Using Adaptive Priority Scheduling for Service Differentiation in QoS-aware Web Servers

Mário Meireles Teixeira

Department of Informatics
Federal University of Maranhão
São Luís, MA, Brazil 65085-580
Phone: +55 98 217 8223
mario@icmc.usp.br

Marcos José Santana
Regina H. C. Santana

Institute of Mathematics and
Computer Science
University of São Paulo
São Carlos, SP, Brazil 13560-970
{mjs, rcs}@icmc.usp.br

Abstract

The current best-effort service model used on the Internet treats all requests uniformly, both in the network and at the application level. However, sometimes it is desirable to provide different classes or levels of service in order to satisfy the needs of different users and applications. In this paper, we propose an architecture for the provision of differentiated services at the web server level. The architecture is verified by means of a simulation model and real web server traces are used as workload. Two priority-based algorithms are implemented in the architecture aiming at service differentiation. The adaptive algorithm, an innovative solution at the application domain, allows the tuning of the priority level provided and determines how strict the use of priorities will be. The system can then adapt itself to various workloads, an essential feature in a highly dynamic environment such as the Web.

Keywords: web servers, service differentiation, priority scheduling, dynamic adaptation, simulation.

1 Introduction

The Internet has experienced an enormous growth in the past few years and there are no signs of calmness in the coming future. The change has not only been in the amount of traffic but also in its nature. Initially, most traffic was composed of plain text, generating a light load, but nowadays we notice a widespread use of images and multimedia content. Moreover, the Web has now become a platform for conducting business transactions, the so-called e-commerce.

The service currently provided on the Internet is based on a best-effort model, which treats all traffic uniformly, without any type of service differentiation or prioritization, a characteristic we find even in the design of critical Internet services, such as the Web. However, not all types of traffic are equivalent or have the same priority to their users [4]. Therefore, it is essential to provide service differentiation with different levels of quality of service (QoS) to different request types [7]. This is also true at the application level, especially in the web servers, which are ultimately responsible for the fulfillment of user requests.

In this work, we propose a novel architecture for a web server capable of providing differentiated services to its users and applications according to their demand characteristics. We consider the existence of diverse classes of users and analyze the implementation of two priority-based service differentiating mechanisms. The results demonstrate that the proposed architecture is actually able to provide service differentiation at the web server level. Our model may be used as a starting point for either the implementation of a server program or the deployment of distributed web server infrastructures.

The strict priority mechanism is an efficient and flexible solution for the provision of differentiated QoS, as it works successfully with various workload profiles and server configurations. However, it may eventually cause the starvation of low priority requests. We also propose an *adaptive priority mechanism*, an innovative solution for service differentiation at the application level. This algorithm allows the fine-tuning of the prioritization level employed by the system and determines how strict the use of priorities will be. Consequently, the server acquires a certain adaptability, becoming able to promptly respond to the changing nature of an environment such as the World Wide

Web.

The remainder of this paper is organized as follows: Section 2 describes the model for the service differentiating web server and Section 3 discusses its validation. Sections 4 and 5 present the strict and adaptive priority algorithms in detail and analyze the results. Section 6 discusses some related work. Finally, Section 7 concludes the paper with some final remarks.

2 Service Differentiating Web Server Model

Current web servers process all requests according to a FIFO (First-In First-Out) discipline, managing a single queue where each request waits to be serviced, in strict arrival order. Although different concurrency control schemes may be implemented in the server to speed up request processing, it generally happens uniformly, without any regard to the peculiarities or urgency of each type of request.

In this section, we propose a generic model for a *Service Differentiating Web Server* (SWDS, in Portuguese) which should be able to provide different levels of service to its clients with quality of service guarantees. Figure 1 describes the proposed architecture, composed of the following modules: a Classifier, an Admission Control module and a cluster of web server processes.

The *Classifier* is the element responsible for receiving requests upon arrival at the server and for dividing them into classes following some previously defined criteria. The *Admission Control* module manages the acceptance of new requests by the server taking into account current service policies and system workload information. In case of system overload, a request may be either rejected (Dropping) or have its QoS requirements downgraded (Negotiation), so that it can be accepted in a lower priority class. After being admitted to the system, the request is assigned to one of the nodes of the web server cluster and is serviced according to the scheduling or service differentiating algorithm currently in operation. After processing, the results are sent back to the clients.

In this work, each cluster node is viewed as a plain web server with a CPU, a disk, a network interface and other resources. The nodes could have also been abstracted as processes, tasks or even CPU's in a parallel computer, since the model does not necessarily imply that the cluster is composed by computers in a distributed system.

3 Model Experimentation

In order to validate our model, we chose a discrete-event simulation using the *SimPack* simulation package[1]. We used real web server traces for workload generation, as seen in other related work. The log files used have been collected from the 1998 World Cup web site [2] and correspond to June 11, 1998, recording accesses to 27 busy web servers. During the simulation, log records are read sequentially and offered as input to the model. If two records have the same timestamp, we assume the arrivals are uniformly distributed in a one-second interval, at the most. The interval upper value is varied in order to generate different arrival rates for the HTTP requests.

A typical web server needs to perform several tasks in order to serve an HTTP request: URL parsing, user authentication, reading of the requested file from disk, transmission of the file to the client and log recording [6] [8]. In the experiments, we set the parameters for the server disks with a data transfer rate of 37 MBytes/s and a seek time of 8.5 ms. We assumed each web server in the cluster was connected to a port on a Fast Ethernet switch with an actual capacity of 80 Mbps. Other processing parameters were obtained from [3] [6].

Our simulation program implements the SWDS server model presented in Fig. 1, with four identical web servers comprising a cluster. All nodes store the same documents. Arriving requests are divided into two classes (high and low priority) by the Classifier module and are subsequently assigned to the cluster nodes in a round-robin fashion. The strict and adaptive algorithms determine the nodes' queuing disciplines. We chose not to use the admission control module in case of system overload in order not to interfere with the performance evaluation of the proposed mechanisms. Therefore, server queues are unlimited.

4 Strict Priority Mechanism

4.1 Concepts

Priority queuing systems treat customers differently, giving preferential treatment to a few privileged classes of users. Strict Priority Scheduling requires that customers are served in strict priority order, meaning that low priority customers will be served only if there are no high priority customers waiting.

In the experiments, customers (namely, HTTP requests) are divided into service classes which are mapped onto the different priority levels provided by the server, numbered from 0 to n. High priority requests are given preference over low priority ones. Requests with the same priority are served, within their class, by the FIFO discipline. We assume the server is non-preemptive and a request assigned to a server

[1]`http://www.cise.ufl.edu/~fishwick/simpack/simpack.html`

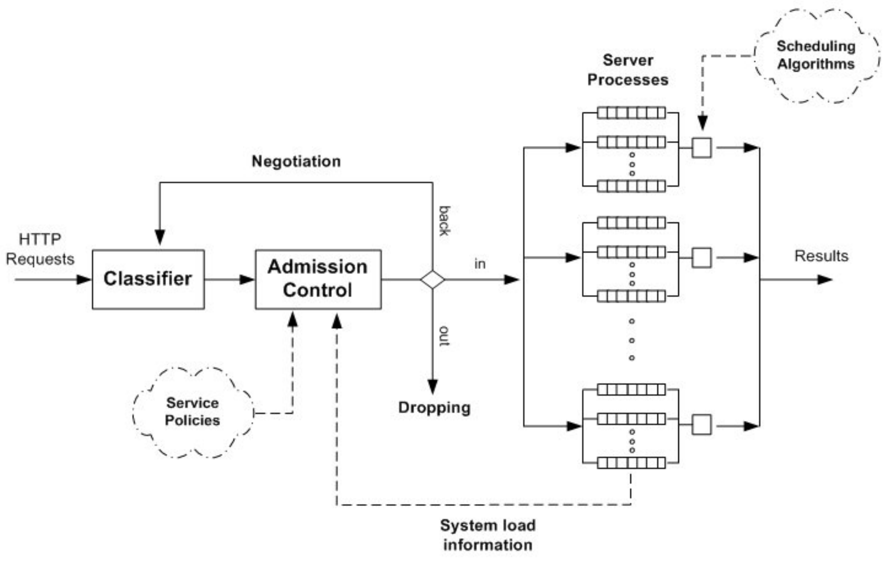


Figure 1: Service Differentiating Web Server (SWDS)

process must remain in that process until service completion. Our experiments aim at determining whether strict priority scheduling is adequate for service differentiation. We use the mean request response time and the ratio of successfully completed requests as performance metrics, for both service classes.

4.2 Experimental Results

Initially, we studied the case where 50% of the clients belong to the high priority class. Figure 2 shows the behavior of mean response time with respect to system utilization. Curve (b) shows the response time pattern when no service differentiating mechanisms are used, the case of a plain web server, such as Apache. Curves (a) and (c) represent the behavior of low and high priority requests, respectively.

We find that non-prioritized response time (b) raises with the increase in system utilization, suffering a severe degradation when the system approaches full utilization. On the other hand, when service differentiation is used, high priority requests (c) exhibit a stable response time even at high utilization levels, without any signs of performance degradation. Moreover, the treatment given to low priority requests (a) is very close to the one provided by an undifferentiated server, which is also an attractive feature of the strict priority mechanism.

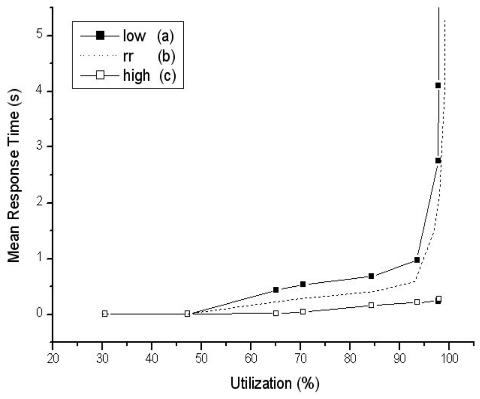


Figure 2: Request response time using strict priority scheduling

4.2.1 High Priority Ratio

We performed other experiments with varying ratios of high priority clients in order to verify their influence in request service time. Figure 3 shows the behavior of the response time of high priority requests, with respect to system utilization, when 20, 50, 70 and 90% of the clients belong to this class. We observe that the higher the ratio of high priority clients, the lower the arrival rate at which the response time curve begins to rise, indicating system overload. Consequently, there is a reasonable limit for the number of clients to be allowed in the high priority class, so as not to hinder

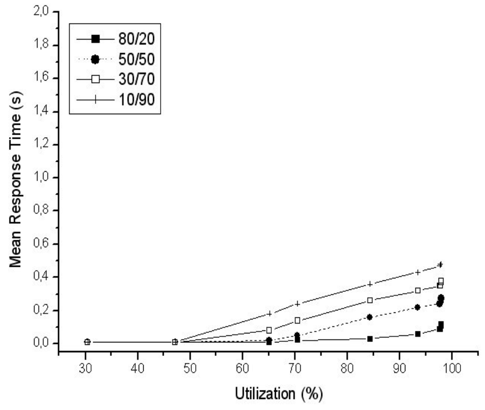


Figure 3: High priority response time for different ratios of privileged requests

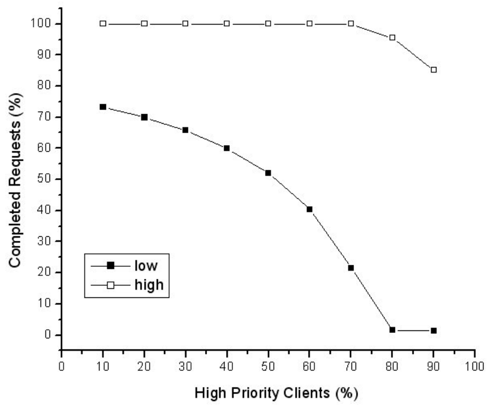


Figure 4: Percentage of completed requests with respect to high priority ratio

service differentiation.

We also analyzed the percentage of completed requests with respect to the total number of arriving requests (completion/arrival ratio). Figure 4 shows the percentage of completed requests, in each class, for high priority ratio ranging from 10 to 90%. The treatment received by low priority requests degrades considerably with the increase in high priority ratio, to the point where less than 2% of them come to a successful completion. This is a typical case of *service denial*, which may cause non-privileged clients to stop submitting requests to the server. High priority clients, on its turn, end up consuming most of the system resources and eventually experience degradation of their own response time.

Our experiments have shown that strict priority scheduling is an efficient mechanism for the provision of differentiated services at the web server level, being more scalable than the Resource Reservation algorithm (RSV) discussed in [10]. However, it is important to pay attention to the performance of all service classes, as it is not usually desirable to heavily penalize low priority requests. Otherwise, we may discourage clients that are potential revenue generators for the business represented by a certain web site. Moreover, if most of the clients arriving at a site receive a high priority tag, it becomes very difficult to realize any sort of service differentiation among them.

5 Adaptive Priority Mechanism

The strict priority mechanism discussed in Section 4 is rather efficient at providing service differentiation among different classes of requests. Nevertheless, it has some disadvantages which can be minimized through the use of a more flexible scheme for request prioritization.

5.1 Algorithm Description

The Adaptive Priority Mechanism proposed here is an innovative solution for service differentiation at the application level, allowing the system administrator to fine-tune the use of priorities according to QoS requirements. Thus, it is possible to associate different degrees of importance with high priority requests in order to prevent them from monopolizing system resources.

In order to implement the algorithm, each server process is defined with a single waiting queue where requests are inserted in strict arrival order. The algorithm uses a look-ahead parameter (k) that specifies the maximum number of positions that will be searched from the head of the queue looking for requests of a given priority (class). If no request of the desired priority is found, the algorithm is repeated for the next lower level and so on. In the worst case, the first request of the queue will be chosen for processing. The higher the value of k, the better the treatment given to higher priority requests. For $k = 1$, requests will be serviced in strict arrival order, i.e., without any service differentiation. We had to modify the source code of the simulator in order to implement this algorithm, as SimPack originally provides only the strict priority discipline on its queues.

The use of the look-ahead as a control parameter for the algorithm allows the tuning of the system's prioritization level. In other words, the look-ahead specifies how strict the priority scheme will be. This algorithm brings adaptability to the SWDS server, which can thus be customized to better respond to variations in system load or to changes in the allocation of clients to service classes.

5.2 Experimental Results

We validated the adaptive algorithm as described in Section 3, using the server model presented in Figure 1. In this case, due to algorithm characteristics, each cluster node is defined with a single waiting queue and not multiple ones, as shown in the model. All cluster nodes use the same value for the look-ahead parameter. We assume 50% of the clients belong to the high priority class. Our aim is to determine whether the use of an adaptive mechanism for the prioritization of HTTP requests is actually able to guarantee service differentiation without over sacrificing low priority requests.

The initial experiments intended to evaluate the influence of the look-ahead value in the ratio of successfully completed requests. Firstly, we executed a simulation with $k = 1$ in order to determine the highest value to be used for the look-ahead. Our results have shown that, for the server configuration used, the maximum size of the nodes' waiting queues (which we will call *max_queue*) was 4,300 requests, an indication of system overload. Maximum mean response time for both service classes was 15.5 seconds, at an arrival rate of 488 requests/sec.

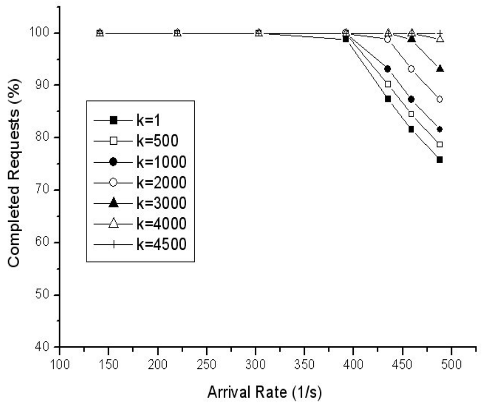
We performed further experimentation in order to analyze the behavior of the ratio of completed requests with respect to the arrival rate, for both service classes. Look-ahead values range from 500 to 4,500. We can notice that higher values of k gradually increase the ratio of high priority requests that reach a successful completion (Figure 5(a)). On the other hand, they markedly worsen the treatment given to low priority requests (Figure 5(b)).

For $k = 1$, the service received by both classes of requests is virtually the same. In this situation, for arrival rates greater than 400 requests/sec, some of the high priority requests do not even reach completion.

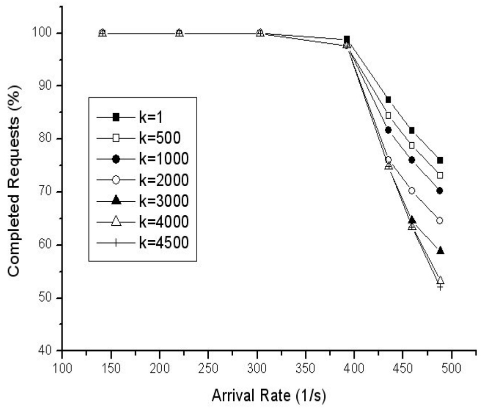
However, for $k > max\_queue$, the adaptive algorithm behaves as the strict priority mechanism discussed above. In this case, low priority requests experience their worst performance, with less than 50% of them finishing successfully. For $500 \leq k \leq 4000$, we observe the other levels of service differentiation realized by the system, according to the initial aim of the adaptive priority algorithm, namely the tuning of the architecture's prioritization level.

The tuning of the look-ahead is critical for the performance of the SWDS server. An extremely high value of k may cause service denial of low priority requests, whereas a low value may hinder service differentiation.

Finally, we analyzed the behavior of request mean response time for different values of the look-ahead, as shown in Figure 6. For $k = 1$, the curves overlap, since the same treatment is given to both service classes. However, for $k = 3000$, the service differentiation becomes evident and the service provided to high priority requests is noticeably better, as initially intended.



(a) High priority requests



(b) Low priority requests

Figure 5: Ratio of completed requests for different values of the look-ahead (k)

5.3 Remarks on the Mechanism

Our experiments have demonstrated that the fine-tuning of the look-ahead value is an efficient mechanism for providing service differentiation. The adaptive algorithm avoids the service denial problems experienced with the strict priority mechanism and allows the system administrator to have a more effective control over the quality of service offered to the clients. The SWDS server can be manually adjusted according to current system load and organization policies.

The look-ahead may also be updated automatically, using information obtained from the Admission Con-

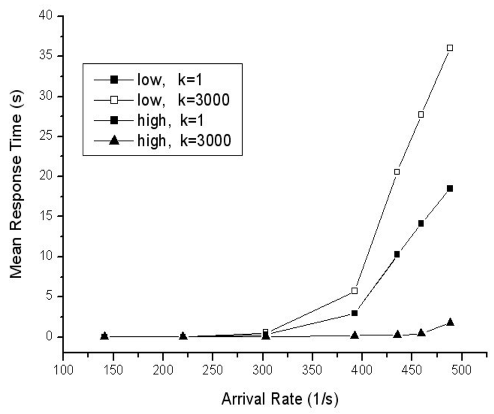


Figure 6: Request response time using adaptive priority scheduling

trol module. In this case, this parameter would be varied so as to follow certain goals set by the system administrator, such as the maximum expected response time for high priority requests or the throughput to be reached for a particular service class. In a real-world scenario, those goals would actually be established in Service Level Agreements (SLA's) contracted by the clients. One possible approach to tune the look-ahead is to describe the situation as an Optimization problem, where the value of k becomes the goal to be met (e.g., a certain response time), subjected to certain restrictions (e.g., the maximum ratio of non-completed requests). It is our intention to pursue this line of research in future work.

6 Related Work

Some previous work has been conducted with the aim of providing differentiated services at the web server level. Eggert & Heidemann [5] propose simple server-side application-level mechanisms for the implementation of a web server with two service classes. Their experiments show that the use of simple mechanisms such as limiting the number of processes per server and priority assignment can provide significant results. The work of Almeida el al. [1] uses priority-based scheduling for service differentiation and implements two scheduling policies. However, the solution presented demands changes in the operating-system kernel. Chen & Mohapatra [3] propose a model for a service differentiating Internet server and validate it by means of a trace-driven simulation, as is the case of our work. The authors also use priority-based scheduling and analyze the admission control of requests in case of server overload. Vasiliou & Lutfiyya [11] propose a QoS module and integrate it with the Apache web server. Apache's FIFO discipline is replaced by a few scheduling algorithms which can be changed at runtime to better suit different goals, a very compelling feature. Rao & Ramamurthy [9] describe a program module that implements two server-side application-level mechanisms to provide different levels of service. The *DiffServer* module intercepts requests coming in from the clients, sorts them according to certain settings and forwards them to the Apache server to be processed. Their results show the average waiting time for high priority requests decreases considerably as compared to a FIFO approach.

7 Conclusions

We proposed an architecture for a service differentiating web server, the SWDS server, which can provide different levels of service to different classes of users. Our model is an evolution from conventional web server architectures, which service clients using a FIFO discipline, without considering the demands of any particular group of users or applications. Our architecture was validated by means of a simulation model and the workload was generated using traces from the 1998 World Cup web site.

We proposed and implemented two priority-based scheduling mechanisms, strict and adaptive, in the SWDS server model. The strict mechanism, commonly found in operating systems and network devices, proved to be an efficient approach for the provision of differentiated services at the web server level as well. It worked satisfactorily with various workload levels and server configurations. Our experiments showed, however, that this mechanism works better when the system is not overloaded and that it is also important to adequately distribute the clients among the service classes in order to prevent high priority requests from monopolizing system resources.

Our adaptive priority mechanism, an innovative solution at the application level, employs a look-ahead parameter in the cluster's waiting queues in order to fine-tune the prioritization level used by the system. Therefore, it is possible to associate different degrees of importance with high priority requests. For high values of the look-ahead, this algorithm behaves similarly to the strict case. On the other hand, when the look-ahead is close to one, the requests are serviced according to their arrival order, without any service differentiation. The adaptive mechanism avoids the problems observed in the strict priority scheme and allows the system administrator to have a more effective control over the quality of service offered to the clients. The adaptive algorithm brings adaptability to the SWDS server, which can thus be customized to better respond to variations in system load and in the allocation of clients to the service classes. We found this mechanism to be the best choice for a highly dynamic environment such as the Web.

As future work, we intend to automatically update the look-ahead using information from the Admission Control module. In addition, we aim at implementing other priority-based mechanisms in the model, such as Weighted Fair Queueing and Earliest Deadline First, which are commonly found in the networking domain. In this case, the challenge is to port those algorithms, devised from a packet viewpoint, to the application level, where we should consider the specific details of web workload. Finally, the implementation of the Admission Control module will allow the provision of even more refined levels of QoS by the SWDS server.

Acknowledgments

We would like to thank Brazilian funding agencies CAPES, CNPq, FAPESP and FAPEMA for their support to the research projects of the Distributed Systems and Concurrent Programming Lab (LaSDPC) at ICMC-USP.

References

[1] J. Almeida, M. Dabu, A. Manikutty, and P. Cao, "Providing differentiated levels of service in web content hosting," in *Proceedings of the 1998 SIGMETRICS Workshop on Internet Server Performance*, mar 1998.

[2] M. Arlitt and T. Jin, "Workload characterization of the 1998 World Cup web site," HP Laboratories, Tech. Rep. HPL-1999-35, sep 1999.

[3] X. Chen and P. Mohapatra, "Providing differentiated services from an Internet server," in *Proceedings of the IEEE International Conference on Computer Communications and Networks*, oct 1999, pp. 214–217.

[4] C. Dovrolis and P. Ramanathan, "A case for relative differentiated services and the proportional differentiation model," *IEEE Network*, sep 1999.

[5] L. Eggert and J. Heidemann, "Application-level differentiated services for web servers," *World Wide Web Journal*, vol. 3, no. 2, pp. 133–42, sep 1999.

[6] Y. Hu, A. Nanda, and Q. Yang, "Measurement, analysis and performance improvement of the Apache web server," in *Proceedings of the 18th IEEE International Performance, Computing and Communications Conference*, feb 1999.

[7] K. Kant and P. Mohapatra, "Scalable Internet servers: Issues and challenges," in *Proceedings of the Workshop on Performance and Architecture of Web Servers (PAWS)*. ACM SIGMETRICS, jun 2000.

[8] D. A. Menascé and V. A. F. Almeida, *Capacity Planning for Web Services: Metrics, Models and Methods*. Prentice Hall, 2003.

[9] G. Rao and B. Ramamurthy, "DiffServer: Application level differentiated services for web servers," in *Proceedings of the IEEE International Conference on Communications*, jun 2001.

[10] M. M. Teixeira, M. J. Santana, and R. H. C. Santana, "Analysis of task scheduling algorithms in distributed web-server systems," in *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, jul 2003.

[11] N. Vasiliou and H. Lutfiyya, "Providing a differentiated quality of service in a World Wide Web server," *ACM SIGMETRICS Performance Evaluation Review*, vol. 28, no. 2, pp. 22–28, sep 2000.