

Load Balancing on Stateful Clustered Web Servers

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

¹ Department of Computer Science
Universidade Federal de Minas Gerais
Belo Horizonte MG Brazil 31270-010
{george,ttavares,coutinho,meira,dorgival}@dcc.ufmg.br

Abstract—

One of the main challenges to the wide use of the Internet is the scalability of the servers, that is, their ability to handle the increasing demand. Scalability in stateful servers, which comprise e-Commerce and other transaction-oriented servers, is even more difficult, since it is necessary to keep transaction data across requests from the same user. One common strategy for achieving scalability is to employ clustered servers, where the load is distributed among the various servers. However, as a consequence of the workload characteristics and the need of maintaining data coherent among the servers that compose the cluster, load imbalance arise among servers, reducing the efficiency of the server as a whole. In this paper we propose and evaluate a strategy for load balancing in stateful clustered servers. Our strategy is based on control theory and allowed significant gains over configurations that do not employ the load balancing strategy, reducing the response time in up to 50% and increasing the throughput in up to 16%.

I. 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 is not always effective, given technological limitations related to CPU and memory speed, among other factors [15]. 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.* [4].

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.

Designing and implementing scalable and efficient distributed services 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 in the area of performance analysis for e-commerce servers has been done in the context of individual servers [1, 5].

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. All these characteristics may result in load imbalance among the servers are responsible for handling requests. Devise, implement, and evaluate load balancing strategies is the subject of this paper. We divide the load balancing problem into two sub-problems: (1) load migration, and (2) balancing strategies. In order to solve the first sub-problem we have to define units of work and how they are exchanged among servers. Solutions to the second sub-problem demand the design of load balancing strategies that also include the definition of imbalance metrics.

This paper proposes and evaluates experimentally a load balancing strategy for transaction clustered servers. The strategy is evaluated through an actual e-store under realist

workload.

The rest of this paper is organized as follows: Section II introduces the architecture and major components of stateful Web servers, and is followed by a discussion of the issues faced when balancing the load in such servers in Sections III and IV. After that, Section V discusses a common stateful server architecture, which is an e-store. Section VI characterizes the workload that is submitted to e-stores and Section VII presents our major results. After that, Section VIII concludes with some final thoughts and suggestions for future work.

II. STATEFUL CLUSTERED WEB SERVERS

In order to understand the details of an stateful server we must understand the relations between the various entities involved and the architecture of the server site as a whole. In this section we address these issues. Without loss of generality, we may discuss the details of stateful servers by analyzing a typical architecture: e-commerce servers. There are basically three main entities to be considered in e-business services: products, clients and sessions.

The first entity represents the 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.

As we aforementioned, we distinguish three entities and also 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 [16].

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[13]. 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 [9].

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 processing elements in an e-commerce server.

In the next two sections we discuss how we migrate data (and thus the associated load) and how we decide what to migrate, which is defined by a load balancing strategy.

III. LOAD MIGRATION

When it becomes necessary for a cluster to migrate load among its machines (for example, in order to improve load balancing) that is achieved by moving some of the data to a new machine, transferring the associated processing demands with them. Considering the types of entities represented in e-business services the likely candidates are sessions and products, since they concentrate the dynamic elements which demand processing. Between them, sessions are the best candidates for migration, since they concentrate most of the processing due to their representation of the application logic. Accesses to products are usually simple and are included in the processing of session requests.

Therefore, this work uses sessions to migrate load from one server to another. Using devices like client-aware level-7 switches [4], session migration may be done in a way transparent to the clients.

In most commercial solutions the servers in the cluster rely on stable storage (managed by a database server, for example) to maintain session data consistent as it migrates among servers. That obviously introduces considerable overhead, since most of a session's data is not persistent in nature, being usually limited to the duration of the session itself. To reduce this overhead, solutions include some caching scheme to take advantage of temporal/spatial locality of accesses [10].

In this work we present a solution where session data is kept solely in memory, and migration is handled by the application itself, avoiding the stable storage and cache consistency overheads. We use a central session directory to keep track of the assignment of sessions to servers in the cluster. The directory communicates with the front-end switch and controls it, maintaining a routing table of sessions and servers. Whenever a new session is identified by the front-end switch, the directory is notified and assigns it to a server. The front-end switch is programmed with the appropriate routing association and the assigned server initializes the session data upon receipt of the first request. Following requests are routed to the same server until the switch is notified to behave otherwise by the directory.

When a session must be migrated, the server currently processing it and the newly assigned server are notified and the session information is transferred between them. The old server keeps track of the migration so that any pending requests that may be received after the session is moved can be

re-routed to the new server transparently. The new server instantiates the appropriate internal structures upon receipt of a migrated session and notifies the directory when the operation is completed. The directory registers the new session-server association and notifies the cluster's front-end to update its tables. Any new communication belonging to the migrated session is directed to the new server, which starts processing requests as soon as the transfer of session state is complete.

It should be clear that not all processing is migrated with the session. As mentioned before, requests that alter product dynamic data, for example, cause processing that is related to the product data entity, besides the session. We chose to distributed product information evenly (and statically) among all servers. However, as discussed earlier and verified in the experiments discussed later, the load due to product manipulation is low compared to other session related processing costs, so the vast majority of the load is migrated with the session.

IV. LOAD BALANCING MECHANISMS

The problem of load balancing in distributed servers has been discussed in the literature, specially considering the increase in scalability that it may provide [2, 3]. The proposed solutions do not consider an integration with data replication techniques and service parallelization, however. In this work we consider all these aspects, making the proposed approach more realistic.

As discussed previously, interaction between clients and the e-business server is based on sessions, whose duration and processing demands cannot be predicted beforehand. Since most processing is session-dependant, service distribution must take sessions into consideration when assigning requests to servers. As discussed later in Section VI, Sessions vary widely in terms of number of requests, duration and generated load, so a static allocation of sessions to servers would lead to serious load imbalance. That means that cluster-based dynamic servers must rely on some dynamic load balancing scheme to improve scalability.

Most solutions nowadays rely on "sticky" load balancing: connections from a certain client are assigned temporarily to a server the first time they are seen by the cluster's front-end switch. (This assignment can be done using a hash function based on the connection identifiers, such as IP addresses and port numbers, for example.) Other packets from the same client are routed to the selected server for some time (a few minutes). After that time the switch clears the entry for that session from its tables and computes a new association if a new request arrives for it. Commercial products like Cisco's LocalDirector [6] and IBM's Network Dispatcher [11] are examples of systems offering that solution.

With sticky load balancing, session migration is always

present, since periodically the cluster front-end may decide to route all traffic from a specific session to a different server. However, the decision of migrating sessions do not take into consideration the load due to each session nor the cost of migration: what is considered is simply each server's overall load. The reasoning behind this approach is that given a large number of sessions, a periodic redistribution of their data is bound to reduce load imbalance over time, although possibly at the cost of some extra migration overhead.

Our solution proposes a session-based load balancing scheme. It might be implemented by a separate load balancer process, but we decided to implement it in the session directory described in Section III, considering its role in the process described below:

1. each server keeps track of how much load is caused by each of the sessions assigned to it and periodically (every 10 seconds, in our implementation) reports its total load to the session directory;
2. the directory computes a measure of the overall imbalance in the cluster and what would be an ideal balanced load for all servers;
3. the directory computes how much each server deviates from that ideal load and matches overloaded servers to underloaded ones to facilitate load transfers between them; this matching uses a greedy algorithm to try to minimize the number of exchanges necessary between servers;
4. based on the information from the directory, each overloaded server selects from its more demanding sessions a set that would account for its excess load and migrates those sessions to the underloaded server specified.

To keep track of load on a per session basis, each server keeps track of the ratio between the cpu time for each session and its duration (which has a lower bound of one minute, to avoid peaks during the beginning of each session). Total load is computed as the aggregate CPU time of all sessions.

The matching of overloaded to underloaded servers is not performed just by a direct pairing of instantaneous load differences, because that could lead to instabilities [18]. The solution is to consider the server cluster as a closed loop controller as illustrated in Figure 1. That means that the developer will determine the performance levels acceptable for the system (in terms of latency/throughput) and the controller will take care of finding the right load distribution pattern that honors such levels. The behavior of a closed loop controller is as follows: the user defines the expected behavior for the system in terms of a measurable quantity (the set-point, SP) — for example, the maximum acceptable latency; the controller measures the actual measured value for that variable (the process variable, PV) and computes the error, $e = SP - PV$. Given that error, the controller must compute an actuation value (controller output, CO) to input to

the system that should minimize the error and bring PV to the value defined for SP.

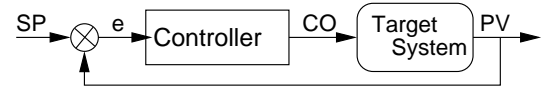


Fig. 1. PID controller

This closed loop controller approach is used in control theory when the relation between the measurable process variable (PV) do not have a clear, straightforward relation to the input variable (CO). In this case, the controller internal behavior is responsible for adjusting the input dynamically, based on the feedback loop, until the desired value is achieved in PV. The traditional form to achieve this goal is to implement what is called a Proportional-Integral-Derivative (PID) controller [17]. In that case, the controller output and the measured error are related by the equation

$$CO = K_p e + K_i \int_0^t e \cdot dt + K_d \frac{de}{dt} + bias$$

The PID constants K_p , K_i , and K_d define the weight of the three components of the error and must be defined by a tuning process for each server.

The directory implements a PID controller for each server based on the information from all servers about their overall load. The controller set point is defined on each iteration as the average load computed by the directory, and the process variable for each server is the actual load reported by that server. Therefore, the controller error is a measure of how much each server load deviates from the average; that error is fed into the PID controller. Since the PID controller tries to cancel the error, it computes how much load should be transferred to/from that server in order to bring it closer to the load average.

V. CASE STUDY: E-STORE

In this section we discuss a popular stateful server architecture: an e-store. Considering the operations performed by an e-store and the different types of information that must be handled, the structure of a server is usually divided into three major components, the WWW server, the transaction server and the database [19, 12]. That organization is illustrated in Figure 2.

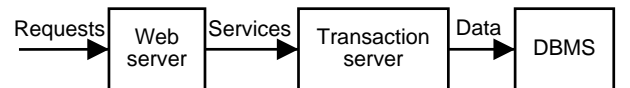


Fig. 2. E-business server architecture

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.

In this paper we analyze and propose a solution for the load balancing problem in 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 smaller as we increase the number of distributed servers for a given workload. The decrease in terms of efficiency comes as a result of the smaller number of requests that a server receives, and thus amount of information 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 imbal-

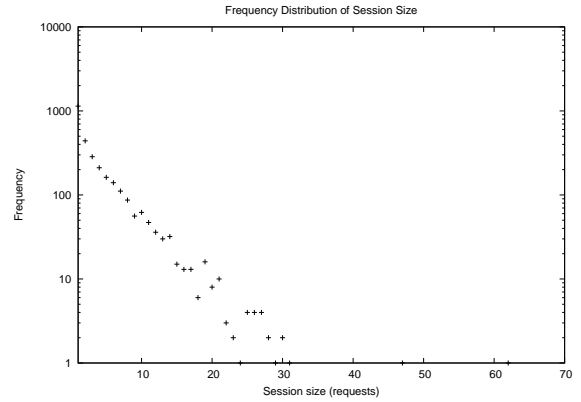


Fig. 3. Frequency distribution of session sizes

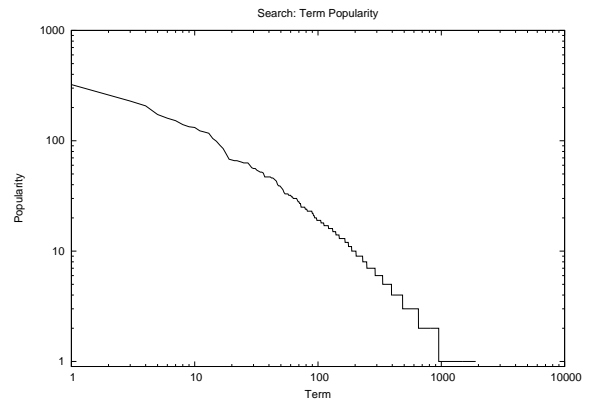


Fig. 4. Search: term popularity

ance 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. This last argument is one of the main motivations of this paper.

VI. 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.

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%). The frequency distribution of requests per session is presented in Figure 3.

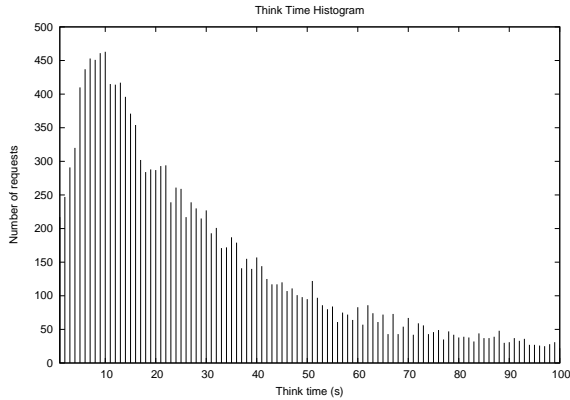


Fig. 5. Think time probability distribution

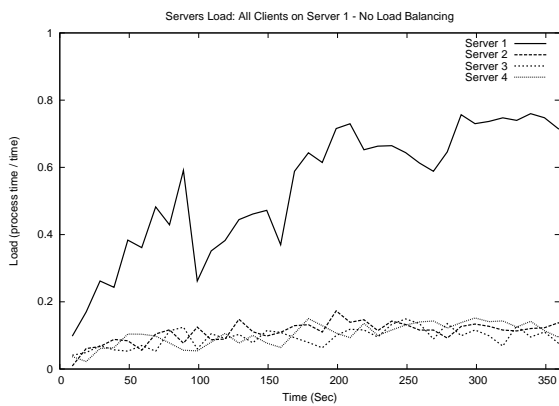


Fig. 6. Servers Load Without Load Balancing

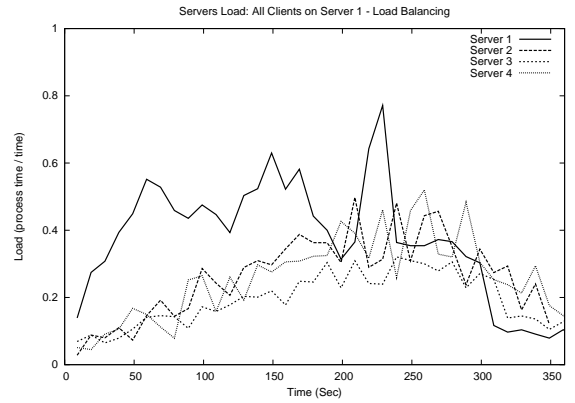


Fig. 7. Servers Load With Load Balancing

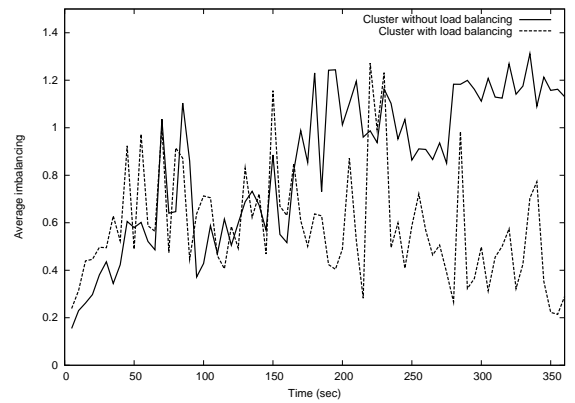


Fig. 8. Average Imbalance

We also characterized the popularity of the parameters for each type of request. Figures 4 presents the frequency distribution of parameters for *search* requests in the workload, which are the query terms. In this case and for all types of requests [7] we can clearly see that the distributions are very concentrated, that is, very few terms account for most of the occurrences, indicating that replication strategies should perform well.

Another evidence of the high variability of this workload is the arrival process of the requests, which may be visualized by the probability distribution of think times, that is, the time between consecutive requests from the same user. This probability distribution is shown in Figure 5 where we can confirm the high variability.

VII. EXPERIMENTAL RESULTS

A. Experimental Setup

In this section we describe the experimental setup we used to evaluate the proposed load balancing strategy.

We distinguish four integrated components in our trans-

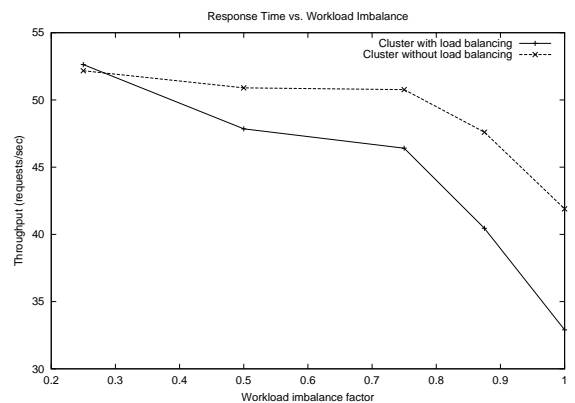


Fig. 9. Servers Throughput

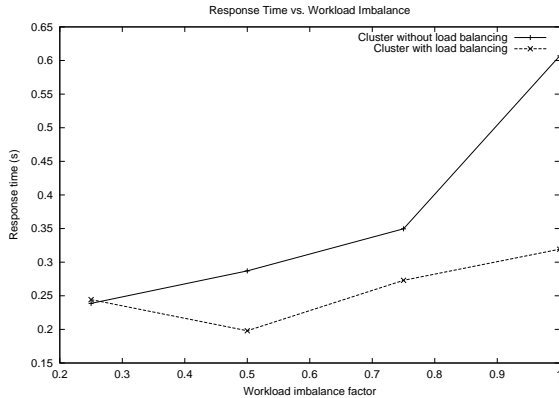


Fig. 10. Servers Response Time

action server: (1) Web server (Apache 1.3.20 [8]), (2) parallelized application server implemented by the authors, (3) database management system (MySQL 3.23.51), and a session directory, described in Section III.

The workload is submitted by Httpperf clients [14] we modified so that they parse server responses and redirect the requests associated with a given session, which was migrated, to a specific server, being the basis of load migration.

The results were gathered in a 10-processor cluster of PCs running Linux 2.4, being six 750 Mhz AMD Duron with 256 Mb RAM and four 1Ghz AMD Athlon with 256 Mb RAM. The machines communicate through a Fast Ethernet Switch. The cluster was organized as follows. The four Athlon machines run the Httpperf clients. One of the Duron machines runs the DBMS and other the directory. The remaining four machines run, each, a WWW server and an application server. Putting the Web and application server together is justified by the fact that the load on the Web server is usually low, since all pages are generated dynamically by the application server. Further, the placement of the two servers on the same machine reduces the communication latency between them.

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.

The workloads generated by each Httpperf client are based on the characterizations shown in Section VI, being the sessions equally distributed among the clients. We should note that distributing sessions equally among clients does not guarantee that the load is the same, since there are significant variations among the load imposed by each session both

in terms of request complexity and in terms of number of requests and arrival process. In order to evaluate the effectiveness of the load balancing strategies proposed, the workloads generated by the Httpperf clients are deliberately imbalanced, that is, a fraction of the requests (that is a parameter and may reach 100%) is submitted to a single server.

B. Metrics

In this section we discuss the metrics we used to evaluate our load balancing strategy. We may divide these metrics into two groups: overall performance metrics and load-related metrics.

The overall performance metrics are response time and throughput. Response time is the server response time for an individual request, that is, the time elapsed between the request arrives and the corresponding response is sent completely. Throughput is the number of requests completed within a period of time. Both metrics allow us to evaluate how well the server is handling requests.

The load-related metrics quantify the operation conditions of the server. We also distinguish two metrics: server load and average imbalance. The server load metric quantifies the amount of processing performed per elapsed time and quantifies the processor usage. The average imbalance is the average of the server imbalances. The server imbalance is defined as the absolute value of the difference between the respective server load and the average server load. These metrics allow us to quantify the effectiveness of the load balancing strategy.

C. Results

In this section we present the experimental results gathered in the experimental setup described. All tests employed four servers and the Httpperf clients generated the same workload.

Our starting point is, as mentioned, a configuration where the workload is completely imbalanced (i.e., one server receives the whole workload) and there is no load balancing strategy being employed. Figure 6 shows the load of each server during the experiment that lasted about 350 seconds. We can see clearly that the load of server 1 is significantly higher than the load of the other servers, although the loads of the latter are not negligible, and are associated with slave threads activity. Further, it is remarkable the impact of the variation of the workload on the load, since there are significant variations (in some cases more than 30%) within few seconds.

We then performed the same experiment employing the load balancing mechanism described in Sections III and IV and the results are shown in Figure 7. We can see that our load balancing strategy was able to divide the load among the servers effectively and consistently. Even under load peaks, as the peak around the experiment time 220, the strategy has shown to be robust, handling the burst and making the load

equal again. We can confirm this evaluation by checking the average imbalance across time for the same experiment, which is shown in Figure 8. We can see that the average imbalance decreases consistently across time, although workload characteristics result in quite significant variation.

We then evaluated the effectiveness of load balancing strategies as a function of the degree of imbalance imposed by the clients, which we call “workload imbalance factor”. For four processors, the workload imbalance factor ranges from 0.25 to 1, which is the fraction of load that is directed to a single server, in our case server 1. For instance, a workload imbalance factor of 0.7 would direct 70% of the sessions to Server 1, while the remaining servers would receive each 10% of the sessions. Figures 9 and 10 show the throughput and response time for various workload imbalance factors. These graphs show again the effectiveness of our load balancing strategy. For instance, for a workload imbalance factor of 1, the throughput of the configuration that employs load balancing is 16% higher and the response time is less than half. The gains reduce as the imbalance decreases, till the point that there is no imbalance and the load balanced configuration gets slightly worse than the other configuration, as a consequence of the load balancing overhead.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper we presented and evaluated a strategy for load balancing of stateful clustered servers. This strategy is based on migrating user sessions among the servers according to PID controller, that defines the sessions that should be migrated according to the overall server load and the load associated with the sessions. The strategy was validated using a e-store application and showed that the proposed strategy allows significant performance improvements (up to 50% for response time and 16% for throughput). We should emphasize the effectiveness of the proposed approach even under workloads that present high variability both in terms of user behavior and in terms of request load.

We see several future work efforts to be pursued. Evaluating the proposed solution for larger configurations is the first one. Devising new load balancing strategies and improve its responsiveness is another area of research. We also intend to evaluate the problem of power management in stateful clustered servers, which would also demand product data migration, since individual servers would be turned off.

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] L. Aversa and A. Bestavros. Load balancing a cluster of web servers using distributed packet rewriting. Technical Report 1999-001, 6, 1999.
- [3] A. Bestavros, M. Crovella, J. Liu, and D. Martin. Distributed packet rewriting and its application to scalable server architectures. In *Proceedings of the International Conference on Network Protocols*, October 1998.
- [4] 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.
- [5] 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.
- [6] I. Cisco Systems. Cisco localdirector series. <http://www.cisco.com/warp/public/cc/pd/cxsr/400/index.shtml>, visited on June 6, 2003.
- [7] B. Coutinho, G. Teodoro, T. Tavares, R. Pinto, W. Meira Jr., and D. Guedes. Assessing the impact of distribution on e-business services. In *First Seminar on Advanced Research in Electronic Business*, Rio de Janeiro, RJ, november 2002. EBR.
- [8] T. A. S. Foundation. <http://www.apache.org/>, 1999.
- [9] 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.
- [10] P. Harkins. Building a large-scale e-commerce site with Apache and mod_perl. In *Proceedings of ApacheCon 2001*, Santa Clara, CA, April 2001.
- [11] G. D. H. Hunt, G. S. Goldszmidt, R. P. King, and R. Mukherjee. Network Dispatcher: a connection router for scalable Internet services. *Computer Networks and ISDN Systems*, 30(1–7):347–357, Apr. 1998.
- [12] D. McDavid. A standard for business architecture description. *IBM Systems Journal*, 38(1), 1999.
- [13] 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.
- [14] D. Mosberger and T. Jin. httpperf: A tool for measuring web server performance. In *First Workshop on Internet Server Performance*, pages 59–67. ACM, June 1998.
- [15] E. Nahum, T. Barzilai, and D. D. Kandlur. Performance issues in www servers. *IEEE/ACM Transactions on Networking (TON)*, 10(1):2–11, 2002.
- [16] 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.
- [17] W. J. Rugh. *Linear System Theory*. Prentice Hall, second edition edition, 1996.
- [18] E. Sontag. A notion of input to output stability, 1997.
- [19] C. Wrigley. Design criteria for electronic market servers. *Electronic Markets*, 7(4), Dec 1997.