

Assessing the impact of distribution on e-business services

B. Coutinho G. Teodoro T. Tavares R. Pinto D. Nogueira W. Meira Jr. D. Guedes

Department of Computer Science
Federal University of Minas Gerais
Belo Horizonte, Brazil

{coutinho,george,ttavares,robert,diego,meira,dorgival}@dcc.ufmg.br

ABSTRACT

The success of Electronic-Commerce (e-commerce) sites is directly related to their ability to serve a large number of clients concurrently. As the number of clients increase, servers must be able to keep up with the resulting load. The issue of scalability is often dealt within this realm by distributing the site services over a group of cooperating servers. Considering the complex architecture of e-commerce sites, understanding the impact of service distribution on the overall performance is a difficult task. In this paper we propose a performance evaluation strategy for distributed e-commerce sites and present the analysis of an actual distributed e-commerce server under realistic loads as we change the number of servers being employed. Results show that the throughput of the whole system increases with the number of servers, but not linearly, as a consequence of the server cooperation overheads and the load imbalance that is inherent to the workload and data reference locality, justifying the development of distribution mechanisms that are more elaborated than those used in static content Web sites.

1. INTRODUCTION

The growth of the Internet in the recent past has been widely linked to the growth of the World Wide Web (WWW). A lot of that growth can be credited to the expansion of electronic commerce (e-commerce) services. E-commerce is definitely one of the driving forces behind the expansion of the network and the increase in the number of users and companies that use it on a daily basis.

In the WWW in general and in e-commerce sites in particular, success is measured by the number of users accessing the site and the ability of the site to satisfy users requests promptly. That means that a store, to be successful, must be able to get the attention of a large number of Internet users and keep them satisfied. The problem is that increasing success is directly related to increasing number of concurrent users and, therefore, increasing demands over the company's server(s). To be able to continue to be a success a popular e-commerce site must be able to accommodate the increasing load gracefully.

Because of that, scalability has become a major concern of every e-commerce site in the WWW nowadays. However, improving the capacity of an individual server can only go so far, given technological limitations related to CPU and memory speed, among other factors [8]. Distribution of the

users requests over a group of servers is a strategy to achieve scalability that is not limited by individual machine limits. There has been a lot of work in this direction for static WWW content, defining different distribution techniques and identifying the main performance bottlenecks of such systems. A good survey of the work in the area of distributed web servers has been compiled by Cardellini *et al.* [2].

One challenge faced by distributed servers, however, is that speed-up (the increase in performance due to the increase in the number of servers) is not linear. The distribution of requests over a number of servers impose new processing demands to guarantee the consistency of information among them. For example, if two clients, accessing different servers, try to purchase a certain product, the two servers must guarantee they are not selling the same item twice. This extra overhead may grow to a point where adding servers may in fact make overall performance even worse.

Understanding the performance of a large, distributed, e-commerce site is quite a complex task, specially if we consider that the architecture of such a system is organized in multiple layers of software and servers. Service distribution brings in a new dimension to the problem, which has not been extensively addressed in the literature so far. Most work on the performance analysis for e-commerce servers has been done in the context of individual servers [1, 3].

Another source of overhead, besides consistency problems, is the management of the distribution of requests in the presence of session state. HTTP is a stateless protocol, so each request for a page is independent of all others and may be directed to any host in a distributed server cluster. On the other hand, in e-commerce sites, the interaction of the client with the site creates a certain amount of state information: the products added to a shopping cart, the user identification, for example. In this case request distribution must consider that information also, what adds more complexity to the distributed server. This paper presents an analysis of the behavior of such a distributed server to identify the impact of consistency and state maintenance overheads on its overall performance.

The rest of this paper is organized as follows: Section 2 introduces the architecture and major components of an e-business service, and is followed by a discussion of the challenges faced when distributing such services, in Section 3.

After that, Section 4 discusses our approach to evaluate e-commerce services and Section 5 presents our major results. After that, Section 6 concludes with some final thoughts and suggestions for future work.

2. E-BUSINESS SERVICES

In order to understand the details of an e-commerce site we must understand the relations between the various entities involved and the architecture of the server site as a whole. The following Sections address those two issues.

2.1 e-Business entities

There are basically three main entities to be considered in e-business services: products, clients and sessions.

The first entity represents commercialized products. There are basically two types of data about them: static and dynamic. Static data comprises information that is not affected by the services provided by the e-Business server, such as the description of a book, or the characteristics of a TV. Dynamic data, on the other hand, contains all pieces of information that are affected by the operations performed while providing services, such as the number of books in stock, or the address of a buyer.

The second entity represents the customers. Again, the server records static and dynamic data. In this case, information such as name and address are static, while other elements, such as the content of the shopping cart, are dynamic.

The identification of static and dynamic elements is crucial for the distribution of the e-business service. While static data can be easily replicated among the elements of a server cluster, each dynamic piece of data must be properly identified and any access to it must be properly controlled to avoid inconsistencies.

The third entity is the relation between the first two and represents the interaction between customers and products. That interaction is usually represented as the user session, built by the server in response to user requests. Each session combines references to static, as well as dynamic, data of a client (or group of clients, in some cases) and some number of products.

We may distinguish some essential operations in this scenario during the interaction of a client with the server site: user requests have to be received and parsed to determine which session they refer to and which operations are needed; session data must be retrieved, as well as customer data, if it is the case, so that operations are performed in the right context; if the operation involves some products (the common case) product data must also be retrieved; once all information needed is available the request must be processed, which may affect dynamic data; the request results must be saved and session state updated; finally, the response page must be built based on the request results and site presentation rules and sent back to the user's browser for presentation. Those operations require different capabilities to be performed, and may in fact be executed by different elements in an e-commerce server.

2.2 e-Business architecture

Considering the operations performed by an e-business service and the different types of information that must be handled, the structure of a server is usually divided in three major components, the WWW server, the transaction server and the database [10, 5]. That organization is illustrated in Figure 1.

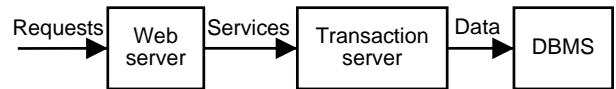


Figure 1: E-business server architecture

The WWW server is responsible for the service presentation, receiving user requests and parsing them. Once that is done, that server must manage all operations. Requests for static data are served directly, while requests involving dynamic data are forwarded to the next level, the transaction server. It is also the responsibility of the WWW server to deliver the page composed by the transaction server as a result of a dynamic request.

The transaction server (also known as the application server, or virtual store) is the heart of the e-commerce service. It is this component that must implement the logic of the business: how to handle client searches for products, how to present the products, how to complete a sale. It must keep track of the user sessions, so it can control the shopping cart, and provide site personalization, for example. Once it receives a command for a certain service from the WWW server, it must implement the logic of the operation. That may include accessing the database server to complete the request and build the response page, which will be handled back to the WWW server.

Most of the information related to the store operation must be kept consistent across sessions and in stable storage to guarantee proper operation over time. That is the responsibility of the database management system, the third element of the server. It holds product and client information, number of items available and other data. It receives commands from the transaction server and returns data as a result.

In centralized, custom-built e-business sites the three elements may be implemented as a monolithic application for performance reasons. However, in most cases the service is implemented using separate processes for each of the three elements. A first approach to improve performance by distribution in this case is simply to use separate machines for each of the three elements. That, although simple to implement, has limited gains, given its intrinsically limited nature. Really scalable solutions must distribute functionalities over a larger number of machines, which will require replicating processing elements and distributing requests properly.

3. DISTRIBUTING E-BUSINESS SERVICES

In this section we discuss how e-Business services may be distributed. The goal of the distribution is to make possible for the various servers employed to satisfy requests as efficiently as possible. The main challenge in this case is how to make the data necessary for answering each request available in the proper server.

As we mentioned in Section 2, we distinguish three types of data that are usually handled by e-Business servers. Product-related data is the first type of data. We further divide product-related data into two categories: dynamic and static. Dynamic data comprise all information that may change as a consequence of performing transactions, such as inventory. Static data are usually attributes of the good or service being marketed, such as description and manufacturer. The second type of data is the customer-related data, which may be also divided into static and dynamic. Interaction-related data is the last type, being almost always dynamic. Although all data may be stored in a database, it is better to maintain replicas, or even the data themselves in an application server, in order to improve the response time of the server.

Distributing static data may be viewed as a caching problem. Each server has a limited amount of storage space and managing the overall storage space should take into account factors such as inter-server communication costs, reference locality, storage costs in the server. Considering those factors, there are several caching management strategies that may be employed. If storage is cheap, a totally replicated cache may reduce access times overall, since all hosts will have local copies of the cached data. On the other hand, if storage is expensive, mutually exclusive caches will make the best use of it, since every cached object will occupy space in only one server's cache. However, such solution may have to handle a lot of inter-server communication in case requests reach servers which do not hold the copy to some required data. Many intermediary solutions are also possible, where some level of replication is allowed to handle very popular elements, or to reduce inter-server communication in some cases [9].

Distributing dynamic data is quite more complicated in the sense that coherence becomes an issue, that is, if a piece of dynamic data is stored in two servers, there must be a way to guarantee that either server will be notified of changes performed by the other one. This problem is not novel and distributed databases have been working on addressing it efficiently for some time. However, it is interesting to notice that in most e-Business services, most of the workload imposed by customers involves static data[6]. That is due to access to information such as product descriptions and images, for example. For that kind of data, replicated caches certainly improve response times [4].

In this paper we assess the impact of distribution of e-Business servers. In particular, we evaluate a very common approach employed by many servers: replicated caches for static data and non-replicated dynamic data. Although restrictive, this approach is simple in terms of implementation and does not impose severe coherence and management requirements on the system. Although replicated caches are very simple to implement, since they do not demand cache cooperation mechanisms, their efficiency tends to be lower as we increase the number of distributed servers for a given workload. The decrease in efficiency comes as a result of the smaller number of requests that each server receives, so less information is embedded in the system. As a result, we may have a larger number of requests to the database, causing an impact on the overall performance of the server.

On the other hand, not replicating dynamic data may result in load imbalance as a consequence of the characteristics of the workload. The problem arises because the session data does not migrate among servers, that is, once a given server starts answering requests for a session, it keeps doing it until the session ends.

4. PERFORMANCE EVALUATION

The basis of our performance evaluation is the set of metrics used. We distinguish two sets. The first set quantifies the overall server performance, employing the traditional server response time and throughput. These metrics are used to evaluate the gain in terms of scalability associated with the addition of processors, and to compare distribution mechanisms.

The second set of metrics provide detailed measures of the application server. These metrics quantify the amount of elapsed time spent performing the various tasks associated with satisfying requests. These metrics are defined based on two criteria. The first criterion is the request processing phase. The processing of a request is divided into five phases: reception, request parsing, customer session handling, request handling, and request response generation. The second criterion is the profile category. We distinguish six orthogonal performance categories in this work:

Computation: It is the amount of time spent performing CPU-bound processing, such as building a response page.

Communication: It is the amount of time associated with the execution of communication-related system calls, such as `send` and `recv`.

Slave Wait: It is the amount of time that a worker waits for a response from another server. This response is usually used for satisfying the requests.

Database Wait: It is the amount of time that a server waits for a response from the database server, such as when the application server retrieves product data from the database sever.

Request acceptance: It is the amount of time for performing the `accept` system call.

Contention: It is the amount of time that the server is blocked, while waiting for a novel connection to handle.

These metrics may be used to diagnose the sources of performance degradation. However, we should also pay attention to the overall server situation. For instance, a high value in the overall server load usually result in larger elapsed computation times, as a consequence of multiprogramming.

There are two main features that characterize our performance profiling strategy. The first is that it allows performance analysis at multiples levels of detail, as we present in the next section. The second feature is its extensibility, since defining new categories and processing phases may be done easily.

5. CASE STUDY: E-STORE

In this section we present a performance evaluation of an e-store, more specifically a bookstore, which is a very popular e-business server. We start by describing the experimental environment on which we performed the various experiments. We then present a characterization of the workload to which our server is submitted. We analyze three types of experimental results. First we analyze the overall performance achieved by the various server configurations under typical customer workload and isolated operations. We then analyze two aspects of the server distribution in the last two subsections: load balancing and impact of replication.

5.1 Experimental environment

Our e-commerce server is three-tier architecture, as discussed in Section 2, comprising a Web server (Apache 1.3.20), a thread-based application server implemented by us, and a database system (MySQL 3.23.51). Our e-commerce server answers six types of requests: *home*, *browse*, *search*, *select*, *add*, and *pay*. *Home* is just an entry page of the store, being static. *Search* and *browse* allow the customer to locate a product of interest. *Select* verifies the details of a given product, which may be reserved through the *add* request and later purchased using the *pay* request. We used the Httperf [7] client to generate the workload described in Section 5.2.

The experiments discussed in the Sections that follow we performed in a cluster of 16 Pentium 3, 800 MHz with 512 MB RAM connected through a switched Fast Ethernet network. All experiments employed four dedicated machines running Httperf for generating the various workloads. One machine is always dedicated to the database server. The remaining machines used in the experiments run both the Web server and the application server each. This strategy was employed because the load on the Web server is marginal, since all pages are generated by the application server, and putting both Web and application servers in the same machine reduce the communication latency between them.

As mentioned in Section 3, the distribution strategy of the application server is based on assigning the management of each product to a single server and keeping session-related information on the server that receives the first request of the session. The sessions are assigned to servers in a round-robin fashion.

The implementation of the application servers is thread-based and comprises two sets of threads: workers and slaves. Workers are responsible for answering requests submitted by clients. While answering these requests, the worker may query the database server or another application server. Slaves are responsible for handling other servers data requests, and may also query the database server whenever necessary.

5.2 Workload characterization

In this section we present a workload characterization of the behavior of customers while using an e-commerce store. Understanding the workload associated with customers is important because they allow us to estimate the amount of work to be performed by the servers and also the impact that the distribution may have on the load on each server.

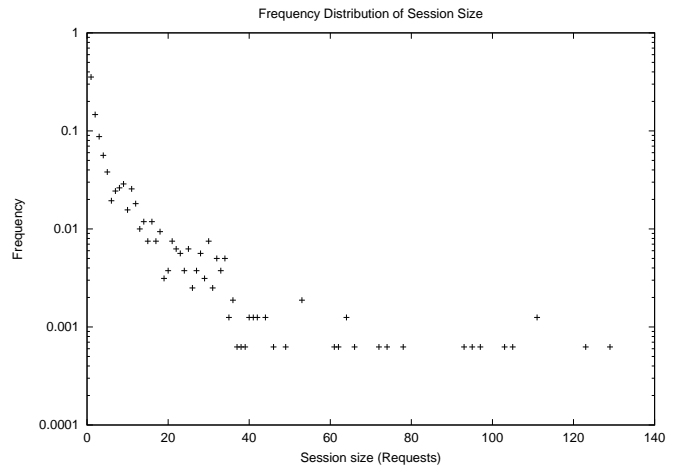


Figure 2: Frequency distribution of session sizes

Our workload is based on 400 logged sessions comprising 8963 requests to an actual e-store. The most frequent requests are *search* and *select* (44.2% each), followed by *browse* (6.27%), *home* (4.78%), *add* (0.5%), and *pay* (0.01%). Figure 2 shows the request frequency distribution per session.

We also characterized the popularity of the parameters for each type of request. Figures 3 and 4 present the frequency distribution of parameters for *search* and *select* requests in the workload, which are the query terms and the products of interest, respectively. Each graph has four plots, and each plot is associated with one workload generator. The graphs were generated by sorting the parameters in non-decreasing order, and plotting their frequency. In both cases and for all workloads we can clearly see that the distributions are very concentrated, that is, very few terms (or products) account for most of the occurrences, indicating that replication strategies should perform well.

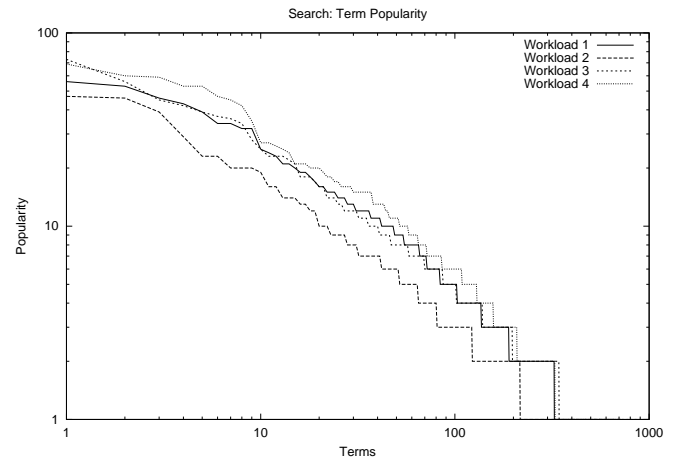


Figure 3: Search: term popularity

5.3 Overall performance

In this section we analyze the performance of the e-commerce server as a whole. Our two basic metrics are request throughput and latency.

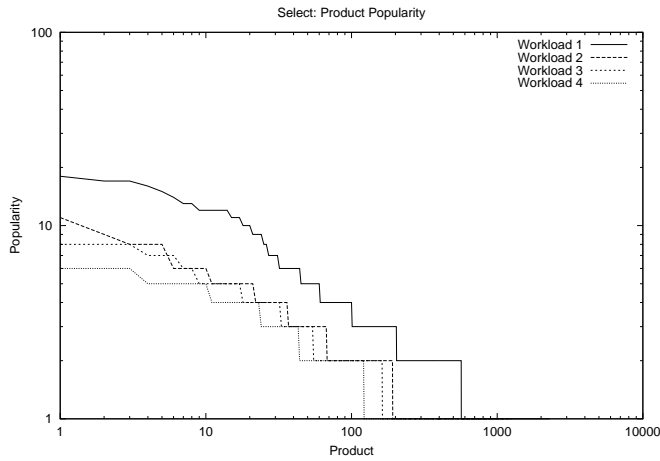


Figure 4: Select: product popularity

In Figure 5 we can observe the throughput of three server configurations (1, 2, and 4 application servers). We can observe that the maximum throughput achieved increases with the number of processors, as expected, from 18 requests per second when using one application server to 33 requests per second when using four application servers. As expected, a larger number of application servers result in higher throughput, although the improvement is not linear with the number of servers employed. The reasons for this limited gains are discussed next.

We start by analyzing the throughput provided by the server configurations under homogeneous workloads, that is, workloads that contain just one type of request. Figures 6, 7, and 8 show the throughput achieved for *select*, *search* and *add* workloads, respectively. In all cases we observe the same tradeoffs we observed in the original workload, but the throughput for the *add* workload is lower, as a consequence of the transactional requirements of this type of request. Furthermore, it is remarkable that although *search* and *select* present similar bounds for four application servers, the bounds for two application servers are significantly different, this behavior is explained by the larger amount of information usually handled by the *select* request.

In order to better understand the tradeoffs associated with the variable number of application servers, we compare two configurations, employing two and four servers, respectively. For each configuration, we measured, for each request type, its number of occurrences, average server response time (i.e., the elapsed time between a request arrives and its response is sent), and the relative standard deviation of the server response time.

We further divide the requests into cooperative and non-cooperative. Cooperative requests are characterized by information exchange between servers. Non-cooperative requests are answered just by the server that received the request. Informing product inventory is an example of information exchange. The performance profiles for both workers and slaves for configurations employing two and four appli-

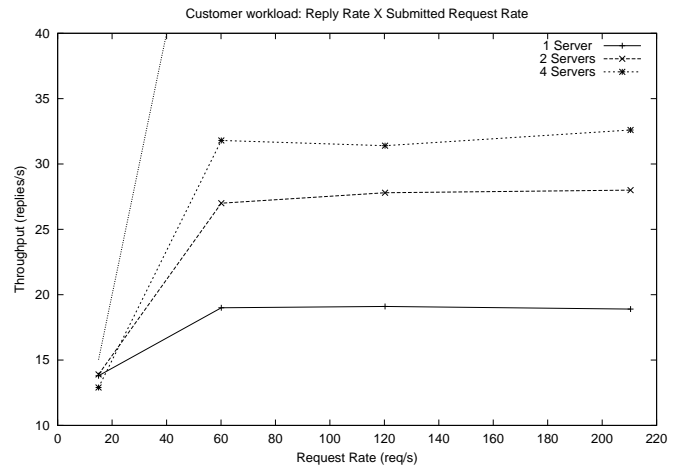


Figure 5: Throughput: customer workload

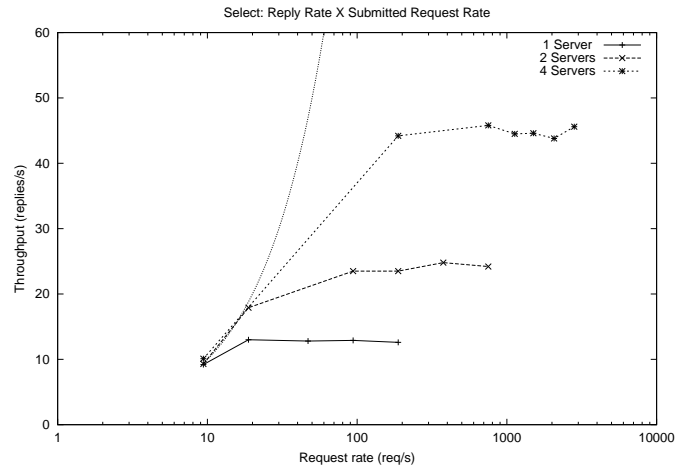


Figure 6: Throughput: select-based workload

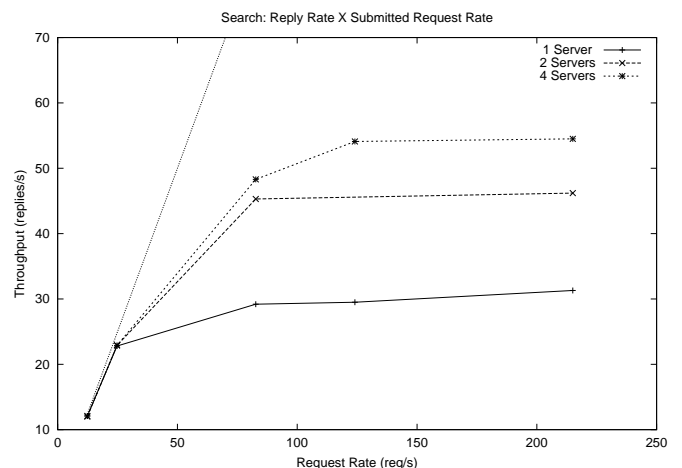


Figure 7: Throughput: search-based workload

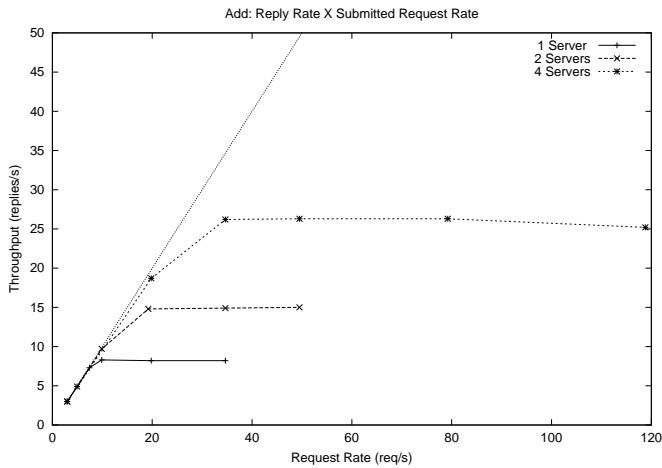


Figure 8: Throughput: add-to-cart workload

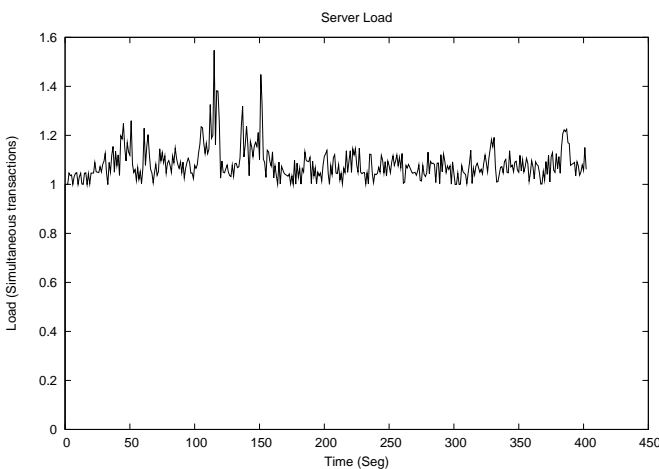


Figure 9: Server load

cation servers are presented in the Tables 1, 2, 3, and 4. The first observation is that cooperation increases significantly the server response time, as expected. The second observation is that the variance of the various response times is very significant, reaching up to 1400%. We can better illustrate this variability through the graph in Figure 9, which presents the number of simultaneous requests being handled by a server. At least in our experiments we could find no correlation between variance and the nature of the request. It is quite surprising that an increase in the number of application servers usually results in higher response times. This increase is explained by the larger number of requests answered by the slave threads. By analyzing Tables 2 and 4, we can observe that that this number increased by 60% with the addition of two servers, from 3127 (two application servers) to 5077 requests (four application servers), increasing the multiprogramming level of the server and thus explaining the larger response time.

The performance profiles for workers and slaves just discussed show that cooperation results in significant increase in the server response time. By analyzing the detailed per-

#Reqs	Request	Type	Response Time (Seg)	Relative Std. Dev.
1414	Home	Ncoop	0.027350	3.79
1280	Search	Ncoop	0.085953	14.14
1223	Search	Coop	0.307138	0.96
431	Browse	Ncoop	0.029934	2.73
423	Browse	Coop	0.062283	2.13
1815	Select	Ncoop	0.375374	1.09
1853	Select	Coop	0.409818	1.74
27	Add	Ncoop	0.043666	1.42
231	Add	Coop	0.806545	0.83
3	Pay	Ncoop	0.001337	0.01

Table 1: Worker performance profile for two application servers

#Reqs	Request	Type	Response Time (Seg)	Relative Std. Dev.
1223	Search	Page	0.000961	3.74
423	Browse	Page	0.001152	6.08
1854	Select	Page	0.297530	1.08
27	Add	Prod	0.000224	0.15

Table 2: Slave performance profile for two application servers

formance profiles, we found that non-cooperative requests should be divided into two categories. The first category comprises requests that were answered completely by the application server, using data replicated locally. These requests are usually answered faster than the other categories of requests, since they do not demand any kind of communication. The second category includes requests that resulted in a query to the database server to gather the necessary data. In these cases, the delay associated with gathering data from the database may be more than 90% of the server response time. In the case of cooperative requests, the delay for exchanging information among servers is the most significant measure for the server response time.

In summary, the limited scalability of distributed e-commerce servers is explained by two factors: cooperation overhead and load imbalance, caused by the high variance among server response times. In the sections that follow, we discuss each of these factors.

5.4 Impact of replication

In this section we evaluate the impact of replication on the performance of the distributed e-commerce server. The gains provided by caching data from the database are a function of their popularity. The analysis presented in the Workload Characterization (Section 5.2) shows that replication should be quite effective considering the high concentration of the popularity. Thus, we want not only to quantify the impact of requesting data to the database server but also to verify the impact of popularity of data on the gains provided by this replication.

We then evaluated three server configurations (1, 2, and 4 application servers) where the caches are initially empty. Al-

#Reqs	Request	Type	Response Time (Seg)	Relative Std. Dev.
1400	Home	Ncoop	0.060519	3.24
733	Search	Ncoop	0.089634	2.15
1733	Search	Coop	0.372940	1.30
240	Browse	Ncoop	0.054810	2.65
604	Browse	Coop	0.148449	3.92
949	Select	Ncoop	0.274108	1.27
2746	Select	Coop	0.426430	1.40
213	Add	Ncoop	0.773568	0.80
40	Add	Coop	0.068545	1.84
3	Pay	Ncoop	0.001362	0.11
1	Pay	Coop	0.001908	0.00

Table 3: Worker performance profile for four application servers

#Reqs	Request	Type	Response Time (Seg)	Relative Std. Dev.
1695	Search	Page	0.001858	6.71
604	Browse	Page	0.004120	6.84
2747	Select	Page	0.304772	1.21
40	Add	Prod	0.000221	0.16
1	Pay	Prod	0.000189	0.00

Table 4: Slave performance profile for four application servers

though this evaluation does not show the effective potential of replication in continuous operation, it allows us to evaluate its impact. Table 5 shows the average server response times for categories of requests of type *search* considering two criteria: cooperation (Coop) or not (NCoop), and requesting data from the database (DB) or not (NDB). We can observe that the use of replication without the need of cooperation resulted in the best server response times. Furthermore, requests satisfied through cooperation usually outperform those that access the database, as expected. Finally, there is an increase in the server response time of all categories as we increase the number of application servers. This increase is explained by the increasing amount of cooperation that replaces database access. In this case, the locality of reference per server decreases (and thus the number of non-cooperative cached requests), and the server response times increase because of the impact of cooperation. Notice that when two servers cooperate, there may be still accesses to the database by the slave that cooperates, which are not shown in Table 5.

In summary, replication is an effective strategy for improving the performance of e-commerce servers, but it should also be implemented cooperatively, that is, the most popular data should be replicated in all servers, quite popular data should be replicated in several servers, avoiding massive cooperation as we observed in the results just discussed. Although this strategy would reduce the overall storage capacity of the caches, the resulting gains seem to be greater, as a consequence of the concentration observed in terms of popularity.

Appl. Servers	#Reqs	Category	Response Time (Seg.)
1	3856 (59%)	NCoop+NDB	0.091386
1	2691 (41%)	NCoop+DB	0.803137
2	2096 (32%)	NCoop+NDB	0.212981
2	3107 (47%)	Coop+NDB	0.673295
2	1344 (21%)	NCoop+DB	1.147184
4	1289 (20%)	NCoop+NDB	0.450686
4	4618 (70%)	Coop+NDB	1.077887
4	640 (10%)	NCoop+DB	1.350085

Table 5: Search: server response time

5.5 Quantifying load balancing

In this section we analyze how balanced is the workload among servers. This analysis has two main goals. The first is to quantify the amount of load imbalance and its impact. The second is to determine whether the variance in terms of response time is a consequence of the load imbalance among servers.

We analyzed the server response times and remote service response times for the request *select* in a four-server configuration. The measures for each server are shown in Table 6. More specifically, we present the number of requests in each category, the percentage of this number compared to all requests in the same category and the average server response time per category. Besides the criteria of cooperation and replication, we also analyzed the type of slave work performed, that is, whether the information requested by a server results in a database access by another server. The criterion used for dividing slave work is whether it accesses the database server (SlaveDB) or not (SlaveNDB). The first observation is that the imbalance among servers is clear and significative. Despite the variance in terms of response times, the number of requests handled by each server also varies significantly. Furthermore, remote services seem to be very significative in creating imbalance. Although server 1 answered the largest number of requests, most of them are cooperative, and resulted in load in the remaining servers, mostly servers 3 and 4.

These results indicate that load balancing policies should take into account not only the number of requests, but also their nature. Considering the variability in terms of server response time and popularity, designing and calibrating this function seems to be an interesting problem to be solved in order to effectively exploit computational resources of a distributed e-commerce server.

6. CONCLUSIONS AND FUTURE WORK

In this paper we presented a quantitative assessment of the performance of distributed e-commerce servers that employ multiple application servers. We propose a performance evaluation strategy and apply this strategy to an electronic store. The results show that, as expected, the performance of the server as a whole improves as we add application servers, but not linearly. Furthermore, we also observed that the server response time also increases with the number of application servers being used. This increase is explained

Category	Application Servers							
	1		2		3		4	
	#Reqs	Response Time(seg)	#Reqs	Response Time(seg)	#Reqs	Response Time(seg)	#Reqs	Response Time(seg)
NCoop+DB	106 (45%)	0.239508	51 (22%)	0.501437	41 (18%)	0.469656	35 (15%)	0.778161
Coop+NDB	2562 (46%)	0.330529	1232 (22%)	0.602249	1013 (18%)	0.656515	813 (14%)	0.829827
NCoop+NDB	777 (46%)	0.339537	334 (20%)	0.581360	338 (20%)	0.708408	234 (14%)	0.805553
SlaveDB	136 (20%)	0.103821	182 (26%)	0.143916	171 (24%)	0.135010	211 (30%)	0.154362
SlaveNDB	870 (18%)	0.099711	1200 (24%)	0.101625	1408 (28%)	0.105340	1445 (30%)	0.094838
Total	4451		2999		2971		2738	

Table 6: Performance profile for select

by the larger cooperation among server and the associated costs. By using the evaluation strategy proposed, we were able to understand the reasons behind such behavior, in particular the cooperation overheads, the impact of reference locality on data replication, and also quantified the load imbalance that is caused by both workload characteristics and reference locality to the various types of data that are handled by the server. In summary, distribution is a good scalability strategy, but achieving high efficiency while using distributed servers demands load balancing techniques and the use of faster communication mechanisms that reduce the communication latency.

One immediate future work direction is to verify whether the behavior observed would be the same in larger configurations. Further, evaluating these tradeoffs for other types of e-commerce servers, such as auction servers, would demonstrate the generality of the approach. Furthermore, the work presented in this paper points to several future work directions. The first direction is to investigate the benefits of a cooperative cache among servers, which would allow a finer grain control on the replication level of each cache entry, that is, popular pieces of information may be replicated in more than one cache, while not very popular would be replicated in just one server. Another direction is to devise and test load balancing strategies that migrate sessions so that all servers have the same workload.

7. REFERENCES

- [1] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Bottleneck characterization of dynamic web site benchmarks. In *Third IBM CAS Conference*. IBM, feb 2002.
- [2] V. Cardellini, E. Casalicchio, M. Colajanni, and P. Yu. The state of the art in locally distributed web-server systems. *ACM Computing Surveys*, 34(2):1–49, jun 2002.
- [3] E. Cecchet, A. Chanda, S. Elnikety, J. Marguerite, and W. Zwaenepoel. A comparison of software architectures for e-business applications. Technical Report TR02-389, Rice University, jan 2002.
- [4] S. Gadde, J. S. Chase, and M. Rabinovich. Web caching and content distribution: a view from the interior. *Computer Communications*, 24(2):222–231, 2001.
- [5] D. McDavid. A standard for business architecture description. *IBM Systems Journal*, 38(1), 1999.
- [6] W. Meira Jr., D. Menascé, V. Almeida, and R. Fonseca. E-representative: a scalability scheme for e-commerce. In *Proceedings of the Second International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems (WECWIS'00)*, pages 168–175, June 2000.
- [7] D. Mosberger and T. Jin. httpperf: A tool for measuring web server performance. In *Proc. of the Internet Server Performance Workshop*, pages 59–67, June 1998.
- [8] E. Nahum, T. Barzilai, and D. D. Kandlur. Performance issues in www servers. *IEEE/ACM Transactions on Networking (TON)*, 10(1):2–11, 2002.
- [9] G. Pierre, I. Kuz, M. van Steen, and A. S. Tanenbaum. Differentiated strategies for replicating Web documents. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, 2000.
- [10] C. Wrigley. Design criteria for electronic market servers. *Electronic Markets*, 7(4), Dec 1997.