Engineering High Performance Database-Driven E-commerce Web Sites Through Dynamic Content Caching

Wen-Syan Li, K. Selçuk Candan, Wang-Pin Hsiung,
Oliver Po, and Divyakant Agrawal

CCRL, NEC USA, Inc., 110 Rio Robles M/S SJ100, San Jose, CA 95134, USA
Email:{wen,candan,whsiung,agrawal}@ccrl.sj.nec.com

Abstract. The fast growing demand for e-commerce brings a unique set of challenges to build a high performance e-commerce Web site both in technical terms and in business terms. To ensure the fast delivery of fresh dynamic content and engineer highly scalable e-commerce Web sites for special events or peak times continuously put heavy pressures on IT staffs due to complexity of current e-commerce applications. In this paper, we analyze issues related to engineering high performance database-driven e-commerce web sites including: (1) integration of caches, Web servers, application servers, and DBMS; and (2) tradeoff of deploying dynamic content caching versus not deploying. We describe available technology in the scope of *CachePortal* project at NEC. We illustrate performance gains through our technology using an e-commerce Web site built based on some of the most popular components, such as Oracle DBMS, BEA WebLogic Application Server, and Apache Web server.

1 Introduction

Forrester Research Inc. [1] expects that by 2002 over 47 million people will purchase goods and services online in the United States alone and the U.S. Internet commerce will grow to $327 billion by that same year. The fast growing demand for e-commerce brings a unique set of challenges to build high performance e-commerce Web sites both in technical terms and in business terms.

In technical terms, to ensure the fast delivery of fresh dynamic content and engineer highly scalable e-commerce Web sites for special events or peak times continuously put heavy pressures on IT staffs due to complexity of current e-commerce applications. In business terms, the brand name of an e-commerce site is correlated to the type of experience users receive. The need for accounting for users' quality perception in designing Web servers for e-commerce systems has been highlighted by [2]. A typical database-driven Web site consists of the following components:

1. A database management system (DBMS) to store, maintain, and retrieve all necessary data and information to model a business.
2. An application server (AS) that incorporates all the necessary rules and business logic to interpret the data and information stored in the database. AS receives user requests for HTML pages and depending upon the nature of

a request may need to access the DBMS to generate the dynamic components of the HTML page.
3. A Web server (WS) which receives user requests and delivers the dynamically generated Web pages.

One possible solution to scale up database-driven e-commerce sites is to deploy network-wide caches so that a large fraction of requests can be served remotely rather than all of them being served from the origin Web site. This solution has two advantages: serving users via a nearby cache closer to the users and reducing the traffic to the Web sites. Many content delivery network (CDN) vendors [3, 4] provide Web acceleration services. The study in [5] shows that CDN indeed has significant performance impact. However, for many e-commerce applications, HTML pages are created dynamically based on the current state of a business, such as product prices and inventory, rather than static information. As a result, the time to live (TTL) for these dynamic pages can not be estimated in advance. As a result, content delivery by most CDNs are limited to handling fairly static pages and streaming media rather than the full spectrum of dynamic content discussed in this paper.

To coordinate cache servers, WS, AS, and DBMS and ensure fast delivery of fresh dynamic contents is challenging since these servers and DBMS are independent components. And, currently there is no effective and efficient mechanism to ensure that database content changes are reflected to the caches. As a result, most e-commerce sites have to specify dynamic contents as non-cacheable. Consequently, each request to an e-commerce site results in both network delay and server delays (i.e. WS delay, AS delay, and DBMS delay) since the request must be processed each time at the Web site.

In [6], we propose a framework for enabling dynamic caching. Here we further address the following issues: (1) integration of cache servers, WS, AS, and DBMS; and (2) tradeoff of deploying dynamic content caching versus not deploying. We also illustrate performance gains of dynamic content delivery using our approach through a e-commerce Web site built based on some of the most popular e-commerce Web site components, such as Oracle DBMS[7], BEA WebLogic Application Server[8], and Apache Web server.

The rest of this paper is organized as follows. In Section 2, we give an overview of the system architecture of a typical database-driven e-commerce Web site and identify factors that impact response times. In Section 3, we present a loosely-coupled approach that we have developed for e-commerce Web site acceleration. In Section 4 we provide cost analysis on our technology and discuss the tradeoff between deploying caches and not deploying. In Section 5 we report experimental results. Finally we discuss related work and give our concluding remarks.

2 Architecture and Scalability Issues of E-Commerce Web Sites

In this section, we start by presenting the typical architecture of e-commerce Web sites and then identify the bottlenecks that limit the scalability of such

architectures. The architecture of a typical e-commerce Web site consists of has back-end systems, such as DBMS or file systems, that store and maintain the most up-to-date information to model the business and the current state of the business. The application server as described earlier incorporates the necessary rules to capture the business logic and retrieves necessary data by querying the DBMS or the file-system. This data is then processed as per the business logic to dynamically generate the HTML pages. In addition to the requests from the Web and the application server, the DBMS at a database-driven Web site also receives updates (through the Web or back-end processes) to the underlying data in the databases.

When a user accesses the Web site, the request and its associated parameters, such as the product name and model number, and cookie information for customization, are passed to an application server. The application server performs necessary computation to identify what kind of data it needs from the database or file system, or external data sources. Then the application server sends appropriate queries to the database or other sources. After the database returns the query results to the application server, the application uses these to prepare a Web page and passes it to the Web server, which then sends it to the user.

Caching has been viewed as an effective way to scale up Web sites. The percentage of the dynamically generated Web pages on the Internet is getting higher. As a matter of fact, dynamic contents count for more than 25 percent of overall traffic within NEC networks. Unfortunately, since the application servers, databases, Web servers, and caches are independent components, today, there is no efficient mechanism to make database content changes to be reflected to the cached Web pages. Since, most e-commerce applications are sensitive to the freshness of the information provided to the clients, most application servers have to specify dynamically generated Web pages as *non-cacheable* or make them expire immediately. Consequently, repeated requests to dynamically generated Web pages with the same content result in repeated computation in the back-end systems (application and database servers).

Apache Web server can serve static content requests up to more than 2000 requests per second and BEA WebLogic Application Server can also serve static content requests up to 1000 requests per second. However, for the Web site serving dynamic content using BEA WebLogic and Oracle 8i, the performance is less scalable: it can serve traffic up to 300 requests per second in our testing. This is understandable since Web servers and application servers are light-weight programs designed for specific purposes. On the other hand, database management systems are *"heavy weight"* general purpose software. Thus, based on the experimental results presented here, we believe that the appropriate approach to Web acceleration for database-driven e-commerce applications should be to *reduce the load on the database* or to *reduce the frequency of DBMS accesses.*

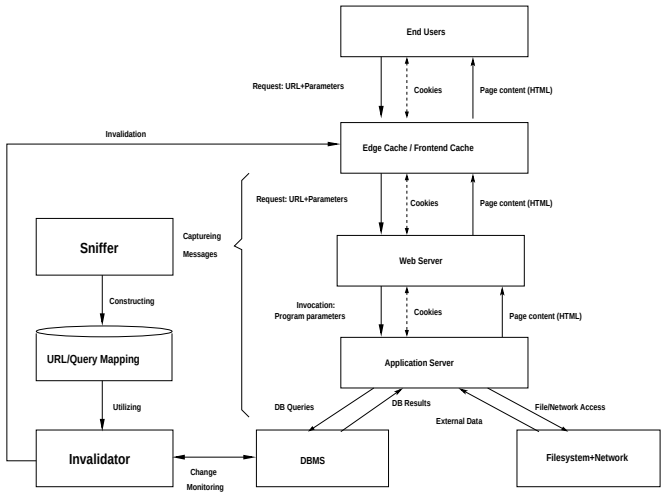

Fig. 1. Architecture of a Database-Driven E-Commerce Site with *CachePortal*

3 *CachePortal* - Technology for Integrating Caches, Web Servers, Application Servers, and DBMS

In this section we describe an open architecture for accelerating delivery of e-commerce content over wide-area networks. The architecture and underlying technology, referred to as *CachePortal*, enables caching of dynamic data over wide-area networks.

3.1 System Architecture

In Figure 1, we show an architecture and data flows in a database-driven e-commerce Web site which employs the `CachePortal` technology. In Figure 1, we illustrate the communication between the caches, the Web server, the application server, and the database as follows: The Web server communicates with the application server using URL strings and cookie information, which is used for customization. The application server communicates with the database using queries. The database changes can be monitored by the tuples and query statements recorded in the database log. However, the database itself, which knows how and what data changes, does not know how the data is used to generate dynamic Web pages. In addition, the database does not know which dynamic content, identified by URLs, is impacted by these database changes.

Our proposed technology *enables* dynamic content caching by deriving the relationships between cached pages and database access via a *sniffer*; and intelligently monitoring database changes to "eject" related pages from caches via an *invalidator*. Note that knowledge about dynamic content is distributed across three different servers: the Web server, the application server, and the database

management server. Consequently, it is not straightforward to create a mapping between the data and the corresponding Web pages automatically in contrast to the other approaches [9, 10, 11, 12], which assume such mappings are provided by system designers.

The sniffer creates the URL to database query mapping (shown in Figure 1). Sniffer collects the query instances, but it does not interpret them. The URL information is gathered either before the Web server using a module which listens to the incoming HTTP requests or before the application server using a module which uses the environment variables set by the application server. The Query/URL map contains **(1)** a unique ID for each row in the table representing the Query/URL map **(2)** the text of the SQL query to be processed by the invalidator and the list of tables in the query, and **(3)** the URL information, including the `HTTP_HOST` string, a list of ⟨cookie,value⟩ pairs, a list of ⟨get variable name,value⟩ pairs, and a list of ⟨post variable name,value⟩ pairs.

The invalidator, on the other hand, listens to the updates in the database and using the Query/URL map, identifies pages to be invalidated and notifies the relevant caches about the *staleness* of the cached page. For this purpose, it interprets the query instances in the QI/URL map. The invalidator consists of three subcomponents. The first component periodically examines the database log to extract the updates since the previous extraction. The second component analyzes the updates (which are in terms of inserts, deletes, and modifications of tuples to specific tables) and identifies the queries that are impacted. These queries are registered in the Query/URL map. The third component identifies the URLs that are invalidated due to impacted queries and generates cache invalidation messages which are destined to the network-wide caches via the API provided by the CDNs. We now show the invalidation process using two examples.

Example 1. Let us assume that we have an e-commerce application which uses a database that contains two tables, `Car(maker, model, price)` and `Mileage(model, EPA)`. Let us also assume that the following query, $Query1$, has been issued to produce a Web page, say $URL1$:

```
select maker, model, price from Car where maker = "TOYOTA";
```

If we observe that a new tuple $(Toyota, AVALON, 25,000)$ is inserted into (or deleted from) the table Car in the database, we would know that the results of $Query1$ is impacted, and consequently $URL1$ needs to be invalidated.

Example 2. In the above example, since the query contains only one table, it is possible to evaluate the impact without any additional information. On the other hand, if we assume the following query $Query2$ which has been issued to produce $URL2$,

```
select Car.maker, Car.model, Car.price, Mileage.EPA
from Car, Mileage
where Car.maker = "TOYOTA" and
      Car.model = Mileage.model;
```

Since $Query2$ needs to access two tables, we can check whether a new tuple inserted into Car does not satisfy the condition in $Query2$ without any additional information. But if the new tuple satisfies the condition, then we cannot check whether or not the result is impacted until we check the rest of the condition, which includes the table $Mileage$. That is, we need to check whether or not that the condition $Car.model = Mileage.model$ can be satisfied. To check this condition, we can issue the following polling query, $PollQuery$, to the database:

```
select Mileage.model, Mileage.EPA from Mileage
where "AVALON" = Mileage.model;
```

If there is a result for $PollQuery$, we know that the new insert operation has an impact on the query result of $Query2$ and consequently $URL2$ needs to be invalidated.

4 Tradeoff between Deploying CachePortal Dynamic Content Caching and Not Deploying

Ensuring freshness of dynamic content in caches does put additional workload to databases due to monitoring database changes and invalidation. We now examine the extra load that polling queries puts on the database management system in greater detail. In particular, we would like to estimate whether the overhead of polling queries on the DBMS completely offsets the savings we realize due to the caching of dynamic content in the network components.

Assuming that the DBMS workload we obtain is that the access rate, A. And the update rate is $U \times A$ and read only access rate is $(1-U) \times A$. Let the cache hit ratio be H. Then, $(1-U) \times A \times H$ accesses to the DBMS are served from the caches, which is the *benefit* of deploying `CachePortal`.

Now let us examine the *cost* introduced by the invalidator to execute polling queries for multi-relation queries. We now model the characteristics of the workload parameters at the invalidator. The polling query workload will depend upon the number of queries in the Query/URL map, which represents the URLs that have been cached in the network.

One major factor that generate the polling queries is the fraction of queries that involve multiple relations although queries that involve single relation do not require execution of polling queries. The cost introduced by the invalidator is $U \times A \times Cost\_Factor$. The more polling queries can be issued and executed for each update, the higher is the $Cost\_Factor$. Let us assume that each update would result in 50 polling queries which can be executed in time within two accesses to the DBMS can be executed. To execute a query, almost 98% of time is connection time. For executing polling queries, we can use persistent connection to execute 50 polling queries at cost equivalent to two units of connection time in our system setting. In this case, $Cost\_Factor$ is 2. Thus, the net benefit of deploying *CachePortal* is $(1-U) \times A \times H - U \times A \times Cost\_Factor$ access workload is removed from the DBMS.

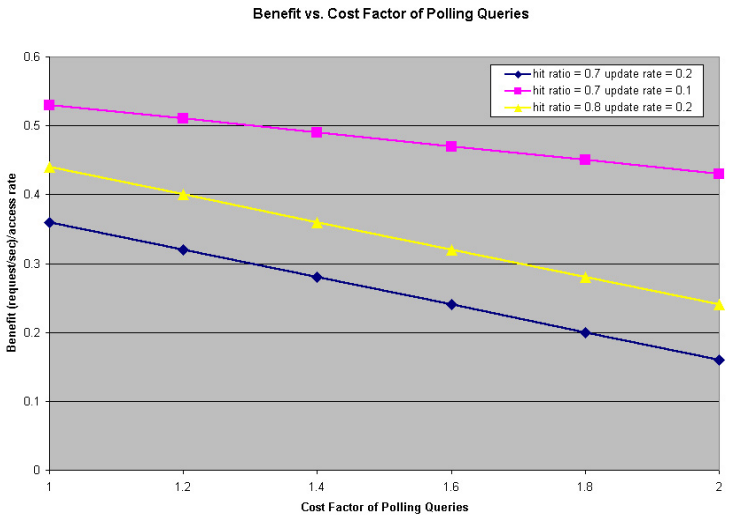

Fig. 2. Benefit of Deploy `CachePortal` with the Respect to Polling Query Cost Factor

There is a tradeoff between using caches and not using caches. Since synchronization would put load to databases, deploying caches would not necessarily be beneficial if content update rates are too high or cache hit rates are too low. In Figure 2, we illustrate the benefit of deploy `CachePortal` technology with respect to polling query cost factor. As we can see, as the cost factor for polling queries increases, the benefit (percentage of requests that are served from caches) reduces. A lower update rate or a higher hit ratio will increase the benefit.

5 Experiments on Evaluating Effectiveness of *CachePortal* Technology

We now report the results of our experimental evaluation of the *CachePortal* technology under different settings. We considered the following two architectural configurations:

Case 1: the Web server and the application server are located on the same machine. and database is located on the separate machine. No cache is used.

Case 2: the Web server and the application server are located on the same machine. The database is located on the separate machine. *CachePortal* technology is applied and a front-end cache, a cache close to servers, is used. The freshness of cached content is guaranteed.

The database used in this experiment contains 7 tables and 25,000 records. The update rate is 1 per second and each dynamic content page request results in a query with one join operation to the database. All machines are Pentium III 700

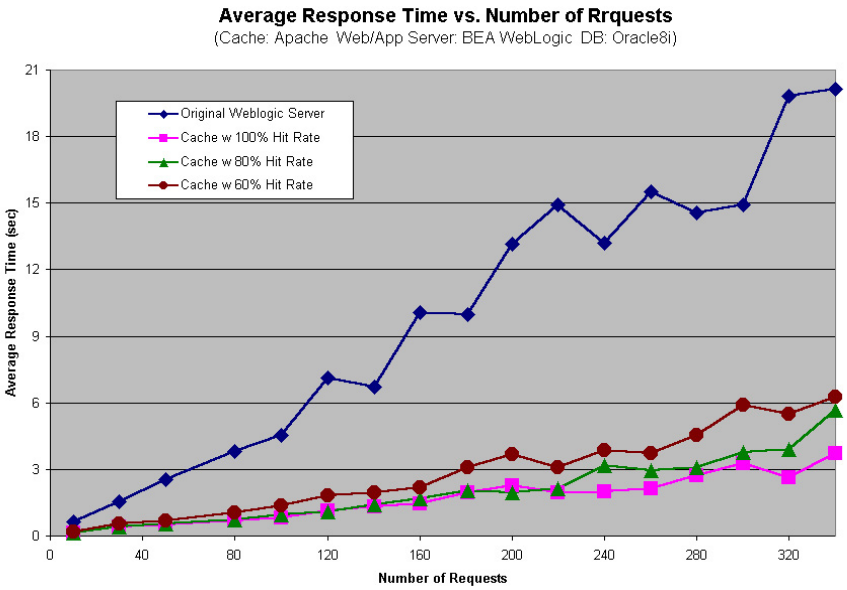

Fig. 3. Evaluation on User Response Time

Mhz CPUs running Redhat Linux 6.2 with 1 GByte of main memory. Oracle 8i is the DBMS employed. BEA WebLogic Application Server is used as Web servers and application servers. Apache Web server is modified to be used as a cache server.

We recorded both the user response time (i.e. round trip time observed by the users) and the server latency (i.e. the time difference between the time when the requests arrive at the Web server and the time when the requests are served from the Web server). Our results indicate that the response time in case 1 increases sharply when the number of HTTP requests increases. On the other hand, the response time in case 2 remains relatively flat for the settings where the hit ratios are 60%, 80%, and 100% respectively. By analyzing response times and server latencies in each case we found that the user response time delay is mainly caused by the server latency rather than the network delay. This means that the requests do arrive at the server without much network delay, but they are queued at the application server waiting for query results from the database.

6 Related Work

Caching of dynamic data has received significant attention recently [13, 14]. Dynamai [9] from Persistence Software is one of the first dynamic caching solution that is available as a product. However, Dynamai relies on proprietary software for both database and application server components. Thus it cannot be easily incorporated in existing e-commerce framework. Challenger et al. [10, 11, 12] at IBM Research have developed a scalable and highly available system for serving

dynamic data over the Web. In fact, the IBM system was used at Olympics 2000 to post sport event results on the Web in timely manner. This system utilizes database triggers for generating update events as well as intimately relies on the semantics of the application to map database update events to appropriate Web pages. Our goal in this paper is to design similar update capability but in the context of general e-commerce setting. Compared with the work [10, 11, 12], where the relationships between Web pages and underlying data are specified manually, our approach automatically generates the query/URL mapping.

[15, 16] propose a diffusion-based caching protocol that achieves load-balancing; [17] uses meta-information in the cache-hierarchy to improve the hit ratio of the caches; [18] evaluates the performance of traditional cache hierarchies and provides design principles for scalable cache systems; and [19] which highlights the fact that static client-to-server assignment may not perform well compared to dynamic server assignment or selection.

SPREAD[20], a system for automated content distribution is an architecture which uses a hybrid of *client validation*, *server invalidation*, and *replication* to maintain consistency across servers. Note that the work in [20] focuses on static content and describes techniques to synchronize static content, which gets updated periodically, across Web servers. Therefore, in a sense, the invalidation and validation messages travels horizontally across Web servers. Other works which study the effects of *invalidation* on caching performance are [21, 22, 23]. Consequently, there has been various cache consistency protocol proposals which rely heavily on *invalidation* [20, 24, 25]. In our work, however, we concentrate on the updates of data in databases, which are by design not visible to the Web servers. Therefore, we introduce a *vertical* invalidation concept, where invalidation messages travel from database servers and Web servers to the caches.

7 Concluding Remarks

In this paper, we analyze the issues related to engineering high performance database-driven e-commerce web sites including: (1) integration of caches, WS, AS, and DBMS; (2) tradeoff of deploying dynamic content caching versus not deploying; and (3) how to design a "cache friendly" Web site. We believe that with `CachePortal` technology, a well-designed cache friendly Web site can be engineered as a high performance e-commerce Web site where the freshness of the Web pages delivered are ensured.

References

[1] Forrester Research Inc., `http://www.forrester.com/`

[2] N. Bhatti, A. Bouch, and A. Kuchinsky. *Integrating User-Perceived Quality into Web Server Design*, International World Wide Web Conference, WWW9, Amsterdam, The Netherlands, pp. 1-16, 2000.

[3] Akamai Technology, `http://www.akamai.com/html/sv/code.html`

[4] Digital Island, Ltd., `http://www.digitalisland.com/`

[5] B. Krishnamurthy and C.E. Wills, *Analyzing Factors That Influence End-to-End Web Performance*, International World Wide Web Conference, WWW9, Amsterdam, The Netherlands, pp. 17-32, 2000.
[6] K. Seluk Candan, Wen-Syan Li, Qiong Luo, Wang-Pin Hsiung, and Divyakant Agrawal. Enabling Dynamic Content Caching for Database-Driven Web Sites. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, CA, USA*, ACM Press, 2001.
[7] Oracle Corporation, `http://www.oracle.com/`
[8] BEA Systems Inc., `http://www.bea.com/`
[9] Persistence Software, `http://www.persistence.com/dynamai/`
[10] Jim Challenger, Paul Dantzig, and Arun Iyengar. *A Scalable and Highly Available System for Serving Dynamic Data at Frequently Accessed Web Sites*, in Proceedings of ACM/IEEE Supercomputing'98, Orlando, Florida, November 1998.
[11] Jim Challenger, Arun Iyengar, and Paul Dantzig. *A Scalable System for Consistently Caching Dynamic Web Data*, in Proceedings of IEEE INFOCOM'99, New York, New York, March 1999.
[12] Eric Levy, Arun Iyengar, Junehwa Song, and Daniel Dias. *Design and Performance of a Web Server Accelerator*, in Proceedings of IEEE INFOCOM'99, New York, New York, March 1999.
[13] Fred Douglis, Antonio Haro, and Michael Rabinovich. *HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching*, In Proceedings of USENIX Symposium on Internet Technologies and Systems, 1997
[14] Ben Smith, Anurag Acharya, Tao Yang, Huican Zhu. *Exploiting Result Equivalence in Caching Dynamic Web Content*, In Proceedings of USENIX Symposium on Internet Technologies and Systems, 1999
[15] A. Heddaya and S. Mirdad. *WebWave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents,* ICDCS 1997.
[16] A. Heddaya, S. Mirdad, and D. Yates *Diffusion-based Caching: WebWave*, NLANR Web Caching Workshop, 9-10 June 97.
[17] M.R. Korupolu anf M. Dahlin. *Coordinated Placement and Replacement for Large-Scale Distributed Caches*, IEEE Workshop on Internet Applications, pp. 62-71, 1999.
[18] R. Tewari, M. Dahlin, H.M. Vin, and J.S. Kay. *Beyond Hierarchies: Design Considerations for Distribued Caching on the Internet*, ICDCS 99.
[19] R.L. Carter and M.E. Crovella. *On the Network Impact of Dynamic Server Selection,* in Computer Networks, 31 (23-24), pp. 2529-2558, 1999.
[20] P.Rodriguez and S.Sibal, *SPREAD: Scaleable Platform for Reliable and Efficient Automated Distribution*, International World Wide Web Conference, WWW9, Amsterdam, The Netherlands, pp. 33-49, 2000.
[21] D. Wessels, *Intelligent Caching for World-Wide Web Objects,* Proceedings of INET-95, 1995.
[22] P. Cao and C. Liu, *Maintaining Strong Cache Consistency in the World Wide Web,* IEEE Transactions on Computers, 47(4):445–457, Apr. 1998.
[23] J. Gwertzman and M. Seltzer. *World-Wide Web Cache Consistency* In Proceedings of 1996 USENIX Technical Conference, pages 141–151, San Diego, CA, January 1996.
[24] H. Yu, L. Breslau, and S. Shenker. *A Scalable Web Cache Consistency Architecture*, In Proceedings of the ACM SIGCOMM'99, Boston, MA, September 1999.
[25] D. Li and P. Cao. *WCIP: Web Cache Invalidation Protocol*, 5th International Web Caching and Content Delivery Workshop, Poster Session, Lisbon, Portugal, 22-24 May 2000.