages 139–, Washington, DC, USA, 2003.

[44] Alexandros Nanopoulos, Dimitris Katsaros, and Yannis Manolopoulos. Effective prediction of web-user accesses: A data mining approach, 2001.

[45] Bin Lan, Stephane Bressan, Beng Chin Ooi, and Kian lee Tan. Rule-assisted prefetching in web-server caching. In *Proc. ACM Intl Conf. Information and Knowledge Management (ACM CIKM 00*, pages 504–511, 2000.

[46] Prefetching and Qiang Yang. Mining web logs for prediction models in www caching, 2001.

[47] Ramesh R. Sarukkai. Link prediction and path analysis using markov chains. *Comput. Netw.*, 33(1-6), June 2000.

[48] Mukund Deshpande and George Karypis. Selective markov models for predicting web page accesses. *ACM Trans. Internet Technol.*, 4(2), May 2004.

[49] Michael Rabinovich and Oliver Spatschek. *Web Caching and Replication*. Boston, USA, 2002. ISBN 0-201-61570-3.

[50] Daniel A Menascé, Virgilio AF Almeida, Rodrigo Fonseca, and Marco A Mendes. A methodology for workload characterization of e-commerce sites. In *ACM EC*, 1999.

[51] Alexander Totok and Vijay Karamcheti. Rdrp: Reward-driven request prioritization for e-commerce web sites. *Electron. Commer. Rec. Appl.*, 9(6):549–561, November 2010. ISSN 1567-4223.

[52] Lakhwinder Kumar, Hardeep Singh, and Ramandeep Kaur. Web analytics and metrics: a survey. In *ACM ICACCI*, 2012. ISBN 978-1-4503-1196-0. doi: 10.1145/2345396.2345552.

[53] Szabolcs Rozsnyai, Aleksander Slominski, and Geetika T. Lakshmanan. Discovering event correlation rules for semi-structured business processes. In *ACM DEBS*, 2011. ISBN 978-1-4503-0423-8. doi: 10.1145/2002259.2002272.

[54] W. M. P. van der Aalst et al. Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.*, 47(2), November 2003. ISSN 0169-023X. doi: 10.1016/S0169-023X(03)00066-1.

[55] Wil M. P. van der Aalst, Boudewijn F. van Dongen, Christian W. Gunther, Anne Rozinat, Eric Verbeek, and Ton Weijters. ProM: The process mining toolkit. In *BPM (Demos)*, 2009.

[56] Anne Rozinat and Wil M. P. van der Aalst. Decision mining in ProM. In *Business Process Management*, 2006.

[57] Wil M. P. van der Aalst, M. H. Schonenberg, and Minseok Song. Time prediction based on process mining. *Inf. Syst.*, 36(2):450–475, 2011.

[58] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

[59] Michael L. Littman, Thu Nguyen, and Haym Hirsh. Cost-sensitive fault remediation for autonomic computing. In *In Proc. of IJCAI Workshop on AI and Autonomic Computing*, 2003.

[60] C. Kenyon. G. Cheliotis. 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.

[61] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[62] Virgílio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing reference locality in the www. In *DIS '96: Proceedings of the fourth international conference on on Parallel and distributed information systems*, pages 92–107, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7475-X.

[63] Mark Levene, José Borges, and George Loizou. Zipf's law for web surfers. *Knowl. Inf. Syst.*, 3(1):120–129, 2001. ISSN 0219-1377. doi: http://dx.doi.org/10.1007/PL00011657.

[64] Daniel Menascé, Virgílio Almeida, Rudolf Riedi, Flávia Ribeiro, Rodrigo Fonseca, and Wagner Meira, Jr. In search of invariants for e-business workloads. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, pages 56–65, New York, NY, USA, 2000. ACM. ISBN 1-58113-272-7. doi: http://doi.acm.org/10.1145/352871.352878.

[65] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgílio Almeida. Characterizing user behavior in online social networks. In *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 49–62, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-771-4. doi: http://doi.acm.org/10.1145/1644893.1644900.

[66] Marcos Andre Goncalves, Jussara M. Almeida, Luiz G.P. dos Santos, Alberto H.F. Laender, and Virgilio Almeida. On popularity in the blogosphere. *IEEE Internet Computing*, 14:42–49, 2010. ISSN 1089-7801. doi: http://doi.ieeecomputersociety.org/10.1109/MIC.2010.73.

[67] C. Stewart, M. Leventi, and Kai Shen. Empirical examination of a collaborative web application. *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, pages 90–96, Sept. 2008. doi: 10.1109/IISWC.2008.4636094.

[68] Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. *SIGMETRICS Perform. Eval. Rev.*, 26(1):151–160, 1998. ISSN 0163-5999. doi: http://doi.acm.org/10.1145/277858.277897.

[69] Emmanuel Cecchet, Julie Marguerite, and Willy Zwaenepoel. Performance and scalability of ejb applications. *SIGPLAN Not.*, 37(11):246–261, 2002. ISSN 0362-1340. doi: http://doi.acm.org/10.1145/583854.582443.

[70] P. Nagpurkar, W. Horn, U. Gopalakrishnan, N. Dubey, J. Jann, and P. Pattnaik. Workload characterization of selected jee-based web 2.0 applications. *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, pages 109–118, Sept. 2008. doi: 10.1109/IISWC.2008.4636096.

[71] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*, pages 171–180, Sept. 2007. doi: 10.1109/IISWC.2007.4362193.

[72] R. Gusella. Characterizing the variability of arrival processes with indexes of dispersion. *Selected Areas in Communications, IEEE Journal on*, 9(2):203 –211, feb 1991. ISSN 0733-8716. doi: 10.1109/49.68448.

[73] Giuliano Casale, Ningfang Mi, Ludmila Cherkasova, and Evgenia Smirni. How to parameterize models with bursty workloads. *SIGMETRICS Perform. Eval. Rev.*, 36(2):38–44, 2008. ISSN 0163-5999. doi: http://doi.acm.org/10.1145/1453175.1453182.

[74] Ningfang Mi, Giuliano Casale, Ludmila Cherkasova, and Evgenia Smirni. Burstiness in multi-tier applications: symptoms, causes, and new models. In *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 265–286, New York, NY, USA, 2008. Springer-Verlag New York, Inc. ISBN 3-540-89855-7.

[75] Ningfang Mi, Giuliano Casale, Ludmila Cherkasova, and Evgenia Smirni. Injecting realistic burstiness to a traditional client-server benchmark. In *ICAC '09: Proceedings of the 6th international conference on Autonomic computing*, pages 149–158, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-564-2. doi: http://doi.acm.org/10.1145/1555228.1555267.

[76] Christopher Stewart, Terence Kelly, and Alex Zhang. Exploiting nonstationarity for performance prediction. *SIGOPS Oper. Syst. Rev.*, 41(3):31–44, 2007. ISSN 0163-5980. doi: http://doi.acm.org/10.1145/1272998.1273002.

[77] J. P. Patwardhan, A. R. Lebeck, and D. J. Sorin. Communication breakdown: analyzing cpu usage in commercial web workloads. In *ISPASS '04: Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 12–19, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7803-8385-0.

[78] M. Ye and L. Cheng. System-performance modeling for massively multiplayer online role-playing games. *IBM Syst. J.*, 45(1):45–58, 2006. ISSN 0018-8670. doi: http://dx.doi.org/10.1147/sj.451.0045.

[79] Dennis F. Galletta, Raymond Henry, Scott Mccoy, and Peter Polak. Web site delays: How tolerant are users. *Journal of the Association for Information Systems*, 5:1–28, 2004.

[80] P. J. Sevcik. Understanding how users view application performance. *Business Communications Review*, 32(7):8–9, 2002.

[81] Jakob Nielsen. Usability engineering at a discount. In *Proceedings of the third international conference on human-computer interaction on Designing and using human-computer interfaces and knowledge based systems (2nd ed.)*, pages 394–401, New York, NY, USA, 1989. Elsevier. ISBN 0-444-88078-X.

[82] Apdex Alliance. Apdex application performance index. http://www.apdex.org, August 2013.

[83] Li Zhang, Zhen Liu, Anton Riabov, Monty Schulman, Cathy Xia, and Fan Zhang. A comprehensive toolset for workload characterization, performance modeling, and online control. In *Computer Performance Evaluation. Modelling Techniques*

and Tools, volume 2794 of *Lecture Notes in Computer Science*, pages 63–77. 2003. ISBN 978-3-540-40814-7.

[84] Mauro Andreolini, Sara Casolari, and Michele Colajanni. Models and framework for supporting runtime decisions in web-based systems. *ACM Trans. Web*, 2(3), July 2008. ISSN 1559-1131.

[85] Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. Self-adaptive workload classification and forecasting for proactive resource provisioning. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 187–198, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1636-1. doi: 10.1145/2479871.2479899. URL http://doi.acm.org/10.1145/2479871.2479899.

[86] Josep L. Berral, Nicolas Poggi, Javier Alonso, Ricard Gavaldà, Jordi Torres, and Manish Parashar. Adaptive distributed mechanism against flooding network attacks based on machine learning. In *Proceedings of the 1st ACM workshop on Workshop on AISec*, AISec '08, pages 43–50, 2008. ISBN 978-1-60558-291-7.

[87] Li Zhang and Danilo Ardagna. Sla based profit optimization in autonomic computing systems. In *Proceedings of the 2nd international conference on Service oriented computing*, ICSOC '04, pages 173–182, 2004. ISBN 1-58113-871-7.

[88] F.I. Popovici and J. Wilkes. Profitable services in an uncertain world. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pages 36–36, 2005.

[89] Bianca Schroeder, Adam Wierman, and Mor Harchol-Balter. Open versus closed: a cautionary tale. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*, NSDI'06, pages 18–18, 2006.

[90] Junliang Chen, Chen Wang, Bing Bing Zhou, Lei Sun, Young Choon Lee, and Albert Y. Zomaya. Tradeoffs between profit and customer satisfaction for service provisioning in the cloud. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 229–238, 2011. ISBN 978-1-4503-0552-5.

[91] Utility-based placement of dynamic web applications with fairness goals. In *Network Operations and Management Symposium, NOMS08*, pages 9–16. IEEE, 2008.

[92] Shuo Liu, Shaolei Ren, Gang Quan, Ming Zhao, and Shangping Ren. Profit aware load balancing for distributed cloud data centers. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 611–622, 2013.

[93] Xi Chen, Haopeng Chen, Qing Zheng, Wenting Wang, and Guodong Liu. Characterizing web application performance for maximizing service providers' profits in clouds. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 191–198, 2011.

[94] Yu-Ju Hong, Jiachen Xue, and Mithuna Thottethodi. Dynamic server provisioning to minimize cost in an iaas cloud. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '11, pages 147–148, 2011. ISBN 978-1-4503-0814-4.

[95] Nielsen. Trends in online shopping, a Nielsen Consumer report. Technical report, Nielsen, Feb. 2008.

[96] Wikipedia. Web scraping. http://en.wikipedia.org/wiki/Web_scraping, June 2013.

[97] M. Hepp, D. Bachlechner, and K. Siorpaes. Harvesting wiki consensus-using wikipedia entries as ontology elements. *First Workshop on Semantic Wikis*, 2008.

[98] Jana Koehler. Business process modeling.

[99] Daniel F. Garcia, Javier Garcia, Joaquin Entrialgo, Manuel Garcia, Pablo Valledor, Rodrigo Garcia, and Antonio M. Campos. A qos control mechanism to provide service differentiation and overload protection to internet scalable servers. *IEEE Transactions on Services Computing*, 2:3–16, 2009. ISSN 1939-1374. doi: http://doi.ieeecomputersociety.org/10.1109/TSC.2009.3.

[100] John M. Ewing and Daniel A. Menascé. Business-oriented autonomic load balancing for multitiered web sites. In *MASCOTS*, pages 1–10, 2009.

[101] M. Al-Ghamdi, A.P. Chester, and S.A. Jarvis. Predictive and dynamic resource allocation for enterprise applications. *Computer and Information Technology, International Conference on*, 0:2776–2783, 2010. doi: http://doi.ieeecomputersociety.org/10.1109/CIT.2010.463.

[102] Neil J. Gunther. Performance and scalability models for a hypergrowth e-commerce web site. In *Performance Engineering, State of the Art and Current Trends*, pages 267–282, London, UK, 2001. Springer-Verlag. ISBN 3-540-42145-9.

[103] Jason Read. What is an ecu? cpu benchmarking in the cloud. http://blog.cloudharmony.com/2010/05/what-is-ecu-cpu-benchmarking-in-cloud.html, May 2010.