E-representative: a scalability scheme for e-commerce *

Wagner Meira Jr. † Daniel Menascé ‡ Virgílio Almeida † Rodrigo Fonseca †

† Dept. of Computer Science
Universidade Federal de Minas Gerais
Belo Horizonte, MG 31270-010 Brazil
{meira,virgilio,rfonseca}@dcc.ufmg.br

‡ Dept. of Computer Science
George Mason University
Fairfax, VA 22030 USA
menasce@cs.gmu.edu

Abstract

In order to meet the quality of service demanded by a growing number of online customers, e-commerce services need to use scalability techniques. This paper introduces the concept of e-commerce representatives, a means of scaling the performance of e-commerce services. E-representatives are programs that execute on a cache server or at nearby machines. E-representatives can be implemented using redirection, a mechanism available in popular cache servers. Using analytical and simulation models, we show the potential performance gains obtained by e-commerce sites that distribute their services among e-representatives and contribute to reduce bandwidth consumption and network latency.

Keywords: e-Commerce, QoS, scalability, simulation, analytic models

1 Introduction

The growth of electronic commerce has resulted in high demands on sites that provide such services. As a consequence, servers often get overloaded and the quality of the services they provide decreases. Furthermore, overloaded communication channels are responsible for a significant portion of the overall user waiting time, regardless of the efficiency of the servers. To meet the quality of service demanded by a growing number of online customers, e-commerce services require scalability techniques.

According to recent surveys, most of the load of e-commerce servers comes from customers that visit the site and do not purchase any goods. These same surveys showed that the buy to visit ratio is around 5% [14]. Non-critical tasks such as searching, browsing, and providing detailed information about products are as important as order placement and payment, since the former help establishing customer habits and preferences. Moreover, the competition among electronic stores is motivating enhancements to existing services and the creation of new mechanisms to attract and keep customers, demanding even more computational power from servers.

A common and successful strategy for improving WWW services is caching [17]. Frequently requested objects are stored in sites close to users, in terms of connectivity. Caching proxy servers reduce user-perceived latency, network traffic, and load on the servers that have objects replicated. Although caching is a proven technology for static objects (i.e., HTML pages), its use is still under discussion [3, 11] for dynamic objects such as those generated by `cgi-bin` scripts, a very common approach for implementing e-commerce servers. The stateful nature of e-commerce services poses another issue in replicating transaction-related information because ACID properties associated with these transactions must be guaranteed.

A central question for scaling e-commerce services is how to distribute the operations offered by an electronic store. Cache servers seem to be an option for performing such tasks since they are usually installed in ISPs and other heavy-traffic network locations. Moreover, cache servers usually take advantage of similarities among customers of a given community. For instance, a cache server may concentrate all traffic generated by employees of a given corporation, who may share some interests.

In this paper, we propose a technique called *e-representatives* to improve the quality of service of e-commerce services and make them more scalable. The idea behind e-representatives is to distribute non-critical tasks (e.g., browsing, searching), relieving e-commerce servers, which may focus on critical tasks such as registering, ordering and paying.

E-representatives are typically small programs that execute on a cache server or nearby machines (connection-wise). These programs are provided by the represented stores themselves, and may, besides answering user queries, enforce security, use private protocols, and keep user pro-

*This research is supported by grants from CNPq/Brazil and NSF, and by Project SIAM - DCC - UFMG, grant MCT - FINEP - PRONEX number 76.97.1016.00.

files, among other tasks. Representatives can be installed in current cache servers by using their redirecting capabilities. In addition to reducing user's latency and bandwidth usage, representative functions can offer some financial reward to companies that operate cache servers, based on the amount of services performed on behalf of the store server.

This paper is organized as follows. The next section shows a simple queuing network model that quantifies the potential benefits of distributing e-commerce services. Section III introduces concepts associated with the e-representative technique. Section IV discusses workload characterization of online retailer sites. In section V, we describe the architecture of e-commerce e-representatives in detail, showing how e-commerce operations are replicated. Section VI presents the simulation model of an e-representative and analyzes the numerical results. Concluding remarks and future work are in section VII.

2 A Case for Distributing E-commerce Services

This section uses a simple queuing network model (QN) to quantify the potential benefits of distributing e-commerce services. The results of this section demonstrate that, under certain circumstances, some of the services provided by a virtual store can be provided by *e-representatives* running closer to the end-users. To that end, some of the data maintained by the e-commerce server has to be replicated at the e-representatives. Documents, or results of queries, not replicated at the e-representative may be temporarily cached by the e-representative or at nearby cache servers. A more detailed description of e-representatives is given in Sec. 4 and the results of a detailed simulation are discussed in Sec. 5.

Consider the QN shown in Fig. 1. Requests arrive at one of the n e-representatives. The overall arrival rate of requests to all e-representatives is λ requests/sec. Assume that this load is uniformly distributed over all e-representatives. A fraction f of the requests arriving at an e-representative has to be sent to the server (e.g., pay, register, etc.). A fraction $(1-f)$ of the requests that do not need to be sent to the server will be satisfied at the e-representative if the data needed to process the request are stored or cached at the e-representative. Assume this happens with probability $(1-p)$. Thus, the fraction of requests that arrive at an e-representative and go to the server is $f+(1-f)\,p$. The average number of visits V_r to an e-representative for each request arriving to the set of e-representatives is $1/n$. Assume that each request requires W units of work, on the average, and that the capacity of the e-representative is C_r units of work/sec. The capacity of the server is C_s units of work/sec. Thus, the average service time, S_r, of a request at an e-representative is W/C_r and the average service time of a request at the server is W/C_s. Therefore,

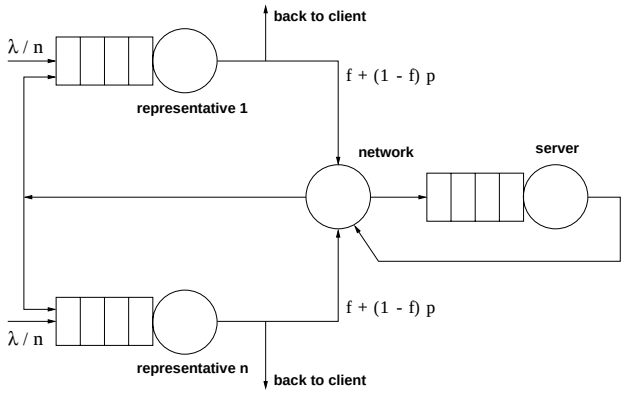

Figure 1. QN for the e-representative case.

the service demand per request (i.e., the product of the average service time by the average number of visits) at the e-representative is $D_r = W/(n \times C_r)$ and at the server is $D_s = W\,[f+(1-f)\,p]/C_s$.

The network is modeled as a delay device with service demand $D_{\rm net} = V_{\rm net} \times S_{\rm net}$ where $V_{\rm net} = f+(1-f)\,p$ and $S_{\rm net}$ is the average service time to send a request and receive a reply from the server. We can then solve the QN model of Fig. 1 to obtain the average response time R_r for the case of e-representatives [13]. The average response time for the case of no e-representatives is

$$R_{\rm no-rep} = \frac{W/C_s}{1-\lambda\,(W/C_s)} + S_{\rm net}. \qquad (1)$$

Fig. 2 shows the average response time as a function of the arrival rate λ for the no-e-representative case and for the case of e-representatives and $f = 0.3$, 0.2, and 0.1. The parameters used for this graph are $W = 10,000$ units of work, $C_r = 100,000$ units of work/sec, $C_s = 4 \times C_r$, $n = 10$, and $S_{\rm net} = 0.48$ sec. The results for this case show that the maximum throughput for the no-e-representative case is 45 requests/sec, while for the case with e-representatives varies from 85 to 95 to 110 requests/sec as f varies from 0.3 to 0.2 and to 0.1. The response time increases as f increases. For low loads and for this set of parameter values, the response time for the no-e-representative case is slightly smaller than the response time for the cases $f = 0.3$ and $f = 0.2$ for $\lambda \leq 15$ requests/sec.

This simple model shows the potential gains obtained with the use of e-representatives. The following sections give a more detailed description of e-representatives and illustrate different protocols that have an impact on the value of f and therefore on the response time perceived by users. A detailed trace-driven simulation is used to assess the relative performance of the various protocols.

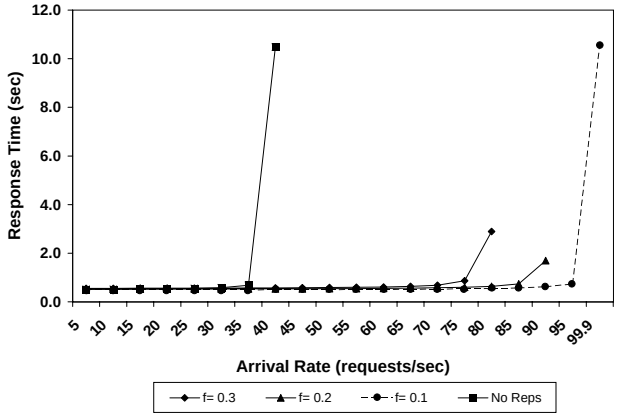

Figure 2. Response time curves.

3 E-commerce Service Issues

This section discusses some issues associated with services provided by e-commerce sites as well as the characteristics of data used to describe the products sold at these sites. Most of our discussion is aimed at services typically found in e-tailers. However, the concepts described here can be easily generalized and adapted to other types of e-businesses such as online trading.

By definition, a representative performs a fraction of the work done by the original e-commerce server. Therefore, representatives must store and/or cache some of the data maintained by the e-commerce site. Some properties of e-commerce data are described as follows.

Retail e-commerce sites maintain information about products (e.g., books, CDs, toys, flowers, cars). The data kept by an e-commerce site to support its operation may be classified into three categories according to its volatility, i.e., the rate at which it is updated:

static. This includes the set of all static pages maintained by a site and the set of all static descriptions about products (e.g., book title, author, publisher, year published, and ISBN).

low volatility. This includes data whose content varies slowly (e.g., the ranking of a book within a virtual bookstore, or the set of reader reviews). Temporary lack of coherence between the originating server and e-representatives can be safely tolerated for low volatility data. This type of content may include cached versions of results of queries to the site's database. Low volatility data does not change as a result of customer interactions with the virtual store, but as a result of actions taken by the site's maintenance staff. *high volatility.* This includes data that changes very rapidly as a result of interactions between customers and the virtual store. An example is product availability. Every time a customer purchases a product, information about its availability has to be updated.

There are some functions commonly present in e-Commerce servers. Without loss of generality, we will use examples from electronic bookstores, which usually provide five functions: Browse, Search, Select, Add to Cart, and Order. The Browse function allows customers to access information filtered by pre-defined criteria such as author and subject. Search queries that may specify author, title, and keywords, provide more flexible access to the catalog items. Browse and Search usually return a list of items that match a given criteria. Customers may get detailed information (e.g., abstract, reviews) about products by selecting them. Once a customer decides to buy a product, it is added to its shopping cart. After adding one or more products to the shopping cart, a customer orders and pays for them. The Browse, Search and Select operations deal with static and low volatility data while the Add to Cart and Order operations handle high volatility data.

E-commerce workloads are composed of sessions. A *session* is a sequence of requests of different types made by a single customer during a single visit to a site [5]. The allowed sequences of requests can be described by a state transition graph called *Customer Behavior Model Graph (CBMG)* [5, 12]. This graph has one node for each possible state (e.g., Home Page, Browse, Search, Select, Add, and Pay) and transitions between these states. A probability is assigned to each transition. Different types of users may be characterized by different CBMGs in terms of the transition probabilities. Thus, workload characterization for e-commerce entails in determining the set of CBMGs that best characterize customer behavior.

An analysis of the CBMG can reveal many interesting properties related to customer behavior. Examples include the average number of times that each operation (e.g., Search, Browse, Add, Pay) is executed per customer visit, and the buy to visit ratio, i.e., the average number of times a purchase is made for each visit to the store.

Key to the efficacy of e-representatives is the reference locality associated with customer requests. We can analyze the reference locality of a stream of requests through frequency distributions of accesses (i.e., requests) to objects. The efficacy of e-representatives is affected by the reference locality of three kinds of requests: browse, search, and select. As described in Sec. 5, we used real logs to generate our simulation workload. As expected, the reference locality varies with the type of request. Search, which comprises the largest number of objects, has the highest reference locality. For instance, 60% of the search queries contain just 1% of the search terms. Although the reference locality varies, it is high for all types of requests, since 10% of the objects are responsible for at least 57% and at most 88% of the accesses.

4 E-Commerce Representatives

E-commerce representatives or simply e-representatives are programs that distribute some of the computation ordinarily performed at the represented server. One possible mechanism to deploy representatives is to have them integrated to proxy cache servers, but other mechanisms to assign requests are feasible, such as DNS Round Robin, in which requests to the store address are resolved to one of several e-representative hosts. In our discussion, we consider the former alternative, in which e-representatives can be integrated to cache servers using redirection, a common feature in those servers (e.g., Squid [6]). Redirection basically rewrites URLs based on regular expressions that describe the transformations to be performed on requests that arrive at the cache server. Thus, whenever requests to a represented server are received by the cache server, they are redirected to the local e-representative. The e-representative parses the requests, and determines which data should be requested to the cache server (and consequently to the server). Although it may not be as fast as requesting data directly to the server for the first request, the existence of reference locality associated with query terms and products (as discussed in Sec. 3) makes this approach efficient.

Four properties are essential for implementing e-representatives: (1) *transparency:* e-representative actions must be transparent to customers, who should not be required to perform any configuration changes; (2) *easy installation:* e-representatives should interface with cache servers through existing mechanisms; (3) *safety:* e-representatives must be as secure as the servers they represent; and (4) *consistency:* information provided by an e-representative should be consistent with the server state.

In this section we discuss how e-commerce operations are performed by e-representatives. We assume that static and low volatility data can be potentially replicated at e-representatives. We discuss below how these data may be organized for the purpose of replication and then describe the interaction between server and e-representatives.

A key issue in replicating e-commerce data is to guarantee that e-representatives can retrieve data from cache efficiently. This goal is accomplished by structuring data about products in two levels: list of products and product info.

Each product has a unique identifier (e.g., ISBN), which is used to locate the product. Lists of products comprise one or more identifiers of products that satisfy some criterion. The semantics of the criteria varies with the e-commerce service; lists of products that result from searches are associated with the query terms; lists of products from browsing are similar, but their criteria are quite static (e.g., subject index). One strategy for addressing these lists is to encode the criteria in the URL of the object that contains the list, making it easy to store and retrieve them as if they were static objects. A list of products changes whenever products that match its criterion are included or excluded from the store selection. However, such changes do not occur very frequently, and standard mechanisms that limit the life time of static WWW objects, such as TTL (Time To Live) fields, are good enough for keeping replicated lists of products consistent with the store selection. This parameter can be dynamically determined by the store each time the representative requests an object.

Product info is divided into two types of objects: product description and product status. Product description comprises static attributes such as title, author, and publisher. Product status, on the other hand, comprises highly volatile data such as pricing and availability.

Replicating the status of a single product may be quite inefficient since it changes frequently and the amount of replicated data is very small. One can improve the efficiency of status replication by grouping, for instance based on the publisher, the status of several products in a single object. Assuming that both product selection and prices change at the same frequency, we can employ the same consistency strategy used for lists of products. Also, both product description and status may be addressed by including the product identifier in the URL of each object. Product availability changes as a result of customer interactions with the site. We describe in Sec. 4.1 four Add to Cart protocols that deal with the update of highly volatile data in different ways, sometimes trading consistency for performance.

Browse is a two-step process. First, after parsing the query, the e-representative requests the product list from the cache server. Then, it requests the product description and status of all products in the list, so that the response to the client can be built. Whenever the objects are not stored in the cache server or are stale, they are requested again from the store server.

Search is similar to Browse, but demands a third step, namely merging the lists of products. Similarly to Browse, Search starts by requesting one product list for each query term. Once the e-representative possesses all needed lists, it merges them, determining the products that satisfy the query. The merge process is performed in accordance to the nature of the query (i.e., conjunctive or disjunctive). Finally, it requests product info and status for the products in the merged list, building the response to the client. The merge step is not necessary to answer one-term queries.

Selection provides detailed information about a given product, based on its description and status. Notice that the response to selection queries is usually fast, since all data has been accessed recently while browsing or searching. Again, if the cache server does not contain a valid copy of the product info or product status, it must request the missing data to the server store. One strategy to make selection more efficient is to store product info temporarily in the e-representative, so that responses to selection requests can be built faster.

4.1 Allocation Protocols

A critical part of e-commerce service distribution is Add to Cart, since it handles highly volatile information, more specifically availability. In practice, whenever a customer adds a product to its cart, the server should allocate the product to the customer, preventing it from being sold during a time interval. This is desirable in businesses that sell items of limited supply, such as seats in a flight. Some e-stores allocate items only when the purchase commits. This works fine in the case of items with almost unlimited supply such as toys or books. It is true that some books (e.g., rare books or limited editions) may be classified as limited supply. In what follows, we assume that a customer must be aware of a product availability when deciding to buy it. Allocations may expire after a pre-defined timeout, be dismissed by the customer, or become actual purchases.

E-representatives allocate products by sending a single message to the server, informing the product and quantity. The allocation is acknowledged by the server in a message that contains a success indicator and the current availability level. As described below, this last piece of information may be used for estimating product availability.

We evaluate four cooperation protocols between an e-representative and the e-commerce server. Each protocol provides a different level of consistency between the data provided by the e-representative and stored in the server. These protocols implement four allocation strategies: eager, speculative, conservative, and optimistic. The strategies differ by when they allocate on the server a product to a customer.

We evaluate the efficiency of cooperative allocation strategies through two criteria: (1) latency reduction, i.e., how much the response time is reduced when compared to the non-cooperative allocation; and (2) allocation utility, that is the percentage of allocations that resulted in a purchase.

Eager Allocation E-representatives employing eager allocation notify the server about a possible allocation as soon as the customer selects a product. By employing this strategy, e-representatives usually receive the acknowledgement for the allocation request before the customer effectively adds the product to the cart, guaranteeing that the product is available, without waiting for an allocation to be processed by the server. On the other hand, this scheme reduces the allocation utility, since many more allocations do not commit. One approach is to watch customer accesses, checking for products that were selected but not added to the cart, and deallocating them. Furthermore, there may be a significant traffic increase, which may affect both store and cache server performance.

Speculative Allocation E-representatives may speculate about a product availability based on its selling history. The representative gets inventory information from the server periodically, and uses this information for estimating whether a product is available or not. The main issue in speculating efficiently is to determine an adequate estimation function, based on frequency of inventory update. Obviously, for the sake of efficiency, these updates may be partial, comprising only popular products.

Intuitively, if a given product is very popular among users of an e-representative, this representative may speculate more aggressively than others. Since each allocation message is acknowledged, we can use allocation responses to also provide inventory information. In practice, when an e-representative receives an Add to Cart request, it estimates the availability of the added product and sends an allocation request to the store. If the product is estimated to be available, the e-rep answers immediately to the requester, otherwise, it waits for the allocation response to confirm the allocation to the customer.

Conservative Allocation E-representatives may also delay the allocation as much as possible, that is, they only send it when the customer adds the product to the cart. The Add to Cart operation resumes only when the e-representative receives the allocation response.This strategy relieves the e-representative from any responsibilities regarding product allocation, but may increase the user latency significantly.

Optimistic Allocation We can optimize the speculative approach by assigning full responsibilities for the allocation to the e-representative. Thus, when it receives an Add to Cart request, it estimates the availability of the added product and only sends an allocation request to the store if this estimation fails, at which point the allocation is performed conservatively.Again, the main issue is determining the estimation function and obtaining inventory updates.

4.2 Design Issues

In this section we discuss some design issues that affect the quality of service provided by e-representatives.

Data Organization Our data organization is based solely on the nature of the various attributes, so that consistency maintenance is implemented easily and efficiently. Browse and Search may be significantly improved by breaking product description into two parts. The first comprises the information needed for generating lists of products, while the second part contains all descriptive data about the product. Furthermore, if latency is usually high between the e-representative and the store server, one may augment lists of products with some description information, avoiding several requests to the descriptions of products that are part of a list.

Prefetching The reference locality in user queries and interests may be exploited through prefetching, that is, an e-representative keeps user profiles and takes advantage of the reference locality associated with queries (and thus products) by loading in advance information about products.

This approach has been successfully employed for representatives of search engines [7, 11]. For instance, if product descriptions expire weekly, the e-representative may prefetch popular products upon expiration. Furthermore, the prefetching may be performed during off-peak periods, making better use of networking and server resources.

Security E-representatives should be carefully coded so that the internals of store servers are not exposed. Since we expect e-stores to take responsibility in implementing and distributing binaries of e-representatives, they may employ security mechanisms such as encrypting all messages between the server and the e-representative, encrypting all files that are replicated in the cache server, and authenticating e-representatives periodically, or replacing keys, avoiding fake e-representatives or transaction snooping. Clearly, these security mechanisms impose a performance penalty. Our simulations do not include the overhead due to security. See [8] for a discussion of the performance impact of security mechanisms in distributed systems.

Privacy Customer privacy is also critical. The use of secure protocols and encryption of all customer data is usually enough for providing reasonable guarantees for customers served by e-representatives.

Computational Demands The cost of the operations performed by the e-representative are similar to the represented operations at the store server and may demand significant computation power as a consequence of heavy request loads. In such cases, it may be necessary to run the representative on a dedicated machine, so that it does not affect the performance of the cache server. In all cases, these costs can still be advantageous, given the better quality of service to customers, bandwidth savings, and representation revenues.

The success of e-representatives depends on the volume of transactions to a store that go through a cache server. The higher the volume, the better the reference locality. E-Malls, for instance, are a good example of a service that may benefit from e-representatives. Stores that sell digital goods, such as software, video, and music can also benefit from e-representatives to reduce download times and bandwidth consumption. Finally, as in any replicating-based approach, the reference locality of the various accesses (in our case, lists of products and product info) is critical for e-representatives to have a good cost-benefit ratio.

5 Performance Simulation Model

In this section we assess the performance gains resulting from the use of e-representatives through trace-driven simulations of an electronic retail store. We first describe the simulator implemented, and then present the results.

The program was developed in C, using the SMPL [10] package, and simulates four entities: client, e-representative, cache, and store. It is designed for simulating a large group of customers interacting with an e-commerce bookstore either through e-representatives or directly. It also simulates the network connections among these entities.

All services described in Sec. 4 are implemented. User sessions, i.e., the sequence of requests submitted by each user, are read from a trace file, one at a time, Each request has three attributes: (1) type, (2) think time, and (3) object list, which is a list of one or more objects that are needed for producing the response. The simulator then schedules the requests individually, according to their attributes. Processing each request comprises several events, each representing a step towards the request completion. For example, a browse request for a client who is using a representative triggers several events in the different entities of the simulation, being processed in turn by the representative, by the cache server, optionally by the remote server, the cache server again, and finally by the representative. The last event associated with every request is always to process the next request in the session, or close the session, if the request just executed was the last. The simulator initiates new sessions at a fixed rate, which is a simulation parameter.

The trace file was generated based on actual logs. The base log [16] comes from a reference site about music equipment, and presents the requests grouped by user, containing not only the requests and their timestamps, but also the referrer URL. This site has two additional types of documents that also appear in the log: (1) the home page, i.e., the entry page to the store; and (2) auxiliary pages, which contain help, credits and other information. We inserted Add to Cart and Pay requests probabilistically. For the simulation results presented in Sec. 5, we derived the conditional probabilities so that 14% of the selected objects are added to the cart and 5% of the sessions where at least one product was added to the cart result in an actual purchase. The base log just indicates that a search request was submitted, but provides neither the query terms nor the characteristics of the search response (e.g., size). We estimated the characteristics of queries based on logs of the Miner Family of Brokers that we analyzed previously [2]. In that case, there is a high reference locality among search terms, since 52% of the terms appear in the stream of queries more than once, and the average hit ratio of terms is 89% for search queries submitted within 24-hour intervals. The search queries comprise, on average, 1.63 words, 52% of the queries contain just one word and the largest query is 12-word long.

In our simulations, we assume that the cache server has infinite capacity, so that objects are evicted from the cache only when they expire. This assumption relies on the fact that secondary storage is not expensive and objects provided by electronic stores are usually small.

Finally, we assume that all allocations are successful, i.e., all books that are added to the cart are available in the bookstore. Although this last premise is unrealistic, unlim-

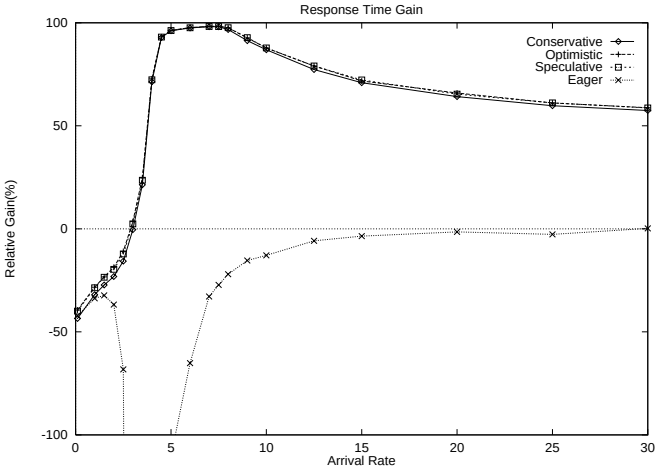

Figure 3. Response Time: relative gains

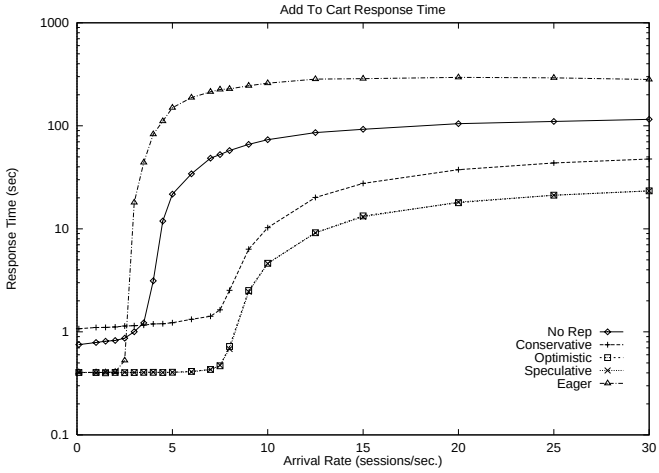

Figure 4. Response time: Add to cart

ited book availability does not disturb our metrics.

We now discuss the results we obtained. First, we compare the performance gains of this retailer operating with representatives, to the same retailer operating without e-representatives. Second, we analyze the trade-offs among the proposed Add to Cart protocols. All experiments were conduced using 10 representatives and one server. The average service time per request for the server is 0.04s, and the service time for each representative, 0.1s. The network bandwidth between the four entities is 256 Kbps, except for the connection between the e-representative and the cache server, which is 10 Mbps.

Figure 3 presents the response time relative gains for each allocation protocol. We observe that e-representatives do not improve performance for a lightly loaded system, i.e., arrival rate < 3 sessions/sec. The eager allocation protocol presented the worst performance (up to 1500% worse), as a consequence of the saturation of the store server by the excessive number of allocation requests. Compared to other allocation protocols, the eager protocol generated 5 times more allocation requests. As the arrival rate increases, the relative performance of the eager protocol also improved, as a consequence of the server saturation in the no-rep configuration, which occurs for arrival rates above 4 sessions/sec.

The three other allocation protocols (i.e., optmistic, speculative, and conservative) provided significant gains in terms of customer response time, as we can observe in Figure 3. The highest performance gain occurs around seven sessions per second. After this peak, the gains decrease slowly as the session arrival rate increases, due to the saturation of the representative's server.

To compare the performance of the allocation protocols, we show the response time of the *Add to Cart* operation as a function of the session arrival rate, as shown in Figure 4. The two worst results were obtained by the the eager approach and by the system without representatives. In both cases, measurements indicate that the e-commerce server got saturated and response times went up. The results of both optimistic and speculative approaches are practically identical, and consistently better than the conservative approach by a factor of 2, which is explained by the time spent waiting for allocation responses. We also confirmed that the use of e-representatives relieved the server from non-critical tasks. For instance, the amount of processing time dedicated to service *Pay* requests increased by 300% at the store server.

Regarding bandwidth consumption, the use of e-representatives reduced the server outbound traffic by a factor of almost 5, from 973 MB (no-rep) to 212 MB (speculative and conservative). The optimistic approach further reduced this traffic by just 1%, but the number of connections between rep and server decreased by 15%. The bandwidth reduction varied from 76.06% (Eager) to 78.32% (Optimistic).

6 Related Work

A few mechanisms have been developed to support quality of service in Web servers. In [1], the authors propose a notion of quality of service by associating priorities with requests based on which document they are requesting, not where they come from. The metric for quality of service is latency in handling the HTTP requests. The priority mechanism is static and the study is restricted to priority schemes for CPU scheduling only. In [15], the authors develop a quality of service model that implements algorithms for scheduling CPU, memory and networking resources. The notion of quality of service is implemented by setting priorities among page requests and is clearly not oriented to

e-commerce sites, which exhibit workloads that are quite different from page requests. An admission control mechanism to improve performance of overloaded Web servers was proposed and analyzed in [4]. The authors introduce the notion of session, consisting of many individual HTTP requests. However, they do not characterize the workload of e-commerce sites, that is composed of typical requests such as browse, search, select, add, and pay. The analysis focuses only on the throughput gains obtained by an admission control mechanism that aims at guaranteeing the completion of any accepted session. The work in [9] proposes a workload characterization for e-commerce servers, where customers follow typical sequences of URLs as they move towards the completion of transactions. However, the authors do not propose any mechanism to guarantee quality of service of e-commerce servers.

There are some proposals for distributing server computation. One approach is to migrate a fraction of the services to the client through applets [19, 18], improving the quality of the services provided. This approach is not applicable to e-commerce services, however, since the reference locality of accesses of a single user does not justify the costs of representing a store. Another approach is to store dynamic objects such as applets in cache servers, and executing them upon a user request to the cache server [3]. This solution is generally applicable, but cache servers may get overloaded by the number of Java Virtual Machines that running, increasing the client response time by up to 400%.

7 Conclusions and Future Work

This paper proposed e-representatives as a strategy for improving e-commerce services by reducing both customer latency and bandwidth consumption. Simulation results show that the use of e-representatives provided response time gains of at least 60% (even for saturated servers) and bandwidth reductions of about 78%.

We are currently implementing e-representatives and plan to investigate several of the issues raised in the paper, such as mechanisms for ensuring security and privacy, prefetching, and alternate data organizations. We are also working on characterizing product availability and customers' reaction to products not being available.

Finally, we also intend to work on adaptive e-representatives, which would support service level agreements, as a function of customer profiles and connectivity status (e.g., if the network is overloaded, the rep may provide less data about the products).

References

[1] J. Almeida, M. Dabu, A. Manikutty, and P. Cao. Providing differentiated levels of service in web content hosting. In *Proc. First Workshop on Internet Server Performance, ACM SIGMETRICS 98*, July 1998.

[2] V. Almeida, W. Meira Jr., V. Ribeiro, and N. Ziviani. Efficiency analysis of e-brokers in the electronic marketplace. In *Proceedings of WWW8*, 1999.

[3] P. Cao, J. Zhang, and K. Beach. Active cache: Caching dynamic contents on the web. In *Proc. of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 373–388, 1998.

[4] L. Cherkasova and P. Phaal. Session based admission control: A mechanism for improving the performance of an overloaded web server, 1998.

[5] D. A. Menascé, V. Almeida, R. Fonseca, and M. A. Mendes. A methodology for workload characterization of e-commerce sites. In *Proc. 1999 ACM Conference on Electronic Commerce*, November 1999.

[6] N. L. for Applied Network Research. Squid Internet Object Cache. `http://squid.nlanr.net/Squid/`.

[7] H. H. Foxwell and D. A. Menascé. Prefetching results of web searches. In *Proc. of the 1998 Computer Measurement Group Conference*, Anaheim, CA, December 6-11 1998.

[8] A. Harbitter and D. A. Menascé. Performance issues in large distributed system security. In *Proc. of the 1998 Computer Measurement Group Conference*, Anaheim, CA, December 6-11 1998.

[9] D. Krishnamurthy and J. Rolia. Predicting the performance of an e-commerce server: Those mean percentiles. In *Proc. First Workshop on Internet Server Performance, ACM SIGMETRICS 98*, July 1998.

[10] M. MacDougall. *Simulating Computer Systems: Techniques and Tools*. The MIT Press, 1987.

[11] W. Meira Jr., M. Cesário, R. Fonseca, and N. Ziviani. Integrating www caches and search engines. Global Internet 1999, Rio de Janeiro, RJ, December, 1999.

[12] D. A. Menascé, V. Almeida, R. Fonseca, and M. A. Mendes. Resource management policies for e-commerce servers. In *Proc. Second Workshop on Internet Server Performance*, Atlanta, GA, May 1st 1999. in conjunction with ACM SIGMETRICS 99/FCRC.

[13] D. A. Menascé and V. A. F. Almeida. *Capacity Planning for Web Performance: metrics, models, and methods*. Prentice Hall, Upper Saddle River, 1998.

[14] J. Nielsen, 1999. http://www.useit.com/alertbox/990207.html.

[15] R. Pandey, J. Barnes, and R. Olsson. Supporting quality of service in http servers. In *Proc. Seventeenth Annual SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 1998.

[16] M. Perkowitz and O. Etzioni. Adaptative web sites: download. http://www.cs.washington.edu/homes/map/adaptive/download.html.

[17] P.Krishnam and B. Sugla. Utility of co-operating web proxy caches. In `Proceedings of WWW7`, 1998.

[18] R. Vingralek, Y. Breitbart, M. Sayal, and P. Scheuermann. Web++: A system for fast and reliable web service. In *Proceedings of the 1999 USENIX Annual Technical Conference*, June 1999.

[19] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler. Using smart clients to build scalable services. In *Proceedings of the 1997 USENIX Technical Conference*, Jan. 1997.