RDRP: Reward-Driven Request Prioritization for e-Commerce Web Sites

Alexander Totok[*,a], Vijay Karamcheti[b]

[a]*Google Inc., 76 9th Ave, 6th Floor, New York, NY 10011, USA*
[b]*Department of Computer Science, Courant Institute of Mathematical Sciences,*
New York University, 715 Broadway, 7th Floor, New York, NY 10003, USA

Abstract

Meeting client Quality-of-Service (QoS) expectations proves to be a difficult task for the providers of e-Commerce services, especially when web servers experience overload conditions, which cause increased response times and request rejections, leading to user frustration, lowered usage of the service and reduced revenues. In this paper, we propose a server-side request scheduling mechanism that addresses these problems. Our *Reward-Driven Request Prioritization* (RDRP) algorithm gives higher execution priority to client web sessions that are likely to bring more service *profit* (or any other application-specific *reward*). The method works by predicting future session structure by comparing its requests seen so far with aggregated information about recent client behavior, and using these predictions to preferentially allocate web server resources. Our experiments using the TPC-W benchmark application with an implementation of the RDRP techniques in the JBoss web application server show that RDRP can significantly boost profit attained by the service, while providing better QoS to clients that bring more profit.

Key words: e-Commerce services, profit maximization, performance, quality of service, request scheduling, admission control, Bayesian inference analysis

1. Introduction

In recent decade, the role of the Internet has undergone a transition from simply being a data repository to one providing access to a variety of sophisticated Internet services, such as e-mail, shopping, social networking, and entertainment. Various e-Commerce services (e.g., online banking, shopping) constitute a significant portion of the services offered on the Internet. Typical interaction of users with such services is organized into *sessions*, a sequence of related requests, which together achieve a higher level user goal. An example of such interaction is an on-line shopping scenario for a retail e-Commerce web site, which involves multiple requests that search for particular products, retrieve information about a specific item, add it to the shopping cart, initiate the check-out process, and finally commit the order. The *success of the whole session* now becomes the ultimate QoS goal, which contrasts with the *per-request success* performance metrics of the early Internet.

---

[*]Corresponding author. Tel.: +1 646 678 4321; fax: +1 212 995 4123.
Email addresses: `totok@google.com` (Alexander Totok), `vijayk@cs.nyu.edu` (Vijay Karamcheti)

Providers of e-Commerce services frequently have to deal with *service overload* conditions. In such situations, clients see increased response times and their requests (and the containing sessions) may get rejected, which leads to user frustration, and as a consequence, to lowered usage of the service and reduced service revenues (Barnes and Mookerjee, 2009). Recent studies showed that 33% of shoppers on a slow-loading e-Commerce web site abandoned the site entirely, while 75% of visitors would never shop on that site again (Moskalyuk, 2006).

Numerous server-side performance management techniques have been proposed to deal with server overload situations. For example, Session-based Admission Control (SBAC) (Cherkasova and Phaal, 2002) admits only as many sessions as can be served by the service. More complex service differentiation mechanisms have also been used to provide stable QoS guarantees (e.g., request throughput, response times) to different client groups, based on prenegotiated Service-Level Agreements (SLAs). Common to such schemes is the consideration that QoS received by a client is determined upfront by his association with a client group or by his service membership status. However, most of these schemes fall short of delivering the best performance in situations where it makes sense to differentiate among clients based on the (*dynamic*) activities these clients perform in a session, rather than on their (*static*) identity, in order to boost service revenues, or for other application-specific goals. Let's consider the following two examples.

- In the online shopping scenario introduced earlier, the service provider might be interested in giving a higher execution priority to the sessions that have placed something in the shopping cart (potential *buyer* sessions), as compared to the sessions that just *browse* product catalogs, making sure that clients that buy something (and so – bring profit to the service) receive better QoS.

- For a service, some of whose web pages contain third party-sponsored advertisements, the service provider's profits may increase with more visits to these pages, as it may increase the chances of the clients following the advertisement links. Consequently, the service provider may wish to provide better QoS to the sessions that visit web pages with advertisements more often.

These examples are unified by the idea that the service may benefit from providing better QoS to sessions, that bring more *profit* (give more *reward*), where the notion of profit or reward is defined in an application-specific fashion. What is important is that the information about the client's possible usage of a service (and its associated contribution to service reward) is not encoded in any static profile, so application-logic-independent SLA-based service differentiation approaches are not as beneficial here.

Instead, to be able to provide better QoS to the sessions that bring more reward, the service provider now needs to predict the behavior of a client. If a client has used the service before and his identity can be determined (e.g., using cookies), than decisions on QoS provided to this client can be based on the history of his service usage (e.g., history of previous purchases). However, the success of this per-client *history-based approach*, is, not unexpectedly, highly dependent on the correlation between the past and the future behavior of a client, and may not work well if such a correlation is absent or weak.

Instead of focusing on individual client behavior, we advocate the approach of predicting a session's activities by associating it with aggregated client behavior or broader service usage patterns,

obtained for example through online request profiling. Specifically, we propose *Reward-Driven Request Prioritization* (RDRP) mechanisms that try to maximize reward attained by the service, by dynamically assigning higher execution priority values to the requests whose sessions are likely to bring more reward. Our methods compare the sequence of a session's requests seen so far with aggregated information about client behaviors, and use a Bayesian inference analysis to statistically predict the future structure of a session, and so – the reward the session will bring and the execution cost it will incur. The predicted reward and execution cost values are used to compute each request's priority, which is used in scheduling "bottleneck" server resources, such as server threads and database connections, to incoming client requests.

We have implemented our proposed methods as a set of middleware mechanisms, which are seamlessly and modularly integrated in the open-source Java web application server JBoss (JBoss, 2010). A *Request Profiling* module performs automatic real-time monitoring of client requests to extract parameters of service usage and to maintain the histories of session requests. It also performs fine-grained request profiling to identify execution times for different service request types. The *RDRP module* uses the information gathered by the Request Profiling module to compute and assign request priorities, that in turn influences queueing behavior for various application server resources.

We evaluate our approach on the TPC-W benchmark application (TPC-W, 2005), emulating an e-Commerce web site selling books, and compare it with both the session-based admission control and per-client history-based approaches. Our experiments show that RDRP techniques yield benefits in both underload and overload situations, for both smooth and bursty client behavior. In underload situations, the proposed mechanisms give better response times for the clients that bring more profit to the service, thus helping to secure their satisfaction and future return to the web site. Note that it is often the case that the bulk of a service customers are returning clients, so providing good QoS to long-time customers is a key factor in service success (VanBoskirk et al., 2001; Barnes and Mookerjee, 2009). In overload situations, when some of the requests get rejected, the mechanisms ensure that sessions that bring more profit are more likely to complete successfully and that the aggregate profit attained by the service increases compared to other solutions. Additionally, we show that the history-based approach matches the performance of our RDRP mechanisms on the amount of profit gained and response times only if the correlation between the clients' past and future behavior is 75% or greater, and 50% or greater, respectively.

The rest of this paper is organized as follows. Section 2 presents models and assumptions used throughout the paper. Section 3 describes the reward-driven request prioritization techniques. Section 4 presents our testing methodology and experimental results. In Section 5 we discuss related work, and we conclude in Section 6.

2. Models and Assumptions

2.1. Web application server architecture

We work with e-Commerce services implemented on top of modern middleware platforms, such as the Java EE component framework (Java EE, 2010). Such services are usually built as complex (and often distributed) software systems, consisting of several logical and physical tiers (e.g., web tier, application tier, and database tier) and accessing multiple backend data sources. We present our request prioritization algorithms in a centralized setting however, to focus on the
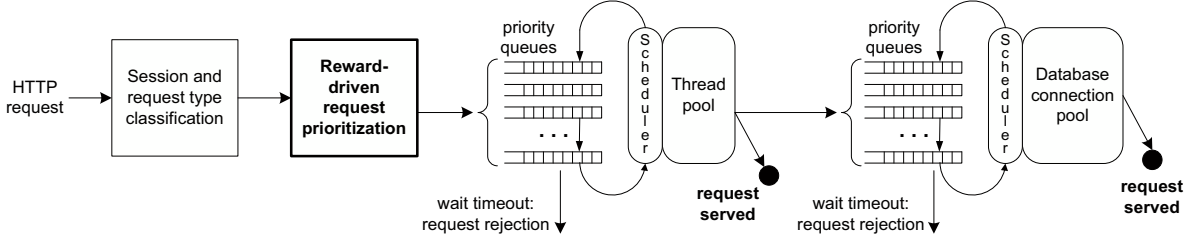
Figure 1: The model of web application server architecture.

benefits of the proposed request prioritization techniques. We expect that the methods will show their utility in a distributed setting as well, where they can be independently applied at every system resource contention point that sees concurrent requests competing for server resources.

Our work uses middleware-level mechanisms for server performance optimization, specifically control over request scheduling policies. We adopt this approach, because the middleware itself often does not have control over the low-level OS resources (e.g., CPU and memory), and uses higher-level mechanisms, such as request scheduling, component pool management, transaction demarcation, etc., to improve server performance. It is often the case that middleware server performance is limited by several "bottleneck" resources, that are *held exclusively* by a request for the whole duration or some significant portion of it (as opposed to low-level *shared* OS resources), such as server threads, or database (DB) connections. The default allocation policy of these resources to requests is FIFO. In the absence of application errors, failing to obtain such a resource is the major source of request rejection.

We advocate and use a request execution model, where a request is rejected (with an explicit message) if it fails to obtain a critical server resource within a specified time interval. This approach is shared by a vast majority of robust server architectures that bound request processing time in various ways (e.g., by setting a deadline for request completion), as opposed to a less robust approach, where a request is kept in the system indefinitely, until it is served (or is rejected by lower-level mechanisms such as TCP timeout). The former approach has the advantage of more efficiently freeing up server resources held by requests whose processing cannot be completed because of server capacity limitations.

Fig. 1 illustrates this application server architecture and the flow of a request through the system. Requests compete for two critical exclusively-held server resources: server threads and database (DB) connections; these resources are pooled by the web server and the application server respectively. Scheduling of requests to available threads and DB connections is done according to the request priority set by the RDRP module. The request with the highest priority is served first, with FIFO used as a tiebreaking policy. Timeout values for obtaining a thread and a DB connection are set to be 10s. If this timeout expires, the request is rejected with an explicit message. Note that some requests do not require database access, so they can be successfully served just by acquiring a server thread.

2.2. TPC-W Application

To test the benefits of the proposed request prioritization mechanisms, we use the TPC-W transactional web e-Commerce benchmark (TPC-W, 2005), that emulates an on-line store that
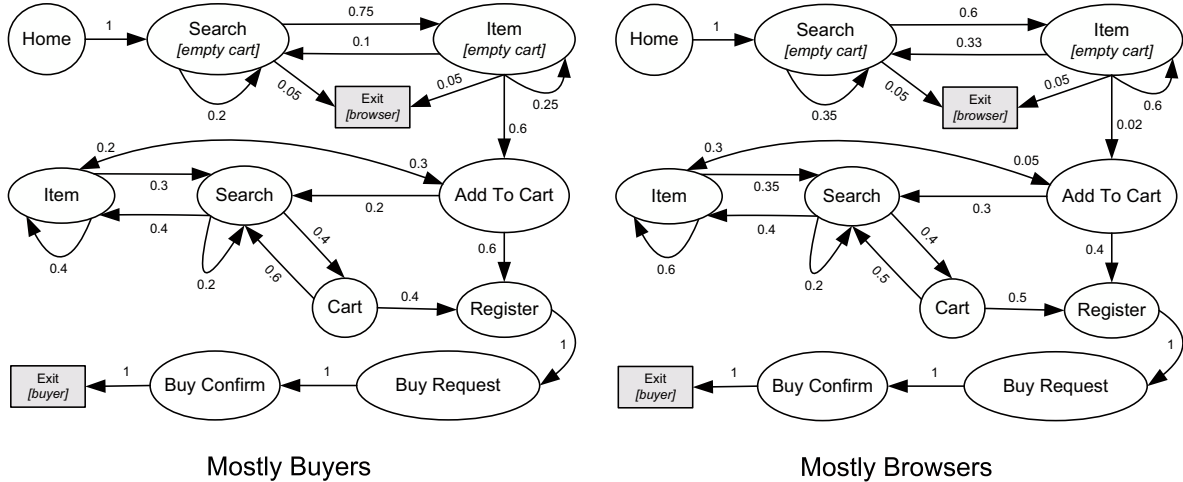
4

Figure 2: Two CBMGs used for TPC-W web workload.

sells books. The TPC-W specification describes in detail the application data structure and the 14 web invocations (WI) that constitute the web site functionality, and defines how they change the application data stored in the database. A typical TPC-W session consists of the following requests: a user starts web site navigation by accessing the Home page, searches for particular products (Search), retrieves information about specific items (Item Details), adds some of them to the shopping cart (Add To Cart), initiates the check-out process, registering and logging in as necessary (Register, Buy Request), and finally commits the order (Buy Confirm). We use our own implementation of the TPC-W benchmark, realized as a Java EE component-based application (TPC-W-NYU, 2005).

2.3. Web workload model

In this study, as in numerous other web server performance studies, we use synthetic web workloads, which are injected into a working application server environment using a load generator machine. Utilizing synthetic workloads is a common and widely adopted way to evaluate web server performance. Although not as realistic as using real web traces, this approach is more convenient for controlled exploration of the range of client behaviors.

Several session-based web workload models have been proposed, based on detailed analyses of real web traces (Menascé et al., 1999; Akula and Menascé, 2007). A dominant fraction of these models (Menascé et al., 1999; Carlstrom and Rom, 2002; Singhmar et al., 2004; Elnikety et al., 2004; Chen and Mohapatra, 2002), as well as workload generators of web server performance benchmarks, such as TPC-W (TPC-W, 2005), use first or higher-order Markov chains to model session structure.

Our study follows this practice and adopts the *Customer Behavior Model Graph* (CBMG) (Menascé et al., 1999) approach for session structure modeling. CBMG is a state transition graph (i.e., a first-order Markov chain), where states denote results of service requests (web pages), and transitions denote possible service invocations. Transitions in CBMG are governed by probabilities $p_{i,j}$ of moving from state i to state j ($\sum_j p_{i,j} = 1$). It was shown that web workloads consisting of several different CBMG session structures can approximate any given sequence of user requests (web request log) as "close" as desired, by appropriately choosing the model parameters (i.e., the

5

Table 1: Average breakdown of sessions by request type, for two CBMGs used for TPC-W web workload.

Request type	Session request breakdown	
	Mostly Buyers	Mostly Browsers
Home	10.0%	5.2%
Search	24.0%	36.0%
Item	24.7%	53.5%
Add To Cart	11.6%	1.2%
Cart	3.9%	1.1%
Register	8.6%	1.0%
Buy Request	8.6%	1.0%
Buy Confirm	8.6%	1.0%
Total	**100.0%**	**100.0%**

number of CBMGs and their transition probabilities), which in turn can be obtained from the web request log by the proposed clustering algorithm (Menascé et al., 1999). The greater the number of CBMGs in the workload model, the closer such an approximation can be made. In web workloads consisting of several CBMGs, each one of them represents a typical navigational pattern exhibited by the service users.

Session structure. Our workload for the TPC-W application is a 50%/50% mix of the two CBMGs shown in Fig. 2. We use simplified user session structures, which use only a subset of the TPC-W request types, but are rich enough to include essential application activities and represent requests with a wide range of functional and execution complexity. Both of the CBMGs in Fig. 2 use the same state transition structure, but with different transition probabilities. The "Mostly Buyers" CBMG produces user sessions that tend to buy products, while the "Mostly Browsers" CBMG produces more browsing-biased sessions. This results in different frequencies of requests being invoked by the two kinds of sessions (Table 1). Note that not all "Mostly Buyers" sessions result in a purchase, and analogously, not all "Mostly Browsers" sessions just browse the product catalog. The 50%/50% mix of the given "Mostly Buyers" and "Mostly Browsers" sessions results in approximately 52% of sessions finishing with a purchase. This value may be higher than most retail e-Commerce web sites see in real life, however such client behavior may be more characteristic of web sites providing online brokerage services, where a greater portion of user sessions results in completion of profit-bringing transactions of selling and buying stocks. We introduce this bias towards purchasing sessions to highlight the benefits of our request prioritization approach. However, we expect our methods to exhibit the same *relative* improvements even in workloads with fewer purchasing sessions.

Timing parameters. We model session inter-request user think times as exponentially distributed with mean 5s for the "Mostly Buyers" sessions and 10s for the "Mostly Browsers" sessions. If not stated otherwise, the flow of incoming new sessions is modeled as a Poisson process with arrival rate λ, which determines the overall load produced on the system (average request rate received by the service is $\lambda \cdot N$, where N is the average session length, in requests). The Poisson process produces relatively smooth sequence of events, and fails to model inherently bursty and self-similar traffic often observed at web sites (Wang et al., 2002). To better model the latter,
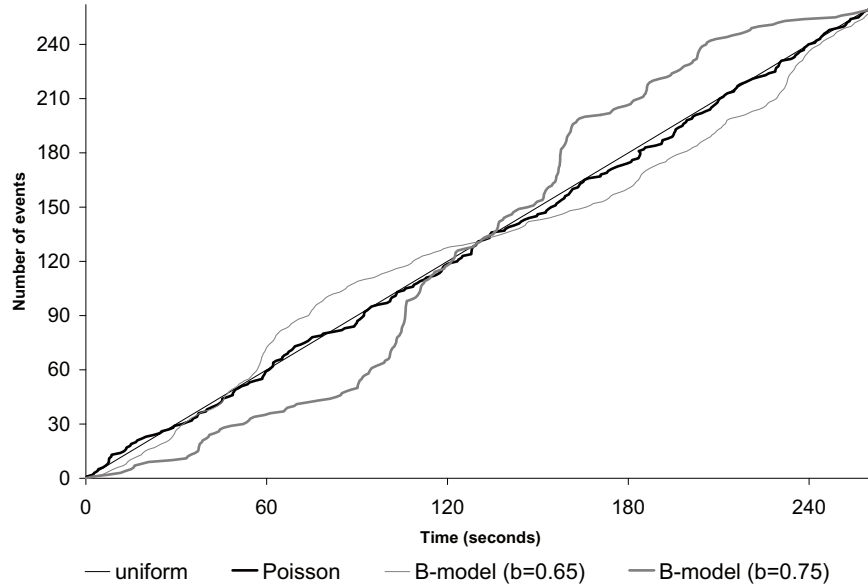
Figure 3: Event arrival patterns for the three processes: Poisson ($\lambda = 1$) and B-model (b=0.65 and b=0.75).

we additionally use the B-model (Wang et al., 2002), which has been shown to produce synthetic traces with burstiness matching that of real traffic. We use this model to produce load with different *degrees of burstiness* (determined by the b-parameter of the B-model), and do it in a way to only imitate local (short-lived) burstiness to avoid substantial shifting of massive event clusters to short time intervals. Specifically, we model two types of bursty load, one with $b = 0.65$ and another with $b = 0.75$,[1] and refer to these as "low-bursty" and "high-bursty" load respectively. In contrast with these two methods, we refer to the Poisson arrival model as "smooth." Fig. 3 shows the event arrival patterns for a Poisson process ($\lambda = 1$), and for the two B-model processes ($b = 0.65$ and $b = 0.75$) with the same average event arrival rate (1 event/s). This graph helps to visually assess the degree of event arrival burstiness produced by the different models.

2.4. Session reward and request cost specification

It is the service provider responsibility to define the *reward* (*profit*) function associated with the client session. The model we adopt in this study is simple yet general enough to encompass several possible applications: a reward value is defined for every request type of the service. The reward of the session is the sum of rewards of the requests in the session. Depending on the specific application, the reward "counts" only if the session completes successfully, or alternatively, whenever a reward-bringing request is invoked.

To illustrate the reward formulation, let us revisit the example scenarios presented in Section 1. In the online shopping scenario the profit of the service is reflected by the volume of items sold. So one way to define a reward function for the online store service is by assigning a reward value of 1 for the Add to Cart request: the shopping cart will contain as many items in it as the number

---

[1]In the original B-model study (Wang et al., 2002), the authors analyzed real web traces and inferred that the b-parameter for that traces ranged from 0.6 to 0.8, so we felt that values 0.65 and 0.75 would be reasonably representative.
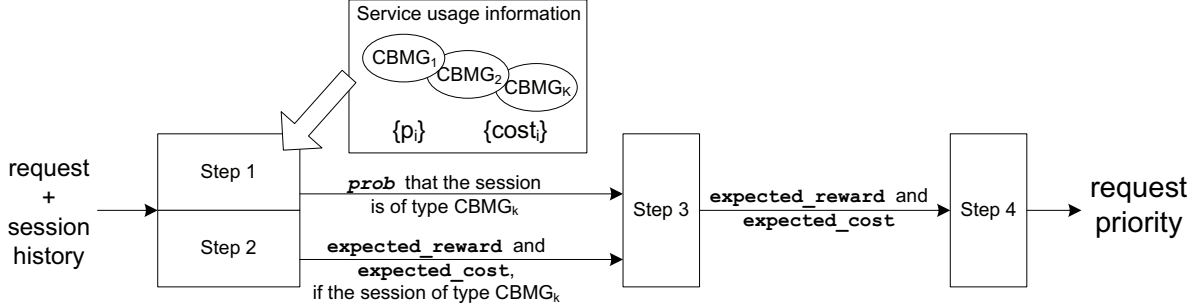
Figure 4: Logical steps of the RDRP method.

of times the Add to Cart request was executed. In the example of third party-sponsored advertisements, one can assign each web page a reward value based on the number of advertisements on it, or alternatively, on how much the advertisement sponsor would pay for a client's (potential) click on the advertisement link(s) displayed in the web page.

The cost of a session is similarly specified in terms of the *relative request execution cost* for each request type. This choice has the following rationale. Processing times for individual requests in typical e-Commerce services, including our TPC-W application, can vary widely by as much as two-to-three orders of magnitude. However, there tends to be much more variation across request types than for requests within the same type but with different request parameters (Chen et al., 2001; Elnikety et al., 2004). Information about the relative execution costs permits the RDRP Bayesian inference algorithm to be able to make adequate predictions of future server resource consumption by a session. Note that request execution costs can either be specified by the service provider (in abstract cost units), or determined by online request profiling as the average processing time of requests of a particular type.

3. Reward-Driven Request Prioritization

Our reward-driven request prioritization (RDRP) algorithms work with the assumption that information about the aggregate structure of the user load is known. Specifically, we assume that the workload consists of K CBMGs: $CBMG_1$, $CBMG_2, \ldots,$ $CBMG_K$. The probability of a session having the structure of $CBMG_k$ is p_k, $\sum_{k=1}^{K} p_k = 1$. For the session of structure $CBMG_k$, the probability of transition from state i to state j is $p_{i,j}^k$. We also assume that as stated earlier, that for each request type i, its *relative execution cost* $- \text{cost}_i -$ is known. We note that CBMG structures can be extracted from web server request logs through offline or online cluster analysis (Menascé et al., 1999), and the various probabilities and per-request type execution costs can be updated at runtime through request profiling.

Given the above, the RDRP mechanism works in the following way. For every incoming request, it looks at the sequence of requests already seen in the session and compares this sequence with the known CBMG structures of the session types comprising the user load. A Bayesian inference analysis estimates the probability that the given session is of type $CBMG_k$, for each $k = 1, \ldots, K$ (step 1). For each session type $CBMG_k$, the algorithm computes the values of *expected reward* and *execution cost*, resulting from the *future* requests of the session, assuming it had the structure $CBMG_k$ (step 2). This information is used to get the non-conditional values of

expected reward and execution cost of the future session's requests (step 3). These values are used then to define the *priority* of the request (step 4), which governs the scheduling of available server threads and DB connections to incoming requests (see Fig. 1). The logical sequence of the RDRP algorithm steps is depicted in Fig. 4 and is explained in detail below.

Step 1. $\Pr\{\mathrm{CBMG}_k \mid \mathrm{req\ hist}\}$, the Bayesian estimate that the session is of certain type CBMG_k (for a given history of session requests) is given by the following formula:

$$\Pr\{\mathrm{CBMG}_k \mid \mathrm{req\ hist}\} = \frac{\Pr\{\mathrm{req\ hist}|\mathrm{CBMG}_k\}\cdot p_k}{\sum\limits_{i=1}^{K} \Pr\{\mathrm{req\ hist}|\mathrm{CBMG}_i\}\cdot p_i} \tag{1}$$

where $\Pr\{\mathrm{req\ hist} \mid \mathrm{CBMG}_k\}$ is the probability of having a certain sequence of L requests $\{i_1, i_2, \ldots, i_L\}$ in a session of type CBMG_k and is determined as

$$\Pr\{\mathrm{req\ hist} \mid CBMG_k\} = \prod_{j=1}^{L-1} p_{i_j, i_{j+1}}^{k} \tag{2}$$

Note, that if the number of requests seen in the session so far is small or if the request sequence is not unique enough to substantially differentiate the session's structure among known session types CBMG_k, than most likely we will not be able to predict with high confidence which session type this session belongs to. But as the session progresses and has more requests in it, it's more likely that we will be able to predict the type of the session. From our experience, it takes at least 4-5 requests in a session until one of the $\Pr\{\mathrm{CBMG}_k \mid \mathrm{req\ hist}\}$ probabilities starts to dominate the others.

Timing parameters. In the basic Bayesian analysis of distinguishing among possible session types (equations (1) and (2)), we took into account only the CBMG state transition information. However, if session inter-request user think times differ for sessions of various CBMG types, than this additional information can be used in an attempt to make the Bayesian inference analysis more accurate. Imagine that we know the distribution of user think times for each CBMG session type comprising the load, in particular – their PDF functions, $\mathrm{PDF}_k(x)$, $k = 1, \ldots, K$ ($\mathrm{PDF}_k(x) = \Pr\{\mathrm{time} < x\}$, for CBMG_k), and that the observed session inter-request times are $t_1, \ldots, t_{L-1}$ (L is the number of requests seen in the session). Then equation (2) can be substituted by the following one:

$$\Pr\{\mathrm{req\ hist}, t_1, \ldots, t_{L-1} \mid \mathrm{CBMG}_k\} = \prod_{j=1}^{L-1} p_{i_j, i_{j+1}}^{k} \cdot \prod_{j=1}^{L-1} \mathrm{PDF}_k(t_j) \cdot (\Delta t)^{L-1} \tag{3}$$

where the infinitesimal time interval Δt appears in the equation, because the session inter-request times have supposedly continuous distributions. When equation (3) is substituted in equation (1), the infinitesimal value $(\Delta t)^{L-1}$ appears in both the numerator and the denominator, and cancels each other out. We call the RDRP scheme that involves inter-request timing considerations **RDRP(state+time)**, and the basic method – **RDRP(state)**.
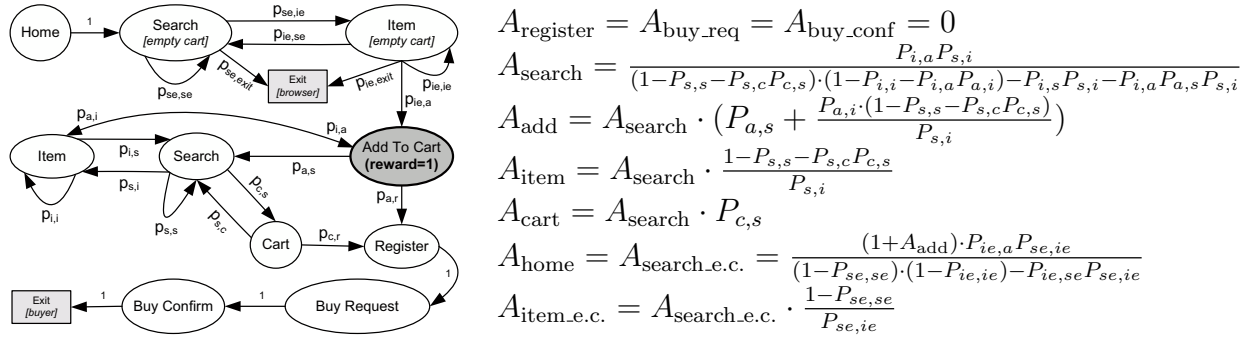
Figure 5: The graph structure of the CBMG used to represent our browsing and shopping scenario (left). Expressions for values A_i (right).

The equations shown on the right of Figure 5 are:

$$A_{\text{register}} = A_{\text{buy\_req}} = A_{\text{buy\_conf}} = 0$$

$$A_{\text{search}} = \frac{P_{i,a}P_{s,i}}{(1-P_{s,s}-P_{s,c}P_{c,s})\cdot(1-P_{i,i}-P_{i,a}P_{a,i})-P_{i,s}P_{s,i}-P_{i,a}P_{a,s}P_{s,i}}$$

$$A_{\text{add}} = A_{\text{search}} \cdot \left(P_{a,s} + \frac{P_{a,i}\cdot(1-P_{s,s}-P_{s,c}P_{c,s})}{P_{s,i}}\right)$$

$$A_{\text{item}} = A_{\text{search}} \cdot \frac{1-P_{s,s}-P_{s,c}P_{c,s}}{P_{s,i}}$$

$$A_{\text{cart}} = A_{\text{search}} \cdot P_{c,s}$$

$$A_{\text{home}} = A_{\text{search\_e.c.}} = \frac{(1+A_{\text{add}})\cdot P_{ie,a}P_{se,ie}}{(1-P_{se,se})\cdot(1-P_{ie,ie})-P_{ie,se}P_{se,ie}}$$

$$A_{\text{item\_e.c.}} = A_{\text{search\_e.c.}} \cdot \frac{1-P_{se,se}}{P_{se,ie}}$$

Step 2. To compute $\text{rew\_exp}\{\text{CBMG}_k\}$, the value of reward expected from the session's future requests, assuming it is of particular type CBMG_k, we just need to know the expected number of future session requests, for each request type (i.e., the expected number of future visits to each of the session's CBMG states). This information, combined with the assumption that reward is brought by individual requests allows us to compute the expected reward for the session. The expected number of future visits to a CBMG state is a Markov property of the CBMG and is determined only by the current state (i.e., by the current request) and the CBMG's state transition probabilities. For our sample shopping scenario, where reward is brought by the Add To Cart request, we have to compute A_i – the expected number of future visits to the Add To Cart state, if the current state is i. These values are determined by a set of linear equations, involving CBMG transition probabilities and can be computed mechanically (Menascé et al., 1999). For the CBMGs we showed earlier in Fig. 2, the general graph structure and expressions for A_i are shown in Fig. 5. We refer the reader to the original work (Menascé et al., 1999), where the CBMG apparatus was introduced and developed. The values of expected execution cost from the session's future requests are computed in a similar way.

Step 3. The (non-conditional) values of expected reward and execution cost of the future session's requests are computed as a linear combination of the corresponding conditional values (i.e., for specific CBMG session types) weighted with the probabilities that the session is of that particular type:

$$\text{rew\_exp} = \sum_{k=1}^{K} \text{rew\_exp}\{\text{CBMG}_k\} \cdot \Pr\{\text{CBMG}_k \mid \text{req hist}\}$$

$$\text{cost\_exp} = \sum_{k=1}^{K} \text{cost\_exp}\{\text{CBMG}_k\} \cdot \Pr\{\text{CBMG}_k \mid \text{req hist}\}$$

Step 4. The underlying idea of request prioritization is very simple – give higher priority to requests from sessions that are expected to bring more reward, while consuming less server resources. We use two different schemes to define request priority – one takes into account the cost of the requests seen in the session (we call this scheme RDRP-1), and the other (RDRP-2) does not:

$$\text{priority}_1 = \frac{\text{rew\_attained} + \text{rew\_exp}}{\textbf{cost\_incurred} + \text{cost\_exp}} \tag{4}$$
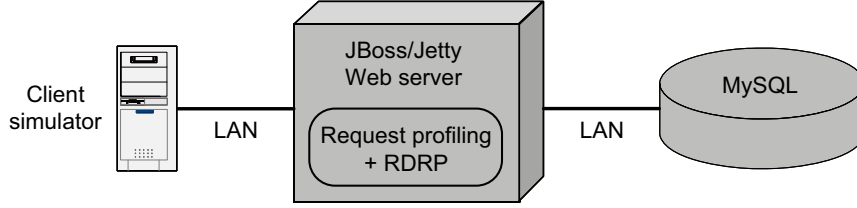
10

Figure 6: The server configuration used in the experiments.

$$\text{priority}_2 = \frac{\text{rew\_attained} + \text{rew\_exp}}{\text{cost\_exp}} \tag{5}$$

rew_attained and cost_incurred are the reward and the execution cost of the requests already seen in the session.

4. Evaluation

We start by describing the experimental setup and then present an evaluation of RDRP against alternative server-side schemes for managing application server resources.

4.1. Experimental setup

Server configuration. Our experimental infrastructure consists of a web application server and a separate database server, each running on a dedicated workstation, connected by a high-speed local-area network (Fig. 6). We use open-source Java EE application server JBoss (JBoss, 2010), bundled with Jetty HTTP/web server (Jetty, 2010), as a web application server and MySQL database (MySQL, 2010) with transactional InnoDB tables, for the database server. The database is treated as a black box and its configuration is kept default, with the exception of switching off database query caching.[2] JBoss/Jetty web application server is augmented as shown in Fig. 1 with the request profiling infrastructure and the RDRP mechanisms, implemented in a modular fashion as pluggable middleware services. We set the size of the server thread pool and the DB connection pool to 70 and 30 respectively (see Fig. 1)

Request profiling infrastructure. The Jetty HTTP/web server is used to gather high-level information about client requests, which are classified by their type (based on the URL pattern) and session affiliation. Various JBoss modules, such as the Database Connection Manager, are augmented with additional execution hooks to gather low-level information about the breakdown of request processing times spent at different request execution phases (e.g., waiting for a thread, processing in the database, etc.). When a request completes, the information about its execution is sent to the Request Profiling module (implemented as a separate middleware service), where it is added to a server-wide in-memory store. The Request Profiling module uses the high-level request flow information to maintain histories of session requests. The low-level request processing information is used to periodically update the values of relative request execution cost cost_i in the

---

[2]This was done intentionally to eliminate the effects of repeated request patterns in the synthetic workload, which resulted in non-uniform request processing performance in presence of database query caching.

Table 2: Average response times for the TPC-W request types used in the study, when executed in isolation (only one request is processed by the server at a time).

Request type	Response time (ms)
Home	30
Search	450
Item	15
Add To Cart	20
Cart	5
Register	5
Buy Request	150
Buy Confirm	100

RDRP algorithm, defined in our experiments as the average request processing time, without the time spent waiting for a thread or a DB connection.

TPC-W application. As stated earlier, this study uses our own implementation of the TPC-W benchmark, realized as a Java EE component-based application (TPC-W-NYU, 2005). The TPC-W application parameters (e.g., for database population) are chosen so as to achieve diverse execution complexity for different request types involved in the simulated sessions. Table 2 shows average request response times for the TPC-W request types (ranging from 5 ms to 450 ms), when executed in isolation (only one request is processed by the server at a time). This information is presented to illustrate relative execution complexity of requests, which range from very light ("Register" request, which does not require database access) to very heavy ("Search" request, which performs execution of complex database queries). When executed concurrently, the requests see larger response times, because of queueing delays for critical server resources (threads and DB connections) and possible database contention.

Client load. A separate workstation is used to produce client load and to gather statistics (Fig. 6). The client load simulates requests according to the CBMG structures discussed in Section 2.3. The maximum sustainable request rate of the server configuration under the resulted request mix is approximately 20 req/s, with the bottleneck being the MySQL database server.[3] The overall load produced on the system is determined by λ – the arrival rate of new sessions. We use different values of this parameter to generate server overload as well as underload conditions and report the load measured as a percentage of the system processing capacity. Each test run generates approximately 5000 sessions, with statistics gathered from the middle 80% portion of the run time to cut off warm-up and cool-down regions.

Reported metrics. For each experiment, we measure the reward attained by the service (measured in number of items bought by successfully completed sessions) and average request response times for sessions bringing different reward. We do not report the reward metric for the server underload

---

[3]This seemingly low server throughput is attributed, first, to the underprovisioned one generation old machines we were using for the experiments, and second, to the fact that we did not perform scrutinized database and TPC-W application tuning. However, we expect the relative performance improvements achieved by the RDRP methods to be similar in more powerful server environments.
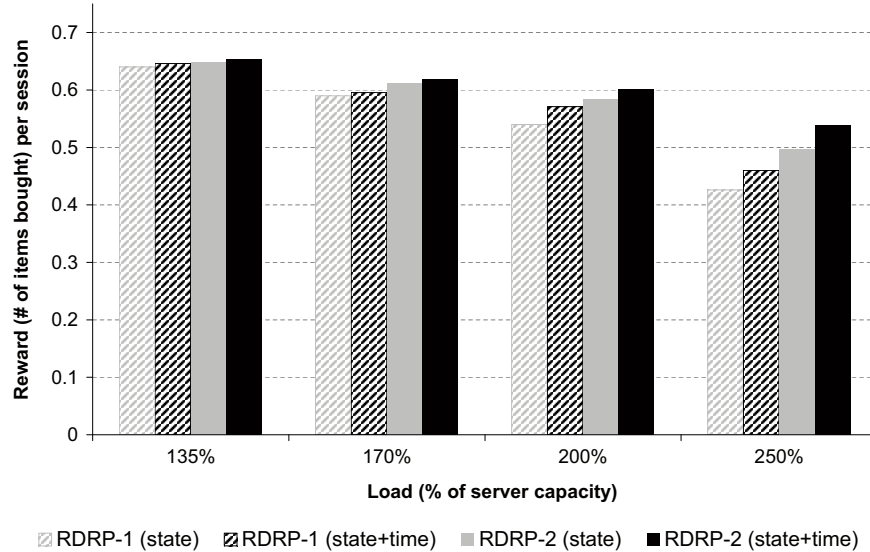
Figure 7: Comparison of benefits brought by the two flavors of the RDRP method. RDRP-1 takes into account the execution cost of all session's requests (seen and expected). RDRP-2 only takes into account the cost of current and the future requests of the session.

situation, because in this situation all sessions complete successfully, and each request scheduling algorithm produces the same reward value. Where absolute values of reward are reported, they are counted *per incoming user session*. This is done to show how close the employed algorithms are to the ideal situation, when all the *buying* sessions complete successfully, which brings the average per-session reward of **0.7** (this value is determined by the mix and the structure of the involved CBMGs shown in Fig. 2). In some of the experiments we show attained reward measured as a percentage of the reward value produced by the default FIFO request scheduling algorithm. This is done to emphasize the relative benefits that employed methods bring, compared with the default web application server policies.

4.2. Comparison of two priority schemes

We ran a set of experiments comparing the performance of the RDRP-1 and the RDRP-2 methods, corresponding to the two priority formulations in equations (4) and (5), under various load conditions. Fig. 7 compares the performance of the two methods under different amounts of server overload, for "smooth" session arrivals. The RDRP-2 method outperforms RDRP-1 in all scenarios, but especially under high client load. To informally understand this outcome, consider a session that is just one or two steps away from its completion (e.g., it is in the Register state in the CBMG of Fig. 2). The RDRP-2 method, according to equation (5), gives this request a higher priority than RDRP-1 (equation (4)), because it does not count the cost already incurred by the session. Consequently, under RDRP-1, the request might get rejected due to a low priority value, which will waste all the effort it took to bring the session to its nearly complete state. A careful examination of the logs produced during the experiments supports this explanation: the primary reason for the poor performance of RDRP-1 is the fact that some sessions are rejected with one or two requests left to complete the session, a phenomenon that never happens with RDRP-2. By ignoring the cost already incurred by the session, the RDRP-2 method appears to increase the like-

13

lihood of session completion as compared to its RDRP-1 counterpart. This also agrees with the economics theory, which argues that *sunk costs* (i.e., costs that have already been incurred and which cannot be recovered, like $\mathrm{cost\_incurred}$ in equation (4)), should not be taken into account when making rational decisions (Varian, 2005). In the rest of the experiments we used only the RDRP-2 algorithm, and refer to it from now on as simply the RDRP method.

4.3. Imitating the "history-based" approach

As stated in Section 1, an alternative method to prioritize client requests to boost service profit (reward) is by using a per-client history-based approach. Broadly considered, such an approach models the behavior of any application-specific technique in which all requests of a session are assigned a constant priority value and are scheduled according to this priority. The priority assignment can have arbitrary logic, for example, it can be done in an attempt to predict the client's future behavior based on the history of the client's previous purchases, or it can be determined solely by the client's membership status. The success of such approaches is determined, of course, by how good they are in predicting the client's behavior or, more precisely, the statistical correlation between assigned session priority and the actual reward brought by this session. To the best of our knowledge, prior work on workload characterization has not addressed such correlation in behavioral patterns (especially with the information that we need). We therefore employed the following scheme for producing a predefined correlation between the assigned session priorities and the actual rewards brought by the sessions. Each session announces in advance the reward it would bring, enabling the session prioritization mechanism to set the session's priority so that the statistical correlation (parameter c) between the assigned priorities and the sessions' rewards meets the predefined value. A value of $c = 1.0$ brings the best achievable performance because the prioritization algorithm always assigns to requests from the session, a priority value in direct correspondence with the reward the session will bring.

4.4. Performance of RDRP

We compare the relative costs and benefits of RDRP mechanisms against the following alternative server-side request scheduling and overload protection methods:

- Default FIFO request scheduling with no request prioritization.

- Session-Based Admission Control (SBAC), which admits approximately as many sessions as can be processed by the server capacity; all of the admitted sessions are allowed to complete successfully. This method is used only in the server overload situation.

- The per-client "history-based" approach described in Section 4.3. We run five sets of experiments with $c = 0, 0.25, 0.5, 0.75$, and 1.0.

- Our RDRP(state) and RDRP(state+time) methods, described in Section 3.

4.4.1. Server overload

First, we evaluate the behavior of the methods in server overload situations. We run four sets of experiments, modeling loads of 135%, 170%, 200%, and 250% of server capacity, for both the "smooth" (Poisson) and "high-bursty" (B-model with $b = 0.75$) client loads (see Section 2.3 for details). Fig. 8 shows the reward attained by the service, relative to the performance of the
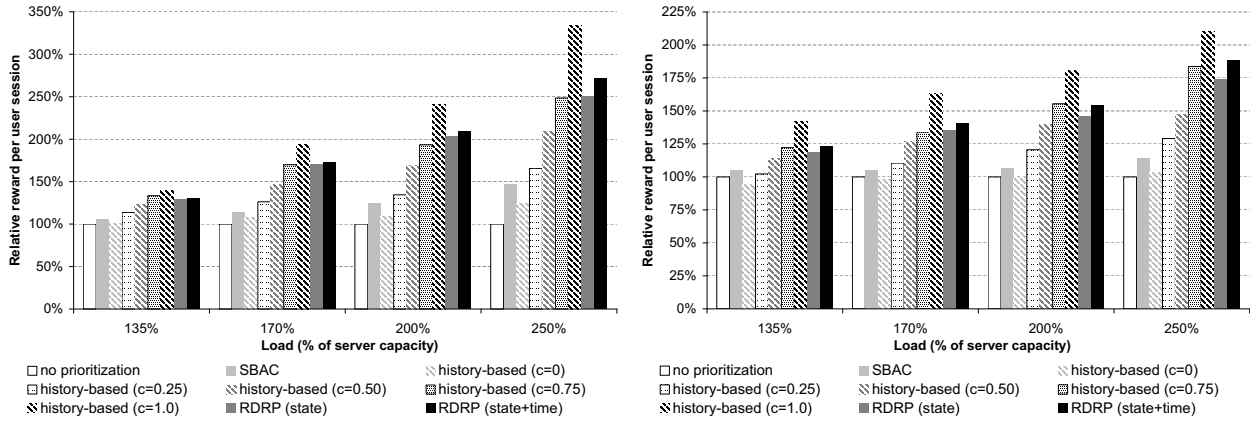
Figure 8: Reward (number of items bought by successfully completed sessions, per user session), relative to the default no-prioritization (FIFO) scheme, for the "smooth" Poisson (left) and "high-bursty" (right) client loads.

default FIFO request scheduling mechanism. Figs. 9 and 10 show average request response times for sessions bringing different reward, for the "smooth" and "high-bursty" client loads. Several conclusions can be drawn from the results of these experiments.

Reward attained. As expected, the default FIFO request scheduling policy shows the worst performance, because a request may get rejected anywhere in the session, which results in low successful session throughput. The SBAC method works better, because it at least allows the sessions that have started to complete successfully, however it does not try to necessarily admit those sessions that bring the greatest reward. The history-based approach shows an increase in reward attained with an increase of the correlation between assigned session priorities and sessions' rewards. Note that even with values of $c = 0.25$, this method already outperforms the SBAC algorithm. Finally, both RDRP methods significantly boost reward attained by the service. The RDRP(state+time) method works slightly better than RDRP(state), because it takes into account the inter-request time differences between more-profitable "Mostly Buyers" sessions and less-profitable "Mostly Browsers" sessions and better distinguishes between them. The theoretically best history-based ($c = 1.0$) method, of course, shows the best performance, however the history-based approach matches the performance of the RDRP algorithms, only for values of $c \geq 0.75$.[4] The performance of all algorithms goes down, when the client load experiences bursty behavior, because under bursty conditions the queues for critical server resources are more susceptible to rapid build-ups, which results in higher rates of request rejections. However, the relative advantages of RDRP over the other methods stay the same.

Request response times. All algorithms that perform request/session prioritization, and manage to correctly guess (at least to a certain degree) the session's reward, decrease request response times for sessions that bring non-zero reward, as compared to the SBAC method. Both RDRP methods perform on par with the history-based approach for values of $c \geq 0.5$. For the "smooth" client load, the RDRP algorithms reduce response times by up to 40% compared to SBAC, and show up

---

[4]Whether such good prediction is possible in real life, remains an open question, due to the lack of publicly available information with such statistics.
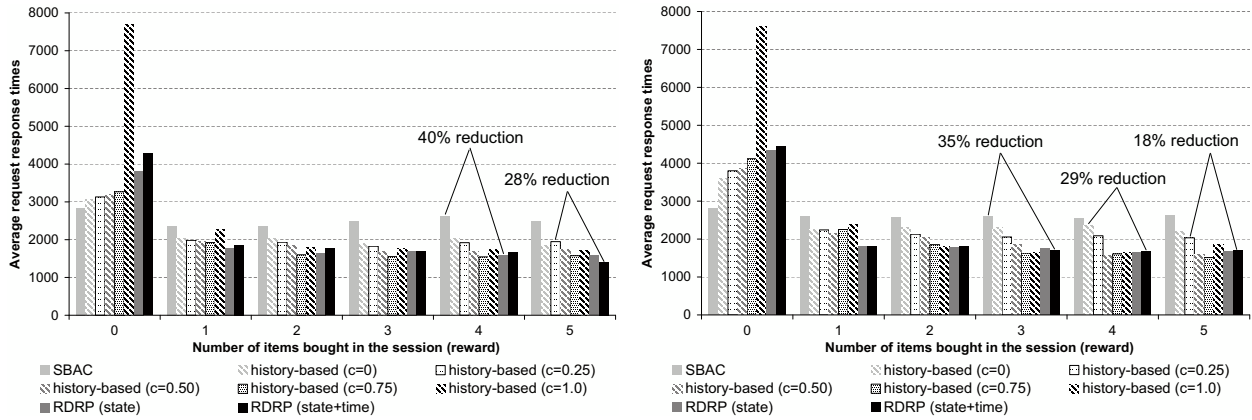
Figure 9: Average request response times for sessions that bring different reward, for "smooth" traffic, for the 135% server capacity (left) and the 170% server capacity (right) overload situations.
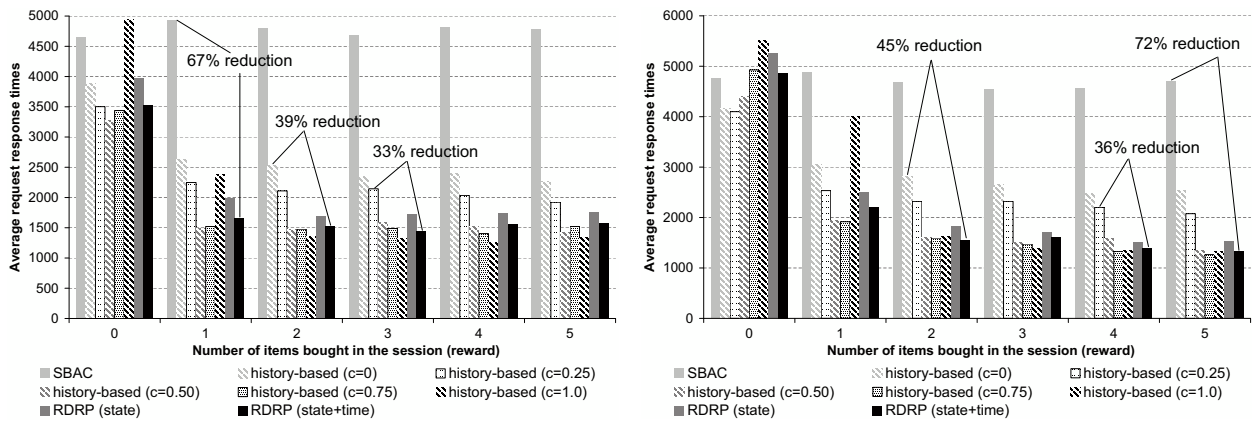

Figure 10: Average request response times for sessions that bring different reward, for "high-bursty" traffic, for the 135% server capacity (left) and the 170% server capacity (right) overload situations.

to 28% lower response times than the history-based approach with $c = 0$ and $c = 0.25$. For bursty client load, the difference is more pronounced: response times from the RDRP methods are lower than that from SBAC and the history-based approach with $c = 0$ and $c = 0.25$ by up to 72%, 45%, and 36%, respectively.

Note, that for "smooth" client load, the sessions with zero reward (i.e., *browsing* sessions) see significantly increased response times, when the history-based approach with $c = 1.0$ is applied (Fig. 9). This happens, because with the history-based approach, all browsing sessions (48% of all sessions, see Section 2.3 for an explanation) get the same (zero) priority, because the priority is defined as the session's reward, while the remaining 52% of sessions get a higher execution priority. Being all stuck in a single lowest-priority queue (with a FIFO tiebreaking policy), browsing sessions see higher rates of request rejections. This in turn produces higher response times for the session because a rejected request spends at least 10s in the system (before experiencing a timeout). Interestingly, this effect is reduced with bursty session arrivals (Fig. 10).
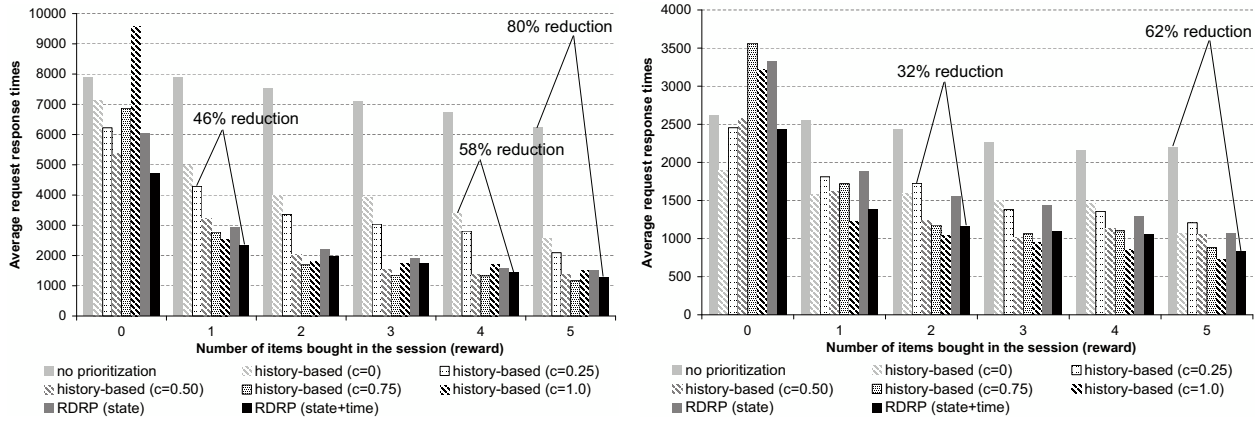
Figure 11: Average request response times for sessions that bring different reward, for 'high-bursty" traffic, for 100% server capacity (left) and the 80% server capacity (right) underload situations.
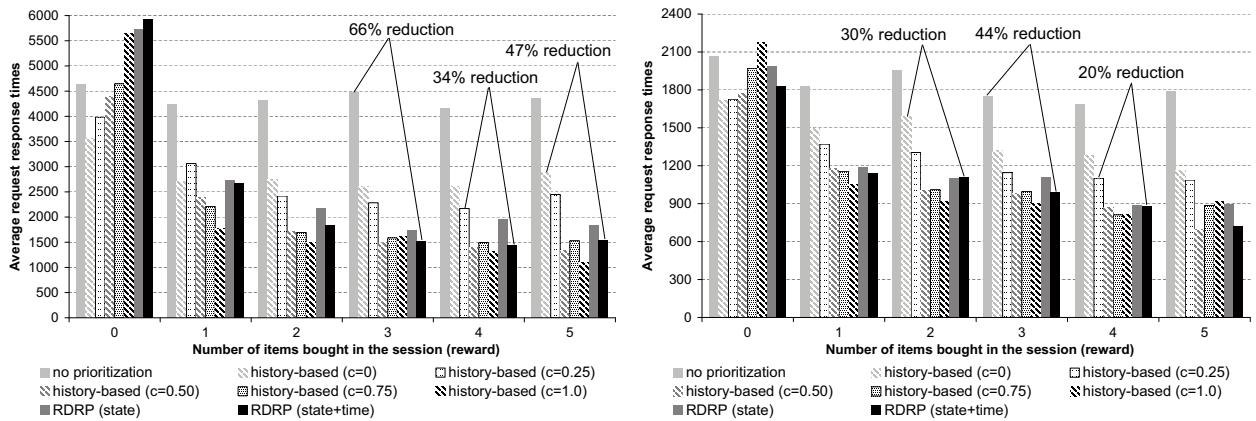

Figure 12: Average request response times for sessions that bring different reward, for "low-bursty" traffic, for the 100% server capacity (left) and the 85% server capacity (right) underload situations.

4.4.2. Server underload

For server underload situations, we ran experiments with both "smooth" (Poisson) and bursty client loads. The Poisson-modeled web workload generated such a smooth flow of request arrivals, that all request scheduling algorithms showed more or less the same performance. This happened because the server queues for the critical resources (threads and DB connections) almost never built up.

Experiments with the bursty client load showed very different behavior. Fig. 11 and 12 show average request response times for sessions bringing different reward, for the two bursty client loads.[5] Several conclusions can be drawn from the results of these experiments.

The RDRP methods (as well as the history-based approaches) decrease request response times for the sessions that bring non-zero reward. This happens because with bursty arrivals (unlike the

---

[5]The experiments labeled as "100% of server capacity" were actually ran at a rate slightly lower than the system capacity, which experienced slight variations because of the non-deterministic behavior of the web application server. This ensured that our experiment did not slide into the overload mode of server operation.

smooth arrival case described above), the queues for the critical server resources (server threads and DB connections) occasionally build up, and the request prioritization mechanisms minimize the queueing delays seen by the sessions that bring more reward by assigning their requests higher priorities. For "high-bursty" traffic, the effects of request prioritization are visible for loads above approximately 70% of server capacity (in Fig. 11 we show experiments with the load of 100% and 80% of server capacity), while for the "low-bursty" traffic, the effects are visible for the load in the range of 85%–100% of server capacity.

As in the server overload situation, the performance of the RDRP methods is matched by the history-based approach only for values of $c \geq 0.5$. Under "high-bursty" traffic, RDRP outperforms the history-based method by up to 58% (for $c = 0$) and 46% (for $c = 0.25$). This advantage of RDRP over the history-based approach diminishes a bit under "low-bursty" traffic conditions (Fig. 12). The default FIFO method performs worst of all. It is interesting to note that even the history-based approach with $c = 0$, which is not supposed to ever correctly guess the session's reward, gives lower response times (for all reward values, including 0) than the default FIFO request scheduling scheme.

To our understanding, this behavior happens for the following reason. The request scheduling algorithm we adopt to imitate a history-based approach with $c = 0$ works by uniformly assigning priorities to sessions as integer values in the range of 0 to 100 (this process does not correlate with the session reward, therefore corresponds to $c = 0$). Some sessions get higher priorities than the other, and all sessions are uniformly sorted into a discrete number of priority buckets. Unlike the FIFO scheduling case, where all requests have to wait in one long queue produced by a traffic burst, the uniform session prioritization scheme permits some sessions to sneak ahead of other sessions. This perturbs the waiting times seen by requests sufficiently so as to achieve an average response time lower than that seen by the FIFO case.

5. Related Work And Discussion

The notion of a *web session* — a cornerstone in this work — representing structural organization of client communication with Internet services was first investigated by Krishnamurthy and Rolia (1998) and by Cherkasova and Phaal (2002). Since then several other studies have explored session characterization of web workloads (Menascé et al., 1999; Shi et al., 2002; Akula and Menascé, 2007). The work by Llambiri et al. (2003) acknowledged that *service usage patterns* affect the performance of Internet services, and that understanding the nature of user workloads is crucial for properly designing and provisioning web servers. Our study follows this trend, by using aggregated information about client usage of an e-Commerce service to boost the service profit or other application-specific reward that clients bring to the service provider. Our work combines information about the workload structures seen at a service with Bayesian inference analysis to guide the scheduling of bottleneck server resources to incoming client requests.

The idea of profiling client requests and gathering service usage statistics, which later can be used for various purposes, is itself not new (Srivastava et al., 2000; Ataullah, 2007). An array of data mining techniques has been proposed to extract information from web access logs for myriad applications, including one most relevant for this work: to discover customers' behavioral patterns (Lee and Yen, 2007; Chena et al., 2009). Our use of this information to influence scheduling of server resources is somewhat novel, differing from past applications of such information that

have ranged from reducing user-perceived latencies for personalized web sites (Frias-Martinez and Karamcheti, 2003) to improving web server caching and prefetching behavior (Yang et al., 2001).

User-perceived response times are determined by two factors: the quality of network transmission and the processing capacity of the server. With the rapid Internet expansion and the client base moved away from the slow dial-up connections, most of the (non-mobile) users nowadays have fast access to the Internet, which makes the server-side request processing time typically a dominant factor in the overall response delay. Therefore, fast execution of requests at the server side has become the key factor in providing user perceivable performance. In our work, we concentrate on *server-side* mechanisms that improve service performance. Our mechanisms are orthogonal to such approaches for improving service performance as service distribution, load balancing, and content adaptation. They are also independent of any network-level performance improvement mechanisms, application of which proved beneficial in the context of mobile e-Commerce (Awan and Singh, 2006; Kim and Seo, 2006).

Various forms of *admission control* have been used to prevent services from being overwhelmed in the presence of persistent or transient overload. Among these, Session-Based Admission Control (SBAC) (Cherkasova and Phaal, 2002) is suitable for session-oriented client loads. An overloaded service can experience a severe loss of throughput measured in completed (successful) sessions while still maintaining its throughput measured in requests per second. This happens because a request can be rejected anywhere in the session, even if the session has already had a lot of its requests served and is very close to completion. The SBAC method works by admitting as many sessions as can be processed by the service, trying to make sure that if a client starts a session with the service, it will be successfully completed. However, this approach is oblivious to any application-specific information, e.g., to the profit brought to the service by different sessions. A variation of this approach was proposed by Guitart et al. (2007): the authors devised an adaptive session-based overload control strategy based on SSL (Secure Socket Layer) connection differentiation and admission control.

Several studies investigated the effects of request *scheduling* and *prioritization* on web server performance, for general database-driven dynamic web sites (Elnikety et al., 2004), and for e-Commerce web sites in particular (Schroeder et al., 2006; Alonso et al., 2007; Zhou et al., 2006). It was shown that request response times and server throughput can be improved by employing such scheduling algorithms as Shortest Job First (SJF) (Elnikety et al., 2004) and Shortest Remaining Processing Time First (Verma and Ghosal, 2003). Some of these studies used request scheduling algorithms combined with admission control policies (Chen et al., 2001; Elnikety et al., 2004). Our work shares the same goals as these efforts, but its use of more sophisticated scheduling policies is somewhat constrained by the hooks exposed by the underlying middleware. In web studies focusing on static content, the *cost* of servicing a job is usually approximated by the size of the downloaded file. For web sites serving dynamic content (e.g., e-Commerce services), it was noticed that request processing times depend primarily on the *request type* rather than on the parameters of the request (Chen et al., 2001; Elnikety et al., 2004). Our work shares the same observation, using fine-grained request profiling to determine first absolute and then relative request processing times for different request types. An analogous technique is used by Elnikety et al. (2004).

A notable difference from our work is that for the most part, the request scheduling studies above do not pursue the goal of increasing likelihood of session completion (even if they take into

account session-oriented nature of client workloads). An exception is the work of Chen and Mohapatra (2002) on the Dynamic Weighted Fairing Sharing request scheduling algorithm (DWFS), which, among other goals, tries to avoid processing of requests that belong to sessions that are likely to be aborted in the near future. In trying to increase session completion rate, this study shares a commonality with our work, which is oriented towards completion of sessions that bring more service reward.

There have been a number of studies on profit-aware performance management and profit maximization for e-Commerce services. One of the first was Menascé et al. (2000), who described a priority-based resource management policy for a retail e-Commerce web site aiming at maximizing profit, where customers are classified based on session length and the accumulated money in their shopping cart. A customer navigating the site for too long with not much value in their shopping cart is given a low priority in terms of processor and disk use.

Several studies proposed, as we do in this work, profit-based admission control and request scheduling techniques (Zhang et al., 2003; Carlstrom and Rom, 2002; Verma and Ghosal, 2003; Tan et al., 2005). Zhang et al. (2003) developed a Profit Aware QoS policy (PAQoS), aimed at maximizing the web site's profit under SLA constraints. Carlstrom and Rom (2002) proposed using queuing of requests based on their types, where a reward function corresponding to the service provider's objective is maximized using techniques for nonlinear optimization. Verma and Ghosal (2003) proposed an admission control technique to maximize profit of a service, given a set of Service-Level Agreements (SLAs) that specify reward and penalty parameters of the service. There are some notable differences between these studies and our approach. These studies assume that profit is brought by individual requests, while our RDRP approach also allows to count reward as attained only if the session is completed successfully. Most of them also assume a Generalized Processor Sharing (GPS) model for request execution, rather than a model of prioritized scheduling of requests to *exclusively-held* resources, such as server threads and DB connections. In our opinion, the latter model is a closer match to modern web application server architectures.

Ataullah (2007) shares our vision that (1) in overload conditions online businesses should identify valuable user sessions and ensure their completion; and that (2) taking into account user behavior is important for maximizing service profit. To this end, the author introduced MyQoS, a framework for identifying valuable user sessions and collecting other service usage information, which can be used in application-specific service differentiation mechanisms intended to boost service profit. Wang and Yue (2009) proposed a simple profit-aware admission control mechanism for shopping web sites that gives a higher execution priority to clients who made purchases before and a lower execution priority to all other clients. This approach is similar to the history-based approach described in this paper, which is shown to have inferior performance (both in profits gained and response times) to our RDRP approach, if the correlation between the past and present behaviors of service clients is weak.

Shaaban and Hillston (2009) proposed Cost-Based Admission Control (CBAC) for Internet Commerce systems, in which rather than rejecting requests in an overload situation, *price variation* is used to encourage customers to postpone their requests and return later, by offering them discounts. For those customers who decide to anyway move forward with the business transaction (e.g., purchase) the prices are increased. First, this approach requires more tight integration of the business and technical components of an e-Commerce service, than usually seen in modern Internet Commerce systems. Such tight integration may not be always possible for various business

reasons. For example, the service provider (business owner) may not be willing to sacrifice some profit, by giving away discounts to those customers who agree to return later. Second, increasing the price for those clients who proceed with the transaction may persuade some of them from using the service, perceived as expensive, in the future. On the contrary, our RDRP techniques are able to achieve profit maximization without incurring the aforementioned drawbacks.

Although this paper has demonstrated our work on a sample TPC-W application emulating an online store selling books, the underlying ideas and the approach are broadly applicable to all e-Commerce services accessed in a session-oriented fashion, because of the generic way the profit (reward) is specified in the RDRP method. This is in contrast to several recent studies that target specific business application or function. For example, in the work of Menascé and Akula (2007), a request dispatching framework was proposed that aims at improving performance of online auction sites. While the work of Singhmar et al. (2004) focused specifically on improving performance of online shopping services. In that work, the authors proposed a combined LIFO-Priority scheme for overload control of a retail e-Commerce web site, where all service requests are divided among browser and *revenue-generating transaction* requests. LIFO scheduling is applied to the browser requests, while the revenue-generating requests are given the highest priority. Note however, that while the study shows successful execution of revenue-generating requests, unlike our work, it does not explicitly address the issue of ensuring successful completion of buying sessions that contain such requests.

In this study, we have used synthetic web workloads. Utilizing artificial workloads is a common and widely adopted way to evaluate web server performance. Although not as realistic as using real web traces, this approach is more convenient for controlled exploration of the range of client behaviors. The ultimate test of the proposed RDRP method, before one can claim that it can be used universally in a real commercial setting, should be done utilizing real e-Commerce web logs, so in the future we plan to evaluate our approach using real web traces.

In the future, we also would like to verify the applicability of the RDRP method to other types of e-Commerce applications, such as online auctions, applications incorporating third party-sponsored advertisements, e-Learning applications, and possibly online games. For example, there has been a report from the industry that the success of a network game highly depends on the response time and request rejections (Claypool and Claypool, 2006). Another interesting direction for future work is to investigate the applicability of the RDRP method in mobile commerce (Awan and Singh, 2006), for example in the context of Location-Based Services (Kupper, 2005).

6. Conclusion

In this paper we have proposed *Reward-Driven Request Prioritization* (RDRP) mechanisms, which maximize the profit (or any other application-specific reward) attained by an e-Commerce service, by dynamically assigning higher execution priorities to the requests whose sessions are likely to bring more profit (reward) to the service. We have implemented the proposed methods as pluggable middleware mechanisms in the Java EE application server JBoss (JBoss, 2010), and tested them on the TPC-W benchmark application (TPC-W, 2005) using CBMG-based web workloads. Our experiments showed that RDRP techniques yield benefits in both underload and overload situations, for both smooth and bursty client behavior, against state-of-the-art alternatives such as session-based admission control and history-based session prioritization approaches. In

the situation of service underload the proposed mechanisms gave better response times for the clients that brought more profit. In the situation of service overload, the mechanisms ensured that sessions that brought more profit were more likely to complete successfully and that the aggregate profit attained by the service increased compared to other solutions. Additionally, we showed that the history-based approach matched performance of our RDRP mechanisms only if the correlation between the clients' past and future behaviors reached the mark of 75% for the profit attained, and 50% – for the request response times.

References

Akula, V., Menascé, D. Two-level workload characterization of online auctions. Electronic Commerce Research and Applications, 6, 2, 2007, 192–208.

Alonso, J., Guitart, J., Torres, J. Differentiated quality of service for e-Commerce applications through connection scheduling based on system-level thread priorities. In Proceedings of the 15th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP'07), Naples, Italy, February 2007.

Ataullah, A. MyQoS: A profit oriented framework for exploiting customer behavior in online e-Commerce environments. In Proceedings of the 8th International Conference on Web Information Systems Engineering (WISE'2007), Nancy, France, December 2007.

Awan, I., Singh, S. Performance evaluation of e-Commerce requests in wireless cellular networks. Information and Software Technology, 48, 6, 2006, 393–401.

Barnes, D., Mookerjee, V. Customer delay in e-Commerce sites: Design and strategic implications. In G. Adomavicius and A. Gupta (eds.), Business Computing, Handbooks in Information Systems, Vol. 3, Emerald Group Publishing, Bradford, England, UK, February 2009, 74–85.

Carlstrom, J., Rom, R. Application-aware admission control and scheduling in web servers. In Proceedings of the 21st IEEE International Conference on Computer Communications (INFOCOM'02), New York, NY, USA, June 2002.

Chen, H., Mohapatra, P. Session-based overload control in QoS-aware web servers. In Proceedings of the 21st IEEE International Conference on Computer Communications (INFOCOM'02), New York, NY, USA, June 2002.

Chen, X., Chen, H., Mohapatra, P. An admission control scheme for predictable server response time for web accesses. In Proceedings of the 10th International World Wide Web Conference (WWW'01), Hong Kong, China, May 2001.

Chena, Y.-L., Kuoa, M.-H., Wub, S.-Y., Tang, K. Discovering recency, frequency, and monetary (RFM) sequential patterns from customers' purchasing data. Electronic Commerce Research and Applications, 8, 5, 2009, 241–251.

Cherkasova, L., Phaal, P. Session-based admission control: A mechanism for peak load management of commercial web sites. IEEE Transactions on Computers, 51, 6, 2002, 669–685.

Claypool, M., Claypool, K. Latency and player actions in online games. Communications of the ACM, 49, 11, 2006, 40–45.

Elnikety, S., Nahum, E., Tracey, J., Zwaenepoel, W. A method for transparent admission control and request scheduling in dynamic e-Commerce web sites. In Proceedings of the 13th International World Wide Web Conference (WWW'04), New York, NY, USA, May 2004.

Frias-Martinez, E., Karamcheti, V. Reduction of user perceived latency for a dynamic and personalized web site using web-mining techniques. In Proceedings of the 5th ACM SIGKDD Workshop on Web Mining and Web Usage Analysis (WEBKDD'03), Washington, DC, USA, August 2003.

Guitart, J., Carrera, D., Beltran, V., Torres, J., Ayguadé, E. Designing an overload control strategy for secure e-Commerce applications. Computer Networks, 51, 15, 2007, 4492–4510.

Java EE. Java Platform Enterprise Edition. `http://java.sun.com/javaee/`. Accessed in February 2010.

JBoss. Java EE Application Server. `http://www.jboss.org`. Accessed in February 2010.

Jetty. HTTP Server and Servlet Container. `http://jetty.mortbay.org`. Accessed in February 2010.

Kim, J.-S., Seo, S.-K. Experiment and analysis for QoS of e-Commerce systems. Journal of Theoretical and Applied Electronic Commerce Research, 1, 3, 2006, 1–15.

Krishnamurthy, D., Rolia, J. Predicting the performance of an e-Commerce server: Those mean percentiles. In Proceedings of the 1st ACM SIGMETRICS Workshop on Internet Server Performance (WISP'98), Madison, WI, USA, June 1998.

Kupper, A. Location-Based Services: Fundamentals and Operation. Wiley, Hoboken, NJ, USA, 2005.

Lee, Y.-S., Yen, S.-J. Mining web transaction patterns in an electronic commerce environment. In Advances in Web and Network Technologies, and Information Management: APWeb/WAIM'07 International Workshops, Huang Shan, China, June 2007, Springer, Lecture Notes in Computer Science, Vol. 4537, 74–85.

Llambiri, D., Totok, A., Karamcheti, V. Efficiently distributing component-based applications across wide-area environments. In Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS'03), Providence, RI, USA, May 2003.

Menascé, D., Akula, V. A business-oriented load dispatching framework for online auction sites. In Proceedings of the 4th IEEE International Conference on Quantitative Evaluation of Systems (QEST'2007), Edinburgh, Scotland, UK, September 2007.

Menascé, D., Almeida, V., Fonseca, R., Mendes, M. A methodology for workload characterization of e-Commerce sites. In Proceedings of the 1st ACM Conference on Electronic Commerce (EC'99), Denver, CO, USA, November 1999.

Menascé, D., Almeida, V., Fonseca, R., Mendes, M. Business-oriented resource management policies for e-Commerce servers. Performance Evaluation, 42, 2–3, 2000, 223–239.

Moskalyuk, A. IT Facts: e-Commerce research blog on ZDNet.com, November 2006. `http://blogs.zdnet.com/ITFacts/?p=12030`. Accessed in February 2010.

MySQL. Relational Database. `http://www.mysql.com/`. Accessed in February 2010.

Schroeder, B., Harchol-Balter, M., Iyengar, A., Nahum, E. Achieving class-based QoS for transactional workloads. In Proceedings of the 22nd International Conference on Data Engineering (ICDE'06), Atlanta, GA, USA, April 2006.

Shaaban, Y. A., Hillston, J. Cost-based admission control for Internet Commerce QoS enhancement. Electronic Commerce Research and Applications, 8, 3, 2009, 142–159.

Shi, W., Wright, R., Collins, E., Karamcheti, V. Workload characterization of a personalized web site – and its implications for dynamic content caching. In Proceedings of the 7th International Workshop on Web Caching and Content Distribution (WCW'02), Boulder, CO, USA, August 2002.

Singhmar, N., Mathur, V., Apte, V., Manjunath, D. A combined LIFO-priority scheme for overload control of e-Commerce web servers. In Proceedings of the IEEE RTSS International Infrastructure Survivability Workshop (IISW'04), Lisbon, Portugal, December 2004.

Srivastava, J., Cooley, R., Deshpande, M., Tan, P. Web usage mining: Discovery and applications of usage patterns from web data. ACM SIGKDD Explorations, 1, 2, 2000, 12–23.

Tan, Y., Moinzadeh, K., Mookerjee, V. Optimal processing policies for an e-Commerce web server. INFORMS Journal On Computing, 17, 1, 2005, 99–110.

TPC-W. Transactional Web e-Commerce Benchmark, 2005. `http://www.tpc.org/tpcw/`. Accessed in February 2010.

TPC-W-NYU. A Java EE implementation of the TPC-W benchmark, November 2005. `http://www.cs.nyu.edu/totok/professional/software/tpcw/tpcw.html`. Accessed in February 2010.

VanBoskirk, S., Li, C., Parr, J. Keeping customers loyal. Industry report. Forrester Research, Cambridge, MA, USA, May 2001.

Varian, H. R. Intermediate Microeconomics: A Modern Approach, Seventh Edition. W. W. Norton & Company, New York, NY, USA, 2005.

Verma, A., Ghosal, S. On admission control for profit maximization of networked service providers. In Proceedings of the 12th International World Wide Web Conference (WWW'03), Budapest, Hungary, May 2003.

Wang, H., Yue, C. Profit-aware overload protection in e-Commerce web sites. Journal of Network and Computer Applications, 32, 2, 2009, 347–356.

Wang, M., Chan, N., Papadimitriou, S., Faloutsos, C., Madhyastha, T. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. In Proceedings of the 18th International Conference on Data Engineering (ICDE'02), San Jose, CA, USA, February 2002.

Yang, Q., Zhang, H., Li, I., Lu, Y. Mining web logs to improve web caching and prefetching. In Proceedings of the 1st Asia-Pacific Conference on Web Intelligence (WI'01), Maebashi, Japan, October 2001.

Zhang, Q., Smirni, E., Ciardo, G. Profit-driven service differentiation in transient environments. In Proceedings of the 11th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'03), Orlando, FL, USA, October 2003.

Zhou, X., Wei, J., Xu, C.-Z. Resource allocation for session-based two-dimensional service differentiation on e-Commerce servers. IEEE Transactions on Parallel and Distributed Systems, 17, 8, 2006, 838–850.