Caching Strategies for Data-Intensive Web Sites

Daniela Florescu, Valérie Issarny, Patrick Valduriez, Khaled Yagoub

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Caching Strategies for Data-Intensive Web Sites

Daniela Florescu - Valerie Issarny - Patrick Valduriez - Khaled Yagoub

No 3871

Janvier 2000

THÈME 3

ISSN 0249-6399 ISRN INRIA/RR--3871--FR+ENG

INRIA
ROCQUENCOURT

Caching Strategies for Data-Intensive Web Sites

Daniela Florescu* - Valerie Issarny† - Patrick Valduriez‡ - Khaled Yagoub§

Thème 3 — Interaction homme-machine,
images, données, connaissances
Projet Caravel

Rapport de recherche n° 3871 — Janvier 2000 — 43 pages

Abstract: A data-intensive Web site is a Web server that accesses large numbers of pages whose content is dynamically extracted from a database. In this context, returning a Web page may require costly interaction with the database system (for connection and querying) thereby increasing much the response time. In this paper, we address this performance problem. Our approach relies on the declarative specification of the Web site. We propose a customizable cache system architecture and its implementation, in the context of the Weave Web site management system developed at Inria. The system can cache database data, XML fragments or HTML files. We illustrate it using a Web site derived from the TPC/D benchmark database. Based on experiments using our test platform WeaveBench, we assess the performance of various caching strategies. The results clearly show that a mixed strategy is generally close to optimal (static files). Finally, we derive guidelines for specifying cache management based on the Web site specifications.

Key-words: web site, declarative specification, caching strategy, xml, xsl.

(Résumé : tsvp)

* Daniela.Florescu@inria.fr
† Valerie.Issarny@inria.fr
‡ Patrick.Valduriez@inria.fr
§ kya@prism.uvsq.fr/Khaled.Yagoub@inria.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : 01 39 63 55 11 - International : +33 1 39 63 55 11
Télécopie : (33) 01 39 63 53 30 - International : +33 1 39 63 53 30

Stratégies de Gestion de Cache pour des Sites Web à Usage Intensif de Données

Résumé : Un site Web à usage intensif de données est un site Web qui gère un grand nombre de pages dont le contenu est construit dynamiquement, à partir de grandes bases de données. Dans ce contexte, la demande d'une page par un client peut nécessiter une interaction coûteuse avec le système de gestion de base de données (pour la connexion au système et l'exécution des requêtes nécessaires à la récupération des données), risquant ainsi d'augmenter considérablement le temps de réponse. Dans ce papier, nous adressons ce problème de performances en nous appuyant sur la spécification déclarative de sites Web. Nous proposons une architecture configurable de caches à plusieurs niveaux et sa mise en oeuvre dans le cadre de Weave, un système de gestion de sites Web développé à l'INRIA. Dans notre approche, il est possible de cacher des données extraites d'une base de données, des fragments XML et/ou des fichiers HTML. Nous illustrons notre approche à l'aide d'un site Web construit à partir de la base de données du benchmark TPC/D. Puis, nous évaluons expérimentalement, les performances de différentes stratégies de gestion de cache en utilisant notre plate-forme de test WeaveBench. Les résultats obtenus montrent clairement qu'une stratégie mixte est meilleure, et généralement, proche de l'optimal. Enfin, nous donnons un ensemble de directives permettant de configurer la gestion du cache en se basant sur les spécifications du site Web.

Mots-clé : site web, spécification déclarative, stratégie de gestion de cache, xml, xsl.

1 Introduction

Recently, much research has been devoted to improving Web performance by reducing client latency and bandwidth consumption, and increasing servers scalability and availability. Proposed solutions include predictive prefetching, caching of Web objects, and architecting network and Web servers for scalability and availability. There has been also many studies to better understand the Web behavior and performance. Since the mid-90's, various traces have been collected and analyzed. The results show that existing solutions are beneficial but need to be yet improved to accommodate the continuously growing number of Web users and services. In particular, the highest achievable hit rate for proxy caches is about 50%. For instance, the recent study of [92] observes that, in a steady state, 45% of the requests are duplicate and cacheable. Furthermore, only about half of the duplicate requests which could benefit from caching are to cacheable objects. Within traces that were collected in 1997 the percentage of uncacheable documents was only 7% [42]. This raises the need for devising specific techniques for efficiently accessing uncacheable objects. A proxy cache may consider documents uncacheable because they are dynamic, e.g., created by a CGI script, as a result of a query to a database. In [92] , queries account for some 40% of the uncacheable documents. Thus, improving the access performance to dynamic documents is key to the improvement of the overall Web performance.

This paper addresses the performance problem for the case of data-intensive Web sites, i.e., Web servers that provide access to large numbers of pages whose content is extracted from a relational database. More specifically, we propose a cache system for such Web sites. Such Web sites are not to be mistaken with sites that intensively serve large numbers of concurrent requests, which may be made scalable using, for instance, cluster-based architectures. Our goal is to reduce the response time for serving page requests within the Web site while minimizing storage consumption, through an appropriate cache system.

To overcome the performance penalty caused by uncacheable dynamic objects, solutions have been proposed at the level of proxy and server caches. For proxy caches, solutions differ depending on the kind of dynamic objects that are targeted. While delta encoding [9, 68] provides a way to handle any type of uncacheable objects, it may incur significant overhead in terms of both processing load for computing encodings and storage capacity for storing versions. It is thus non suited for documents changing at a high frequency rate, which is the case of documents produced as results of database queries. Alternatively, a dynamic object may be divided into a static part (or template) and a dynamic part where retrieval of the former may benefit from caching while retrieval of the latter leads to Web server access [34]. In this way, the performance cost of accessing a dynamic object is reduced to the minimum. However, in the case of query processing, the load offered to the Web server remains unchanged and hence still requires adequate management at the server level for effective performance improvement. A more general solution is the introduction of cache applets [15] which customize the caching policy for each document. Cache applets enable Web servers to attach computation with Web objects, which may be conveniently exploited by proxy caches (e.g. the proxy cache may request the server only for the document part that

is actually dynamic). Close to this work is the proposal of [10], which introduces the CacheL language for customizing cache management policies according to the specifics of accessed documents and/or services. Here again, these solutions do not improve the Web latency when the requested page is built from database content; the data has still to be requested from the Web server. However, it is our opinion that customized cache management is required for the effective handling of Web documents in general and dynamic documents in particular. The diversity of Web services cannot accommodate a single cache management policy. For instance, in the case of Web documents produced out of database contents, the rate of changes together with the popularity of the data are to be taken into consideration for assessing the relevance of caching HTML files. Hence, the above references give a good base ground towards improving the Web performance. Regarding caching of dynamic objects at the level of Web servers, the proposal of [19] introduces an algorithm for efficient update propagation to caches. It is a first step towards improving Web performance when handling database queries. However, it is aimed at a specific service, i.e., the 1998 Olympic Game Web site. In particular, the targeted service allowed for storing in memory all the dynamic pages without overflow, yielding a cache hit rate close to 100% without applying a replacement algorithm. This assumption cannot be made for Web servers interfacing with databases, especially data-intensive Web sites.

Reducing the Web latency for pages produced by data-intensive Web sites first requires to address timely construction of those pages within the server. In the worst case, returning a page whose content is extracted from a large database leads to interact with the database system, which adds to the -already non-negligible- base cost of Web page delivery. It is thus mandatory to devise an adequate caching strategy within the targeted Web sites. Our approach capitalizes on results from the database research community, which has proposed a new paradigm for building and maintaining data-intensive Web sites, based on declarative specifications [36, 4, 5, 73, 23]. Two main features underlie this paradigm. First, a declarative specification is based on a logical model of the Web site, as opposed to direct manipulation of HTML files. The logical model captures the content and structure of the Web site and is meant to be independent of its graphical presentation. Second, the logical model of the site is defined as a view, in some declarative language, over the data underlying the site. Web-site management systems based on declarative representations have been shown to provide good support for common tasks, which are otherwise tedious to perform (e.g. automatic site updates, enforcement of integrity constraints). Closer to our concern is the extension of this paradigm proposed in [38]. for the efficient processing of documents within data-intensive Web sites. This work exploits declarative Web site specifications for data caching and lookahead computation within the database system according to the users' access patterns and the update frequency of the data items. This improves performance of handling database queries, allows for efficient update propagation, and enables caching of data that are shared among various pages.

In this paper, we propose a cache system for data-intensive Web sites. This work is done in the context of the Weave Web site management system developed at Inria. Our solution enables data caching at the various levels of data elaboration within the Web site, ranging

from results of database queries to HTML pages. In addition, Weave comes along with two declarative languages: one for specifying the Web site's content and one for specifying the customized cache management within the site. This paper makes the following contributions:

- We propose a customizable cache system architecture for data-intensive Web sites and an implementation. The system can cache database data, XML fragments or HTML files. We illustrate our caching architecture using a Web site derived from the TPC/D benchmark database [88].
- We propose a test platform called WeaveBench to assess the performance of various caching strategies for data-intensive Web sites.
- Based on experiments, we assess the performance of the various caching strategies and derive guidelines for specifying cache management based on the Web site specifications.

The paper is organized as follows. Section 2 discusses declarative specification of data-intensive Web sites, and highlights the performance issues with existing strategies for the evaluation of corresponding HTML pages. Section 3 introduces our cache system and the associated language for specifying customized cache management. Section 4 describes our experimentation. Section 5 concludes.

2 Declarative Specification of Data-Intensive Web Sites

The general architecture of declarative data-intensive Web site management systems [4, 5, 73, 23] is based on five fundamental principles:

1. The data content of the entire Web site is managed by a (not necessarily dedicated) Database Management System (DBMS).
2. The *specification* of the Web site is distinguished from the *implementation* of the Web site. By specification of a Web site, we mean the description of the HTML pages forming a Web site, which has to be separated from the imperative code to be executed in order to solve a page request.
3. The specification of the Web site decomposes into: (*i*) that of the Web site's *structure*, and (*ii*) that of the Web pages' *graphical presentation*. The former relates to the set of pages in the Web site, the data content attached to each page and the hyperlinks emanating from each page, while the latter concerns the page's layout such as the placement of data in the page, the color of the background and the page's icons. This approach is already globally accepted since the XML standard aims at isolating the data content of a Web site from the graphical presentation, usually described as independent XSL style sheets.

4. The structure of the Web site is described in terms of a *logical model*. Many different models have been considered for this task [4, 5, 73, 23], most of them being (naturally) based on the notion of graph.

5. The mapping between the raw data (in the original model) and the logical model of the Web site is described via a *declarative language*.

As one of the prototypes pioneering this approach, the Weave system obeys to all five principles [38]. The rest of this section presents the Weave system and illustrates the associated language with an example. Then, it discusses evaluation strategies for data-intensive Web sites.

2.1 The Weave system

The Weave system targets Web sites whose data content is derived from large relational databases, and adopts the graph data model proposed for XML [33] for the Web site logical model. An instance of this data model corresponding to a particular Web site is then called the *XML site graph*. In this context, XML becomes the natural language to represent site graphs. Moreover, XML site graphs are defined in the Weave system *intensionally*, via *XML site schemas*, rather than extensionally (i.e. one Web page –XML fragment– at a time). A site schema represents then nothing else than an XML view definition over a relational database. Applying the site schema to a particular instance of the database results in a complete XML site graph. Two components are then fundamental in the Weave system: (*i*) the **XML Generator**, which applies the intentional definition of the site schema to the underlying data and produces (fragments of) the XML site graph, and (*ii*) the **HTML Generator**, which applies XSL templates to XML fragments, resulting in browsable HTML pages. We further detail the notions of site graphs and schemas below.

XML site graphs: Formally, an XML site graph is a directed labeled graph with two types of nodes: internal nodes corresponding to Web pages, and leaf nodes corresponding to data values attached to pages. Links between pages are modeled as arcs between the internal nodes representing them in the graph; these are called *hyperlink arcs*. The site graph also associates with every Web page the data that is displayed on the page. These associations are represented by *data arcs* from internal nodes to leaves. In addition, the following elements are associated with every hyperlink arc l in the site graph: a string valued attribute, which specifies the semantics of the hyperlink between the two nodes (e.g. "client"), and a string valued anchor, which is the string shown on the HTML link corresponding to the arc (e.g., the name of the client "Joe Smith").

Declarative modeling of Web sites is especially effective when the pages in the site can be classified into a small number of homogeneous collections [36](e.g., collections corresponding to pages of customers or suppliers). Such collections are referred to as *site classes*. Note that we can always model a highly specialized Web page (e.g., the root) as a class with

one member. Pages in a Web site (and hence nodes in the XML site graph) can then be uniquely identified by their class and a set of items from the underlying database. Hence, we associate a unique identifier with each Web page (and as a consequence with each internal node in the XML site graph) of the form $C(X_1, \ldots, X_n)$, where C is a site class name and $X_1, \ldots, X_n$ are a list of data items from the database. This identification mechanism of the pages in the Web site is one of the fundamental concepts of the Weave system.

XML site schemas: XML site schemas intensionnaly define an XML view over a relational database. Formally, an XML site schema G can be modeled as a labeled directed graph which contains one internal node for every site class. The node for a site class C of arity n is labeled by an atom of the form $C(X_1, \ldots, X_n)$ where $X_1, \ldots, X_n$ are variables. The leaves of the site schema are labeled with single variables and correspond to data items. Arcs in the site schema are labeled as follows:

- **Hyperlink arcs:** a hyperlink arc in the site schema between $C_1(\bar{X}_1)$ and $C_2(\bar{X}_2)$ is labeled by a triple: $(q, anchor, label)$, where: (a) q is a SQL query denoting the condition on the values of the variables $\bar{X}_1 \cup \bar{X}_2$ that has to be satisfied in the database such that there is a hyperlink between the instance of C_1 and the instance of C_2, (b) *anchor* is either a constant string or one of the projected variables of q determining which value will appear on the HTML anchor associated with the link, and (c) *label* is a string denoting the name of the attribute resulting in the site graph.
- **Data arcs:** a data arc in the site schema between $C_1(\bar{X}_1)$ and Y is labeled by a pair: $(q, label)$, where q and *label* have the same meaning as above and $\bar{X} \cup \{Y\}$ are projected variables of q.

In Weave, XML site schemas are described using the WeaveL language. A WeaveL program consists of a set of site class specifications. Each Web site class specification includes: the declaration of the parameters identifying an instance of the class, the SQL query whose result gives all possible instances for the above parameters (therefore describing how to produce all instances of that class), the specification of the data contained in an instance of the respective class, and the specification of the hyperlinks from an instance of the respective class. The XML Generator component has the task of evaluating the queries in the site schema and produce the corresponding XML data. In order to produce a complete page, all the data attributes and hyperlinks (with their anchors) emanating from the given page have to be computed. However, it is important to note that the XML Generator is capable to generate on request either the XML data corresponding to an *entire* page, or a *fragment* of it, corresponding to an attribute or a link type. Finally, a complete Web site specification in the Weave system consists of a WeaveL program, which gives the XML view definition, and a set of XSL style sheets, which state how to map XML fragments in browsable HTML.

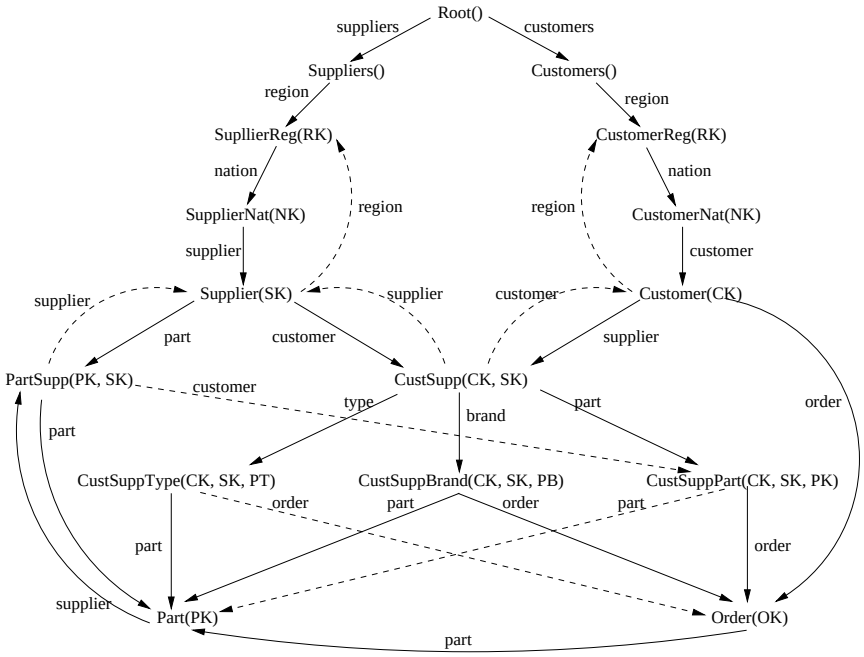

Figure 1: The site schema for the TPC/D example.

2.2 Example: a Web site derived from the TPC/D database

We use the following example throughout the paper and in our experiments. Suppose we want to produce a browsable version of the data contained in the TPC/D benchmark [88]. The database contains information about products, customers and client orders. A simplified version of the TPC/D relational schema is given below.

Part(partkey, name, brand, type, size)
Supplier(suppkey, name, address, nationkey, phone)
PartSupp(partkey, suppkey, availqty, supplycost, comment)
Customer(custkey, name, address, nationkey, phone)
Nation(nationkey, name, regionkey, comment)
Region(regionkey, name, comment)
Lineitem(orderkey, linenumber, partkey, suppkey, quantity, shipdate)
Order(orderkey, custkey, orderstatus, totalprice, orderdate, orderpriority)

The site schema shown in Figure 1 provides the following organization of the data. There is a root page with two links to suppliers (node labeled `Suppliers()`) and customers (node

labeled `Customers()`). Both suppliers and customers are grouped by geographical region (e.g., `SupplierReg(RK)`), and inside each region by nationality (e.g., `SupplierNat(NK)`). Suppliers and customers have further links to detailed information about the orders. Specifically there is one page for each customer-supplier pair (`CustSupp(CK,SK)`) where the customer `CK` ordered from the supplier `SK`: this page is accessible both from the supplier and customer pages. From here, there are further links to pages detailing orders placed by that customer to that supplier. Of course, in designing the Web site, we also add sufficient links back to facilitate navigation. For example, the page grouping links to customers in France can be identified as CustomerNat(6), where 6 is the key in the database identifying the country "France".

Let us give the specification of the site class CustomerNat in the WeaveL language. A Web page instance of the CustomerNat class depends on a single parameter: the key in the database of the given nation. Each such page contains different types of data: the name of the nation and the comment attached to this nation, and different types of links: link to the page of the corresponding region and links to the pages of all the customers of this nation. The WeaveL specification of this class is as follows:

```
Class CustomerNat parameters NK
  {
    data:  name obtained as
  /* The name appearing on an instance of this page is obtained by the following query */
       select name
       from Nation
       where nationkey=$NK
    data:  comment obtained as
  /* The comment appearing on an instance of this page is obtained by the following query */
       select comment
       from Nation
       where nationkey=$NK
    hyperlink: customer to instances of the class Customer
     parameters $CK obtained as
  /* An instance of this class will have hyperlinks to all the instances of the class Customer,
  whose parameters are obtained by the following query.
  The query also specifies how to obtain from the database the anchors
   of the respective hyperlinks.*/
       select custkey as $CK, name as anchor
       from Customer
       where nationkey=$NK
    hyperlink: region to instances of the class: CustomerReg parameters $RK
    obtained as
  /* An instance of this class will have hyperlinks to all the instances of the class
   CustomerReg, whose parameters are obtained by the following query:*/
       select r.regionkey as $RK, r.name as anchor
       from Customer c, Nation n, Region r
       where c.nationkey=n.nationkey and n.regionkey=r.regionkey and c.nationkey=$NK
  }
  {
  values for NK
  /* the Web site contains one instance of this class per nation in the database */
    select nationkey as $NK
    from Nation
  }
```

Given a particular binding for the parameter NK of the class CustomerNat, the XML Generator produces the XML fragment corresponding to the respective Web page. For example, given NK=6, the following XML fragment is generated:

```
<Web_page id="CustomerNat_6" class="CustomerNat" parameters="6">
  <data_fragment attribute_name="name">
    <data_value>France </data_value>
  </data_fragment>
  <data_fragment attribute_name="comment">
    <data_value> 3mjmizlS3L3k2h </data_value>
  </data_fragment>
  <link_fragment link_name="region">
    <link>
      <anchor>Europe </anchor>
      <Web_page id="CustomerReg_3" class="CustomerReg" parameters="3"/>
    </link>
  </link_fragment>
  <link_fragment link_name="customer">
    <link>
      <anchor> Customer#000000402 </anchor>
      <Web_page id="Customer_402" class="Customer" parameters="402"/>
    </link>
    <link>
      <anchor> Customer#000000413 </anchor>
      <Web_page id="Customer_413" class="Customer" parameters="413"/>
    </link>
    ...
  </link_fragment>
</Web_page>
```

The XML Generator can be invoked with the request of generating only a fragment of the above XML page. For example, it can be requested to build the "customer" fragment of the page, in which case, it will return an incomplete XML containing only t he XML link fragment associated with the link named "customer".

The links corresponding to the missing fragments appear within the optional missing_fragments attribute of the root element of the generated XML fragment as described bellow:

```
<Web_page id="CustomerNat_6" class="CustomerNat" parameters="6"
          missing_fragments="name/comment/region">
  <link_fragment link_name="customer">
    <link>
      <anchor> Customer#000000402 </anchor>
      <Web_page id="Customer_402" class="Customer" parameters="402"/>
    </link>
    <link>
      <anchor> Customer#000000413 </anchor>
      <Web_page id="Customer_413" class="Customer" parameters="413"/>
    </link>
    ...
  </link_fragment>
</Web_page>
```

Let's notice that all the XML fragments (complete or incomplete) that we manipulate adhere to the same DTD, which is independent of the database schema and the structure of the Web site. The DTD we use is the following:

```
< !ELEMENT Web_page (data_fragment | link_fragment)* >
< !ELEMENT data_fragment (data_value)+ >
< !ELEMENT data_value (#PCDATA) >
< !ELEMENT link_fragment (link)+ >
< !ELEMENT link (anchor, Web_page) >
< !ELEMENT anchor (#PCDATA) >
< !ATTLIST data_fragment attribute_name CDATA #REQUIRED >
< !ATTLIST link_fragment link_name CDATA #REQUIRED >
< !ATTLIST Web_page id CDATA #REQUIRED >
          class CDATA #REQUIRED
          parameters CDATA #IMPLIED
          missing_fragments #IMPLIED >
```

Finally, to get a complete Web site, we need to provide a set of XSL style sheets, one per site class, in addition to the WeaveL specification. The XSL style sheet corresponding to class "CustomerNat" is given bellow. The XSL is for the complete XML fragment.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0">
<xsl:output method="html"/>
<xsl:param name="urlbase"/>
<xsl:template match="/Web_page">
<HTML>
 <HEAD>
  <TITLE> TPC-D Data Base Customers/Nation</TITLE>
 </HEAD>
 <BODY>
  <xsl:variable name="URLBASE">
   http://caravel.inria.fr/weave?
  </xsl:variable>
  <H1>
    TPC-D Data Base <BR/>
    Customers of
    <xsl:value-of select="data_fragment[@attribute_name='name']"/>
    Nation
  </H1>
  Name:
   <xsl:value-of select="data_fragment[@attribute_name='name']"/> <BR/>
  Comment:
   <xsl:value-of select="data_fragment[@attribute_name='comment']"/>
    <BR/>
  Region:
   <xsl:value-of select="link_fragment[@link_name='region']"/> <BR/>
  Customers: <BR/>
   <xsl:for-each select="link_fragment[@link_name='customer']">
    <UL>
    <xsl:for-each select="link">
     <LI>
     <A href=
      "{$URLBASE}id={Web_page/@id}
      &class={Web_page/@class}
      &parameters={Web_page/@parameters}">
      <xsl:value-of select="anchor"/>
     </A>
     </LI>
    </xsl:for-each>
    </UL>
   </xsl:for-each>
 </BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

Given XSL style sheets, the HTML Generator is able to match the XML fragment generated by the XML Generator with the corresponding XSL style sheet and to produce the final HTML file. The HTML file generated for the XML fragment corresponding to NK=6, is as follow:

```
<HTML>
<HEAD>
 <TITLE> TPC-D Data Base Customers/Nation</TITLE>
</HEAD>
<BODY>
 <H1> TPC-D Data Base <BR> Customers of France Nation </H1>
 Name: France <BR>
 Comment: 3mjmizlS3L3k2h <BR>
 Region: Europe <BR>

 Customers: <BR>
  <UL>
   <LI>
   <A href="http://caravel.inria.fr/weave?id=Customer_402
    &class=Customer
    &parameter="> Customer#000000402
   </A>
   </LI>
   <LI>
   <A href="http://caravel.inria.fr/weave?id=Customer_413
    &class=Customer
    &parameter=413"> Customer#000000413
   </A>
  </LI>
  ...
 </UL>
</BODY>
</HTML>
```

2.3 Evaluation strategies for data-intensive Web sites

Up to this point, we have concentrated on the declarative specification of Web sites with the Weave system. A multitude of strategies may then be applied for the evaluation of the HTML pages. We now discuss various evaluation strategies ranging from purely dynamic to fully static.

Let us first consider a purely *dynamic* evaluation strategy. Upon each page request, the Web server invokes the XML Generator to produce the XML fragment corresponding to the requested page. In order to produce the result, the XML Generator runs the appropriate

parameterized SQL queries on the database, according to the site schema specification, and packages the result in XML format. The XML fragment is then sent to the HTML generator, which applies the appropriate XSL style sheet and generates the final HTML file. Thus, the total waiting time can be decomposed as follows: (1) the *communication time*, (2) the *HTTP connection time*, (3) the *Web application startup time*, (4) the *DBMS connection time*, (5) the *SQL execution time*, (6) the *XML generation time* and (7) the *HTML generation time*. In the case of dynamic evaluation of Web pages from large databases, the entire process can be unacceptably expensive. Various solutions have been proposed to reduce the waiting time for a page in this context. Some solutions focus on limiting or eliminating one of the above 6 costs. For example, expensive CGI calls can be replaced by efficient APIs, or servlets. Furthermore, most products avoid systematic database connection through a pool of connections. However, these solutions do not solve the hard performance problem in producing HTML pages derived from large databases.

A more general solution, used by most existing products [43, 86, 46, 47], relies on materializing HTML pages. The materialization can be done either on the fly (i.e. by applying various caching policies), or offline, before any user starts browsing. Despite very good performance, the latter *static evaluation* strategy has several major drawbacks. First, it incurs significant space overhead, which can be even greater than duplicating the entire database since the same data item may appear in multiple pages. Furthermore, the same HTML template is replicated for each page in a class. Second, propagating updates from the database to the Web site is a serious problem once the site has been materialized [49, 53]. Third, the materialization granularity (i.e. a page) is not always appropriate: different fragments in a page can have different update frequencies, and materializing at the page level imposes the recomputation of the entire page, even if some parts of the page did not change. Finally, the static approach cannot always be applied since it cannot accommodate forms (i.e. the page content depends on the values of some parameters which are only known at run-time).

In [38], the authors addressed the problem of run-time management of data intensive Web sites. Their solution relies on the observation that, in the case of pure dynamic evaluation, the parameterized queries issued from the Web server and executed on the DBMS server share much of their computation, hence leading to redundant work. The proposed solution is thus to cache on the DBMS the result of parameterized SQL computation, under the form of relational tables called *cache functions*, and reuse the result for subsequent requests. This approach has several advantages. First, if the SQL execution time is the dominant cost and if this cost is high, there are significant performance improvements. Second, since the cached data and the raw data are under the control of the same DBMS, simpler update propagation mechanisms can be deployed. Third, it is possible to control the granularity of the cached data, ranging from entire parameterized SQL queries to simpler sub-queries. In this way, a single cache function can help solving requests for multiple types of pages [38].

However, it is easy to remark that this solution is not the panacea and can alter the Web site's performance under certain conditions. First, since the cached data is under the control of the DBMS, every cache action (e.g. search, insert, delete) accesses the DBMS via expensive SQL statements. Hence, using a cache with a low hit ratio incurs a large penalty.

Second, the SQL execution time is not always the most prominent cost in evaluating a page and may even be negligible. In this case, the overhead of caching tuples in the DBMS may outweigh the performance improvement. Finally, another drawback of this solution is to overload the DBMS server.

An alternative solution to the two above extremes is to cache intermediate XML representations of data. Compared to caching HTML files, XML caching has the advantage of storing less data. Moreover, XML representations allow for carefully controlling the granularity of cached data, ranging from the entire page to fragments of the page. For instance, we can cache the name of a product, which is somehow stable, but not its price which varies a lot over time. However, caching XML data instead of HTML files will not exhibit a clear advantage in terms of space saving when the ratio between the size of the XML representation and that of the corresponding HTML file is close to 1. Under such condition, caching large XML data will not only bring no benefit in terms of space saving, but will additionally incur runtime overhead to convert XML data into HTML on the fly. Compared to DBMS caching [38], caching XML has the advantage of eliminating (in the case of a hit) the costs of database connection, SQL execution, and of generating XML data. Moreover, this technique allows to diminish the load generated on the DBMS by the Web server. On the other hand, update propagation from the DBMS to the cached data is made more difficult (e.g. [19]).

From the above discussion, we can easily conclude that there is no *universally* good solution for improving the performance of a data-intensive Web site. Each of the above caching techniques (i.e. HTML, XML or DB) will be effective under certain circumstances and disastrous under others. This leads us to introduce a 3-level caching architecture including the aforementioned HTML, XML, and DB caches. Moreover, the run-time behavior of a Web site has to be controlled by a customizable *run-time policy* so as to make optimal usage of the caches according to behavioral information such as the users' access patterns and the data's evolution patterns. Intuitively, a run-time policy specifies *which* kind of data to prefetch or cache (HTML pages, XML fragments, relational tables, or any combination of those), *which* particular items to prefetch or cache (e.g. which particular HTML pages or XML fragments), and *which* actions to execute under different events: page requests, data updates, environmental changes. The foreseen advantage of a 3-level caching architecture, coupled with *run-time policy* customization, is that it makes possible to specialize the run-time behavior of a Web site, according to each user, to the user's context or to the portion of the Web site being accessed, and to automatically balance the load of such system. From that perspective, declarative Web site specification may be exploited for characterizing the construction of HTML pages from underlying data but also for inferring the optimal run-time policy when combined with behavioral information about the Web site. The latter capability is a topic of future research with many open issues.

3 Customizable Cache System

In the section, we introduce the Weave cache system with a 3-level caching architecture. Its behavior is customized according to a high-level specification of the –foreseen optimal– caching behavior to be enforced, which is based on declarative Web site specification using WeaveL. In the rest of this section, we present the architecture of the cache system, the specification of customized cache management which we illustrate with an example of the TPC/D Web site, and the system implementation.

3.1 Architecture

Figure 2 depicts the overall caching architecture of the Weave cache system. The architecture includes a manager for each individual cache (i.e. DB, XML, and HTML) and the global cache manager. These managers implement usual cache operations for data retrieval, addition and removal, which are triggered following events such as HTTP requests, invalidation, cache overflow. These operations are exported through the cache *Interface* that is similar for all four managers. Given a page to be retrieved following a HTTP request, the *Global manager* interacts with one or more of the individual caches, which may themselves interact with other individual managers. The interaction pattern among the cache managers for page retrieval depends on the caching policy prescribed for the given page. For instance, if the page is requested to be stored in the HTML cache, then the global manager first issues a request to it. In case of a hit, the page is simply returned. In case of a miss, the HTML cache may then contact directly the DB cache or both the XML and DB caches depending on whether fragments of the page are requested to be stored. Let us notice here that when a data is to be retrieved from the database, there is always an interaction with the DB cache even if the data is not cached within the DBMS. This is because the DB cache interfaces with the DBMS, and offers additional capabilities such as pooling of DBMS connections. The interaction between each manager also includes a protocol for data translation; protocols are depicted by the circles in Figure 2, and correspond to the XML and HTML Generators discussed in Section 2.

As already said, the benefit of data caching depends on the Web server's behavior, including actual data re-accesses, and data update frequency. Performance improvement due to caching may further be increased using prefetching if there are recurring sequences of requests. The server's behavior is best approximated through the collection of statistics, which may be exploited for *a priori* deriving, possibly automatically, optimal cache management or even dynamically adapting it. The latter capability is depicted in Figure 2 by the presence of tools dedicated to statistics collection. As pointed out previously, our system does not integrate means for the automatic generation of an optimal, possibly dynamic, run-time policy for Web servers out of statistics. However, to ease the developers' task, we introduce a declarative language for the abstract specification of the cache system's behavior. Such a language builds upon declarative Web site specification, and enables the definition of per-site-class customized caching. The language comes along with a compiler, which adapts

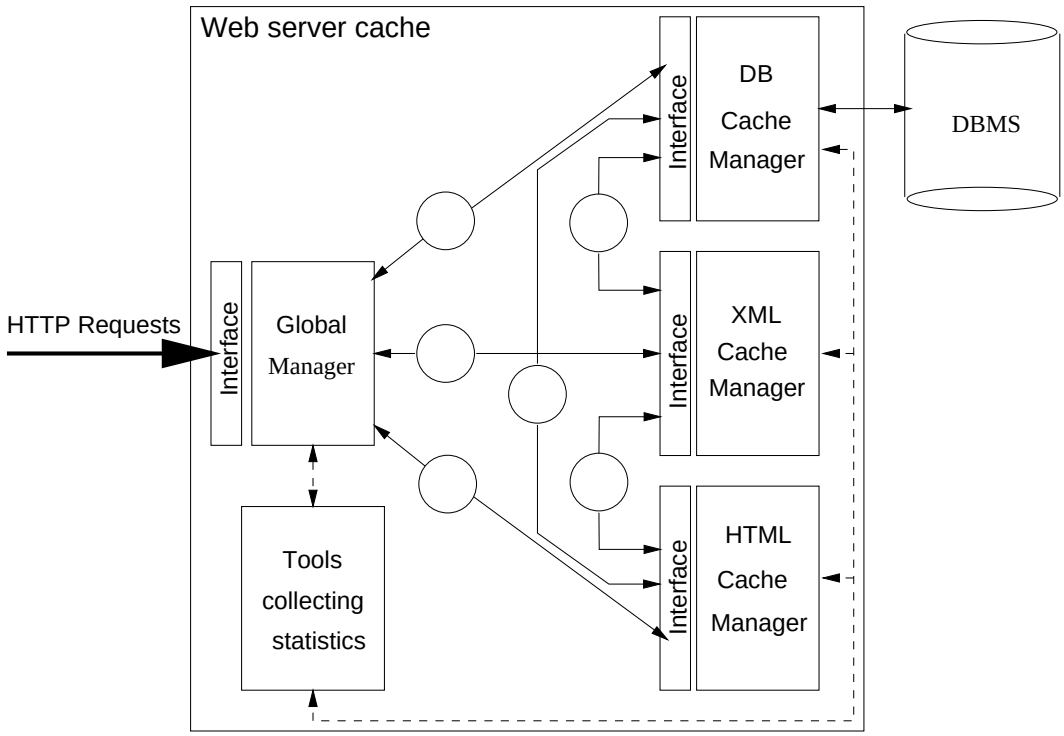

Figure 2: 3-level caching architecture

the behavior of the system's global and individual cache managers according to the given specification.

3.2 Specifying customized cache management

Given the notion of site graph defined in Section 2.1, cache management may be specified at a fine grain. For each site class, the action to be taken within the cache can be specified based on the occurrence of relevant events and according to the cache's state and dynamic information (e.g. statistics, subset of the collection to which the request applies). Precisely, the specification of cache management for a Web site using our system has two parts:

(*i*) **Global cache management**: it specifies the global policies implemented by the individual cache managers for setting the overall features of the cache (e.g. maximum cache size, cache distribution) and specifying the actions to be carried out upon a global event (e.g. overflow).

(*ii*) **Data cache management**: it prescribes the actions to be taken within the cache upon the occurrence of an event (e.g. retrieval) on some data of the Web site as identified by the site graph.

The proposed declarative language for specifying cache management enables to declare event-condition-action rules, written as "*event*: **if** *condition action*" for the two above parts. The events include at least the operations listed in the interfaces of the various managers.

Global cache management: We now give an overview of the overall specification for global cache management for an individual cache, which may be the DB, XML or HTML cache. The following specification gives the cache features together with the rules relating to its global management.

global management CacheName:
features:
MaxSize: *Sets the maximum size for the cache.*
Timer: *Sets period for the alarm.*
Containers: *Sets distribution pattern over the cached data.*
Instance: *Sets the basic implementation choices for the cache, e.g., storage on disk.*
management:
OnTimer: *Action to be taken upon the alarm occurrence.*
OnAccess: *Action to be taken upon a request, independent of the requested data, e.g., use of cached connections in the case of the DB cache.*
OnFull: *Action to be taken upon cache overflow, which relates to specifying the replacement algorithm to be used.*

The `Container` part serves setting the distribution pattern for cached data. For instance, in the case of a cluster-based architecture, one may want to distribute the HTML files among the machines in a way that group commonly accessed pages according to the users' access patterns. For the DB cache, containers correspond to the cache functions of [38] . In the following, definition of containers is exploited only for the DB cache since their handling is already supported in our system as it is the basis for DB caching. On the other hand, our system prototype does not currently integrate support for the distribution of cache managers, and hence the definition of containers for the XML and HTML caches would not bring any benefit. The `Instance` part enables prescribing the implementation that is chosen for the given cache. For instance, the overall number of fragments that are requested to be cached together with their access pattern and the storage capacity of the underlying server may allow for storage of the XML cache in main memory. As another example, we may consider distinct implementations for the XML cache with respect to the chosen encoding (e.g. DOM, ASCII).

In the proposed abstract description of global cache management, we have only made explicit events appertained to rules. However, this does not mean that the listed events are the only ones supported, these are defined by the developers according to the cache

management specifics. The condition and action parts of the rules have been left quite imprecise under the form of comments. In the most general form, the condition is a boolean expression over behavioral information (i.e. combination of statistics data and state information). Regarding the action, it may be expressed using either some abstract specification language or a more implementation-oriented approach which would specify a system function implementing the relevant mechanism. Our language supports both approaches since they are equally useful: the former eases the design of the system's customization while the latter prescribes an implementation that must match the design decisions. However, from a practical point of view, abstract specifications are expressed as comments and hence are not processed, and implementation-oriented specifications are given in terms of the names of the operations that are implemented within our system prototype.

Data cache management: For specifying customized data cache management, we exploit the definition of site classes. For each class, the language helps prescribing customized management within each individual cache –if relevant (e.g. the changing rate and data sharing among pages make only DB caching valuable). Basically, customized management amounts to specifying how to handle events related to data retrieval, addition, removal, and staleness (i.e. behavior of the global and individual cache managers of Figure 2). Let us recall here that a site class characterizes a collection and hence a set of pages. In that context, cache management may be customized for a subset of pages of the collection by prescribing the values identifying the subset within the conditions of the related rules. In addition, as previously stated, conditions may embed expressions related to behavioral information about the Web site. The actions prescribe how to handle events according to the caching decision for the specific data. For instance, if there is an action associated to the retrieval event for a given cache, this means that the data is to be cached. Similarly, if a miss within a cache leads to request for the data within another cache, this means that the data is assumed to be cached by the latter. Thus, a number of static checks can be carried out upon a given specification to detect inconsistencies among the various rules. We give below the general form for such a specification.

```
customized management
 /*States the site class for which cache management is to be customized: */
     Class: name parameters params:
 /*Specifies the actions to be taken by each manager upon events relating to the collection:*/
     DB: rules
     XML: rules
     HTML: rules
     GLOBAL: rules
```

We do not go into the detail of rule specifications since these may be exploited for characterizing either abstract design or concrete implementation choices. In the case of implementation-oriented specification, actions can name: internal operations offered by the

embedding cache manager, events processed by the various managers, and calls to the XML and HTML Generators for data translation among caches.

3.3 Customized cache management for the TPC/D Web site

As an illustration, we consider the TPC/D example that was introduced in Section 2.2. We further assume the following properties for the classes:

- The `Root` class down to the `SupplierNat(NK)` and `CustomerNat(NK)` classes correspond to pages whose underlying data change unfrequently and that have a high rate of re-access. Hence, those pages are requested to be cached within the HTML cache and only this one.
- The intermediate level class `Supplier(SK)` and `Customer(CK)` relate to pages that decompose into fragments which are changing at different rates. Precisely, fragments embedding data values and links to `CustSupp(CK,SK)` are quite stable and hence stored within the XML cache. On the other hand, remaining fragments are not cached and require access to the DBMS.
- The `CustSupp`, `CustSuppPart`, `CustSuppBrand` and `CustSuppType` classes are cached within the DB cache only.
- Remaining classes are not cached.

We get the following management specification for the overall caching architecture. In particular, regarding the events that appertain to the Web site's data, we provide only the specification for the `Get` and `Retrieval` events, which are the most representative of the system's usage. In addition, we give only conditions relating to the cache state in the rules definitions, and do not consider those based on the server's behavior. Let us first give the specifications that are global to the individual caches:

```
global management DBCache:
  features:
     MaxSize: 3MB
     Timer: 20mns
     Containers:
       define function F as
            select o.custkey, l.suppkey, p.partkey, p.name, s.name
            input LineItem l, Order o, Supplier s
            where l.orderkey=o.orderkey and s.suppkey=l.suppkey
         input custkey, suppkey
         max size = 2M
         min hit ratio=0.3
         max age = 20mns
     Instance: DBMS
global management XMLCache:
  features:
     MaxSize: 300M
     Instance: DOMMain
  management:
     OnFull: LRU
global management HTMLCache:
  features:
     MaxSize: 500M
     Instance: HTMLFile
  management:
     OnFull: LRU
```

Let us notice here that there is no rule prescribed for the DB cache since its global management is actually enforced by the DBMS. The `Containers` part of the DB cache states that the DBMS stores the `F` cache function, which is a relational table that is to be dynamically updated by the DB cache upon relevant data accesses (see [38] for detail).

Let us now give a sample of the classes' specifications from the standpoint of cache management according to the design choices that were stated previously. The following notations are used: event names begin with a capital letter while operations that are internal to the managers are all written in small letters. The specification provided below for the `CustomerNat(NK)` class embeds only rules for the DB and HTML caches. It prescribes that a requested page within the collection is first searched within the HTML cache. Then, the page is returned to the global manager upon a hit if the page is considered as being valid. Otherwise, the page is built through the generation of a `Get` event towards the DB cache manager, which issues the corresponding query to the DBMS and returns the XML fragment encoding the Web page to the HTML cache. This latter action is actually achieved using the Weave XML generator, which issues the necessary queries to the DBMS, using the definition given for the `CustomerNat` class. Upon response from the DB cache, the HTML cache produces the page corresponding to the returned XML fragment using the associated XSL style sheet and the Weave HTML Generator. The generators are named

`XMLGenerator` and `HTMLGenerator` and both provide the `gen` function. The `gen` function of the latter takes as input a complete XML fragment and associated XSL style sheet, and returns the corresponding HTML page. The `gen` function of the former takes as input a site class name, a list of sub-fragments (possibly empty if the request is for the complete page), and parameter(s) characterizing the required fragment, and returns the corresponding XML fragment. In the remainder, we further use the `aggregate` function of the XML Generator, which aggregates into a single fragment, a set of XML fragments provided as input.

```
customized management
    Class: CustomerNat parameters NK:
    DB:
      OnGet(CName, FSet, Param): return XMLGenerator.gen(CName, FSet, Param)
    HTML:
      OnGet(CName, Param): return Retrieval(search(CName, Param))
      OnRetrieval(Page, CName, Param):
        if (hit and not stale) return Page
        else if (hit and stale)
          remove(Page),
          let p = HTMLGenerator.gen(DB Get(Cname, {}, Param), "CustomerNatXSL")
                  in add(p), return p
        else if miss
          let p = HTMLGenerator.gen(DB Get(CName, {}, Param), "CustomerNatXSL")
                  in add(p), return p
    GLOBAL:
      OnGet: return HTML Get("CustomerNat", $NK)
```

We now detail cache management for the `Customer(CK)` class. Upon access to a page of this collection, the fragment giving data values and lists of customers is cached within the XML cache while the other fragment is not cached at all. This thus leads to introduce independent requests for these two types of fragments to the DB cache, i.e. SQL queries to the database are split. A requested sub-fragment is identified by naming the embedding class together with relevant sub-fragment names. We get the specification given below, which does not prescribe any rule for management of the HTML cache since pages of the `Customer(CK)` class are not requested to be cached. Computation of an HTML page out of the cached fragment and the complementary one is achieved within the global manager, which calls the HTML generator that is provided with fragment aggregating the two fragments composing the given page.

```
customized management
    Class: Customer parameters CK:
    DB:
      OnGet(CName, FSet, Param): return XMLGenerator.gen(CName, FSet, Param)
    XML:
      OnGet(CName, FSet, Param): return Retrieval(search(Cname, FSet, Param))
      OnRetrieval(Fragment, Cname, FSet, Param):
        if (hit and not stale) return Fragment
        else if (hit and stale)
          remove(Fragment),
          let F = DB Get(Cname, FSet, Param) in add(F), return F
        else if miss
          let F = DB Get(Cname, FSet, Param) in add(F), return F
    GLOBAL:
      OnGet: return HTMLGenerator.gen(
                      XMLGenerator.aggregate(
                          XML Get("Customer", {"name", "comment", "region"}, $CK),
                          DB Get("Customer", {"customer"}, $CK)),
                      "CustomerXSL")
```

As a last sample of caching specification, let us consider that for the `CustSupp(CK, SK)` class, whose underlying data are cached within the DBMS using the `F` cache function as declared in the `containers` part of the DB cache. Upon request for a page of this collection, the global manager calls the HTML generator, which is provided with the XML fragment corresponding to the page. This fragment is computed by the DB cache after getting the necessary underlying data from the DBMS out of `F`. Furthermore, prior to issue SQL queries for computation of the requested XML fragment, the DB cache requests the DBMS to (possibly) update the `F` function with the entries relating to the given fragment. For the sake of brevity, we do not provide the corresponding specification, the interested reader is referred to [38] for detail.

Up to this point, we have discussed how to define customized cache management for Web sites using a declarative specification language. Such a specification precisely states design choices for the cache system's customization, and may be exploited for reasoning about the resulting behavior of the Web site with respect to performance, by exploiting either expected or traced patterns of accesses and of data evolution. In addition, behavior of the Web site may be dynamically adapted using statistics so as to, for instance, cache only relevant subsets of data out of those that are requested to be cached. Typically, such selective caching will be achieved through the addition of conditions according to statistics within management rules. In general, evolution of the Web site may be easily mastered given the level of abstraction of the proposed management specification. Such an evolution may either be handled at run-time if anticipated throufinely tuned cache management forgh adequate conditions or lead to update the specifications if the needed adaptation was not considered in the first place (e.g. integration of new site classes).

3.4 System implementation

Ultimately, we want to automate most of the caching management out of the declarative specifications of the Web site's data and relevant information about the Web site's behavior. Considering the specifications that were introduced above, we would like the generation of run time policies and their execution to be automated based on the definition of the site classes and the statistics about the Web site's behavior. This issue is currently dealt with through explicit specification of run time policies using the language introduced in this section. However, let us notice that, even in the absence of an automatic way to generate run time policies, the fact that we support a high level language for specifying such policies already eases the production of data intensive Web sites, compared to existing approaches. Using our system, the site administrator is simply requested to abstractly characterize cache management. The resulting specification can then be easily exploited for reasoning about the resulting performance improvement compared to the low-level specification of the cache management strategy given by an implementation in some imperative programming language. In addition, as already stated, revision of the cache management strategy can easily be addressed for further performance improvement. However, for this argued simplicity to be actually beneficial, the implementation of cache management out of its specification should be automated. We have thus devised a compiler for our specification language, which customizes cache management accordingly within our cache system.

The current prototype implementation of our cache system builds on the Java-based Apache server [77] and integrates a number of Java classes for the implementation of the cache managers. The managers interact with existing systems (i.e. the DOM store and HTML Generators from IBM [3, 2] where the former has been extended with persistence capability, and the Oracle DBMS) through Java interfacing where JDBC is used for interfacing with the DBMS. The overall caching architecture as depicted in Figure 2 is currently centralized and hence all the managers run within a single JVM. Each manager is implemented using a dedicated Java class, which integrates base methods for event handling and internal management (i.e. methods related to global management such as the replacement algorithm, and those appertained to data management such as data retrieval). Let us now address implementation of customized cache management out of provided abstract specifications within our system prototype. Given specifications, the corresponding implementation derives as follows:

- The specifications related to global management that are provided for each individual cache are first used to identify the Java classes to be used for the corresponding managers, while we use the system `Global` Java class for implementation of the global manager. Precisely, the `Instance` part of any global management specification is required to give the name of an available Java class, which corresponds to a base manager implementation. This class further offers methods for remaining features so as to set the specified values, and methods for events and internal management as given in the global management specification. For instance, considering the specification for the HTML cache given in the previous subsection, our prototype embeds

the `HtmlFile` Java class, which includes methods for: setting the value of `MaxSize`, handling the `OnFull` event, and the LRU replacement algorithm. Finally, each base Java class implementing a manager includes methods for internal management and for events appertained to data management, i.e., the ones given in the customized cache specifications of site classes. Considering specifically the implementation of the DB cache, we further have to deal with the definition of containers (i.e. cache functions). This is addressed by stating adequate queries to the DBMS for creating the corresponding tables, using JDBC methods.

- The second generation step amounts to complete the body of the event-related methods according to the global management specifications. This is achieved through translation of the condition and action parts of rules into Java code. This is quite straightforward given the simplicity of our language. Basically, the language comprises the `let` and conditional constructs, which are trivial to translate into Java. In addition, we have event generations and operation calls stated through corresponding names. Since those names equate to methods declared within the base Java class for manager implementation, the two above features translate into method calls.

- The last generation step lies in producing per-object customized cache management according to the specifications provided for site classes. Our solution consists in generating a Java class per site class for which customized management is specified, according to the same principle as above. The event-related methods of corresponding Java objects are then invoked within the methods of the embedding manager as necessary. Let us notice that in the absence of customized caching for site classes, corresponding data are not cached at all and the page is produced using a default Java class, which gets the XML fragment from the database and then generates the HTML page.

Although the proposed cache system greatly simplifies the implementation of customized cache management for data-intensive Web servers, its usefulness primarily lies in the performance gain that can be obtained using it instead of existing solutions. This issue is addressed in the next section, which assesses through experiment the actual performance improvement that our system enables to achieve.

4 Experimentation

To measure the performance of various caching strategies, we built a test platform called WeaveBench and performed various experiments. In the following, we present the test platform and its configuration, the experiments and the performance results.

4.1 The WeaveBench test platform

Although benchmarks exist for measuring the performance of Web servers, for instance, WebStone 2.0 [67] and SPECweb99 [84], they do not consider database access. Therefore, we decided to build our own test platform WeaveBench.

WeaveBench generates a load for testing Weave on a Web server by running one or more Web client processes on one or more client computers. We can choose the number of processes to run and the type of data requested by the processes. Each Web client process sends HTTP requests similar to those that a Web server would normally receive from Web browsers or from other Web data retrieval software. Each client process sends requests as fast as it can receive data back from the server. Thus, it generates a load much heavier than that of a single interactive user.

A single process manages the testing done by the client processes. It starts the benchmark runs, each one by a different client process, and combines the performance results into a single summary report.

WeaveBench considers a Web site derived from the TPC/D database [88] factor 1 with a database size of 1.2 GB and 15 million pages. Each client submits page requests based on a trace file that describes the pages of the TPC/D database. Thus, we must generate a trace file that captures a realistic workload of page accesses.

In the experiments, each client sends requests to the Web server according to a trace file. Each page is identified in the trace by a class name (e.g. `Customer`) and one or more parameters (e.g. $CK=5). The class and the parameters are used to generate HTTP GET requests for dynamic pages that Weave will build on the fly in the Web server. Each client has its own trace file. However, all the trace files for a given run are generated based on the same probability distribution, as follows. First, a workset of N (N=10.000 in the current implementation) distinct pages have been chosen from the entire Web site. In so doing, the pages closer to the root of the Web site are considered with a (slightly) higher probability. Second, C (C being the number of clients) sequences of length M (M=1000 in the current experiments) of pages are chosen from the workset, according to a Zipf distribution.

The database is stored in Oracle v8 on a dedicated Ultra Sparc I machine (143MHz and 384MB of RAM), running SunOS Release 5.5. The cache global manager, the three caches and the Web server, Apache v3.3.3, are all on the same machine, a 300MHz Pentium with 520MB of RAM, running Linux Release 6.1. We use the Apache JServ servlet engine [77] to run the cache global manager, which we implemented as a servlet. Two other machines are used to hold WeaveBench clients. Each machine, a 200MHz Pentium with 96MB of

RAM, ran about 1-50 clients in increments of 10. All the machines in the test platform are interconnected by a 10Mbps network that could be isolated from other networks.

4.2 Experiments

The experiments are the following.

1. **Dynamic pages**. This is the worst case scenario where client requests always yield access to the database, generation of the XML fragment and generation of the HTML page. The goal is to measure the system performance for dynamic page generation.
2. **Static files**. This is the best case scenario where all the HTML files have been statically generated. This scenario is unrealistic because of the high costs for generating (after each change) and storing the files. The goal is to observe the ideal response times.
3. **DB caching**. Only the DB cache is used here. The goal is to measure the benefit of caching data within the database.
4. **XML caching**. Only the XML cache is used here. The goal is to measure the benefit of caching XML fragments.
5. **HTML caching**. Only the HTML cache is used here. The goal is to measure the benefit of caching HTML fragments.
6. **Mixed caching**. This is the realistic scenario where all three caches are used. The goal is to assess the benefit of caching at various levels (database, XML fragment, HTML files).

For ease of reading and interpretation, we use a two-dimensional table to present the performance results of each experiment. Each row is for a page class (`Customer`, `Supplier`, etc.) and gives the distribution of the various execution times (wsconnect, queryexec, etc.), the average response time and the average size of the HTML file for a number of page requests. For clarity, each execution time is normalized as a percentage of response time and is computed as follows. Let T be an execution time (such as wsconnect). Let N_p be the number of page requests of page class P.

The average execution time for T is computed as:

$$Avg(T) = \frac{\sum_{i=1}^{N_p} T_i}{N_p} \tag{1}$$

The average response time is simply:

$$resptime = \sum_{i=1}^{4} Avg(T_i) \tag{2}$$

Then the normalized execution time for T is computed as:

$$Norm(T) = \frac{Avg(T)}{resptime} \quad (3)$$

The table is sorted by decreasing order of response time. All results are shown for 1 client generating 500 page requests. However, we also experimented with up to 50 clients to verify that the numbers for the cost components remain constant (in some cases, response time was slightly increased). In addition, we provide only results in the absence data update. However, the experiments we ran in the presence of data update gave the expected result in that its penalty upon performance increases as the grain of caching gets greater. The database queries generated by the client requests have been implemented the best possible way, with significant amount of tuning using indices on large tables.
Notice also that we have done all the experiments without turning off the Web server and the Java servlet engine logs.

4.3 Performance results

4.3.1 Dynamic pages

Table 1 shows the results when there is no caching. The average response times are between 15 s and 300 ms. For page classes `Customer`, `Supplier`, `CustSup`, `CustSuppPart` and `CustSuppBrand`, the query execution time dominates because the queries involve joins with large tables. For page classes `CustomerNat` and `SupplierNat`, the XML and HTML generation times dominate because the queries are simple but produce a large amount of data. For the other page classes, the response time drops to 450 ms and below, and is divided between the Web server connection and the HTML generation. Thus, these page classes need not be optimized.

4.3.2 Static files

Table 2 shows the results when the HTML files have been statically generated. Thus, the only relevant execution times are those for connecting to the Web server and transferring the HTML files to the client (htmlgen). Response times are optimal and are between 3 s and 10 ms for the most expensive pages. For page classes of larger HTML size (`CustomerNat`, `Supplier`, `SupplierNat` and `Customer`), the time to transfer the HTML files dominates. For the others (2 KB and below), the Web server connection time dominates and the response time gets quite small.

page class	wsconnect(%)	queryexec(%)	xmlgen(%)	htmlgen(%)	resptime(ms)	size(KB)
Customernat	1.52	0.81	44.36	53.32	15202.34	660
CustSupp	1.84	97.38	0.04	0.74	12530.95	2
CustSuppBrand	1.85	96.73	0.05	1.37	12459.98	2
CustSuppType	1.91	96.57	0.05	1.47	12063.53	2
Supplier	2.30	82.96	6.36	8.38	10044.03	100
CustSuppPart	2.52	96.51	0.06	0.92	9160.41	2
PartSupp	8.23	88.40	0.28	3.09	2802.67	1
Suppliernat	22.30	3.63	25.78	48.29	1034.73	60
Customer	30.26	43.80	6.73	19.21	762.52	15
Order	51.11	24.74	2.89	21.27	451.44	1
Customers	51.88	0.00	0.75	47.37	444.73	2
Supplierreg	69.03	6.73	1.14	23.10	334.23	1.5
Suppliers	72.77	0.00	1.30	25.93	317.06	1
Customerreg	74.10	6.96	1.14	17.81	311.40	1.5
Part	74.78	6.52	1.21	17.49	308.53	2
Root	75.47	0.33	0.22	23.99	305.73	1

Table 1: The results of the dynamic evaluation.

page class	wsconnect(%)	htmlgen(%)	resptime(ms)
Customernat	0.35	99.65	3027.30
Supplier	3.30	96.70	323.60
Suppliernat	4.87	95.13	219.04
Customer	31.12	68.88	34.30
Order	64.35	35.65	16.59
Supplierreg	69.88	30.12	15.27
CustSuppBrand	70.22	29.78	15.20
CustSuppType	70.65	29.35	15.11
Customerreg	71.28	28.72	14.97
CustSuppPart	73.37	26.63	14.55
Suppliers	73.65	26.35	14.49
Part	74.19	25.81	14.39
Customers	74.43	25.57	14.34
CustSupp	77.46	22.54	13.78
PartSupp	78.21	21.79	13.65
Root	96.97	3.03	11.01

Table 2: The results of the static evaluation.

page class	wsconnect(%)	queryexec(%)	xmlgen(%)	htmlgen(%)	resptime(ms)
PartSupp	5.42	93.50	0.18	0.89	4244.72
Supplier	5.64	58.12	16.30	19.93	4080.14
CustSupp	26.78	69.08	0.53	3.62	859.64
CustSuppPart	28.70	65.86	0.67	4.78	802.09
CustSuppBrand	30.22	62.20	0.71	6.88	761.79
CustSuppType	34.74	56.94	0.78	7.54	662.60
Customer	38.26	35.57	8.63	17.54	601.61

Table 3: The results when data is cached in the database.

4.3.3 DB caching

Table 3 shows the results with DB caching only. We have used three cache functions: one for pages of class `Customer` only, one for pages of class `Supplier` and `PartSupp` and one for pages of class `CustSupp`, `CustSuppPart`, `CustSuppType` and `CustSuppBrand`. The selection of these caches is close to optimal and was found after several trials. No cache was used for the other page classes because the number of pages or the locality of reference are low. Thus, the results are not shown for those because they are the same as in Table 1. For pages of class `CustSupp`, `CustSuppPart`, `CustSuppType` and `CustSuppBrand`, the improvement in response time is quite significant (factor 11 or more) The response time for pages of class `Supplier` has been improved by 2.4. However, that for pages of class `PartSupp` has been worsened by a factor 1.5 because the hit ratio does not compensate for the cost of inserting tuples in the cache.

4.3.4 XML caching

Table 4 shows the results with XML caching only. In our configuration, the XML cache takes memory away from the Web server. XML caching essentially improves on query execution time and XML generation time. For page classes with a good hit ratio (`CustSuppType`, `CustSupp`, `CustSuppBrand`, `CustSuppPart`, `Supplier` and `PartSupp`), the response time can be improved by a factor between 3 and 7. For the other page classes, there is either small improvement or slight degradation due to competing memory access by the Web server. Note that our implementation of an XML cache could be improved by using a persistent XML store [32]. Compared to DB caching, the improvement is not as good mainly because of the memory consumption which is high.

4.3.5 HTML caching

Table 5 shows the results with HTML caching only. HTML caching essentially improves on HTML generation time, in addition to the improvements of XML caching. The performance

page class	wsconnect(%)	queryexec(%)	xmlgen(%)	htmlgen(%)	resptime(ms)
Customernat	3.45	0.36	22.18	74.00	7337.22
CustSuppType	8.02	89.27	0.07	2.64	3157.17
CustSupp	8.37	90.57	0.07	0.01.00	3025.36
CustSuppBrand	8.79	89.58	0.07	1.55	2880.49
CustSuppPart	10.01	88.39	0.10	1.50	2528.84
Supplier	13.73	21.65	18.09	46.53	1843.35
Suppliernat	28.44	1.47	16.67	53.42	890.21
Customer	60.27	9.46	5.82	24.45	420.05
PartSupp	68.50	20.83	0.91	9.76	369.61
Order	70.91	8.79	1.61	18.69	357.04
Customers	84.91	0.00	0.00	15.09	298.17
Supplierreg	85.74	3.15	0.54	10.57	295.27
Customerreg	87.51	1.57	0.31	10.61	289.32
Part	88.28	1.26	0.27	10.20	286.80
Suppliers	90.42	0.00	0.13	9.45	279.99
Root	96.26	0.00	0.13	3.61	263.01

Table 4: The results of the XML caching.

improvement is slightly higher than that of XML caching because the HTML cache consumes less memory (all files are stored on disk). But the improvement is still not as good as that of DB caching.

4.3.6 Mixed caching

Table 6 shows the results with a mixed strategy where we combine DB caching, XML caching and HTML caching. For each cache, the decision of what to cache and how was taken according to the observations made in the previous experiments. Thus, we use the DB cache for pages of class `CustSupp`, `CustSuppPart`, `CustSuppType` and `CustSuppBrand`. However, we use the XML cache for `Supplier` and `Customer` pages. Unlike in the XML caching experiment, we decided to cache not all the pages but only the fragments corresponding to the customers of a given *Supplier* and suppliers of a given *Customer*. We assume that these fragments are not frequently updated and worth being cached. The other fragments (i.e. parts of a *Supplier* and orders of a *Customer*) are supposed to be frequently updated and so should be built on demand. Finally, the HTML cache is used for the `Root` page and the pages of class `Customerreg`, `CustomerNat`, `SupplierReg` and `Suppliernat`. There is no caching for the other page classes.

As a result, the performances are much better than with each cache alone. The highest improvements in response time are with the DB cache (factor 11 or more). The improvement of HTML caching for pages of class `Customers` and of XML caching for pages of class

page class	wsconnect(%)	queryexec(%)	xmlgen(%)	htmlgen(%)	resptime(ms)
Customernat	3.33	0.38	23.40	72.89	7517.07
CustSuppType	8.16	91.02	0.07	0.75	3071.73
CustSupp	8.62	90.80	0.06	0.51	2905.92
CustSuppBrand	9.20	89.92	0.07	0.80	2723.02
CustSuppPart	10.96	88.10	0.10	0.84	2286.69
Supplier	13.46	24.24	18.51	43.79	1862.03
Suppliernat	29.01	1.53	16.70	52.76	863.83
PartSupp	46.33	49.66	0.61	3.39	540.91
Customer	62.51	10.66	6.10	20.73	400.92
Order	75.31	11.05	1.75	11.89	332.78
Customers	88.89	0.00	0.12	11.00	281.94
Supplierreg	90.01	3.27	0.43	6.29	278.40
Customerreg	94.03	1.71	0.24	4.01	266.50
Part	95.02	1.38	0.22	3.38	263.73
Suppliers	96.60	0.00	0.07	3.33	259.42
Root	99.14	0.00	0.07	0.79	252.77

Table 5: The results of the HTML cacheing.

`Supplier` is also significant. The results for the uncached pages remain the same as in Table 1.

4.3.7 Concluding remarks

To summarize the results, Figure 3 shows the response times of the different caching strategies for our TPC-D web site. For clarity, only the most expensive page classes are shown. The response time of the other pages is under 450 ms. The figure clearly shows that the mixed strategy is generally close to optimal (static files). Thus, caching at different levels is a good strategy. Then, an important question for the Web site administrator is what to cache (which page classes) and where (in which caches). Based on the results of our experiments, we can make the following observations and hints:

- A database cache yields good performance improvement when the hit ratio is high and the query execution time is important. But it incurs high overhead when the hit ratio is low. This was the case for the pages of class `PartSupp`. Therefore, the database cache should be used only if confident about the high hit ratio. A particular case where the hit ratio is guaranteed to be high is when the same cache content is updated and used by multiple classes of pages.
- When the query execution time or the XML generation time is high and the database cache is not optimal, then it is better to cache XML data. In this case, a hit avoids the

page class	wsconnect(%)	queryexec(%)	xmlgen(%)	htmlgen(%)	resptime(ms)
Customernat	3.21	0.34	22.32	74.13	7569.62
PartSupp	8.84	89.55	0.29	1.32	2747.78
Supplier	11.63	25.95	18.19	44.24	2090.05
Suppliernat	28.44	1.51	15.11	54.95	854.50
CustSupp	29.69	66.01	0.57	3.73	818.44
CustSuppPart	30.77	63.83	0.62	4.78	789.89
CustSuppBrand	32.02	60.83	0.69	6.46	758.88
CustSuppType	39.82	50.23	0.87	9.08	610.32
Order	52.72	20.27	2.89	24.13	460.97
Customer	52.83	6.71	7.57	32.89	460.02
Part	81.62	5.36	1.18	11.84	297.73
Supplierreg	89.08	3.19	0.44	7.29	272.81
Customers	93.22	0.00	0.64	6.14	260.68
Customerreg	93.77	1.77	0.27	4.19	259.16
Suppliers	97.20	0.00	0.18	2.62	250.01
Root	98.98	0.00	0.07	0.95	245.51

Table 6: The results obtained when the three caches are combined.

database connection, and reduces the query execution time and the XML generation time. As an evidence, although the database cache for `Supplier` pages improved performance somewhat, the improvement using the XML cache was much higher.

- If the HTML generation time dominates, it is worth using the HTML cache. The overhead of XML and HTML generation is proportional to document size, and in most cases, both the XML and the HTML caches yield similar performance. For the same performance, the XML cache should be used because an XML document is smaller than the corresponding HTML document (it contains less data and does not include the graphical representation which is done using XSL style sheet). Furthermore, the XML cache allows to cache document fragments. This is what we did in the mixed caching experiment for the pages of class `Supplier` and `Customer`). This facilitates update propagation and therefore garanties data freshness.

- When using the HTML cache, it would be better to chop long pages into smaller ones. In this case, navigation facilities should be added in the resulting pages (for instance, a next page icon). An alternative would be to use pages with indexes (shortcuts) so the user could choose the pages to see.

- Static files should be used for page classes with a small number of instances when the access frequency is high.

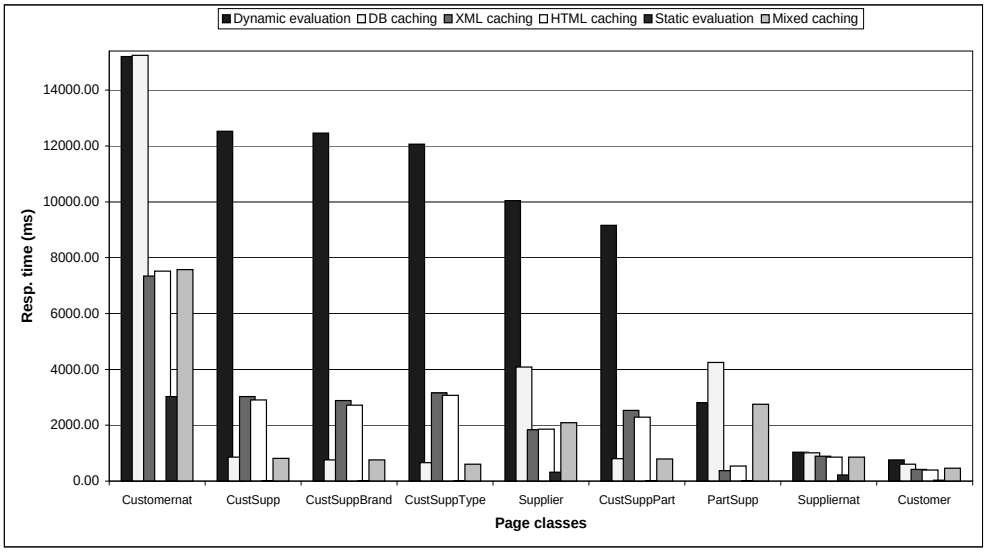

Figure 3: Comparison of different caching strategies.

5 Conclusion

In a data-intensive Web site, returning a Web page may require costly interaction with the database system (connection and querying) to dynamically extract its content. The database interaction cost adds up to the non-negligible base cost of Web page delivery, thereby increasing much the response time. Although useful, techniques for proxy and Web server caches do not help reducing the Web latency in this case since they work at the level of HTML pages.

In this paper, we have addressed the performance problem of accessing dynamic Web pages in data-intensive Web sites. This work has been done in the context of the Weave Web site management system developped at Inria.

Our approach relies on the declarative specification of the Web site through a logical model and the automatic generation of the run-time policy, in particular caching. The logical model of the Web site is based on the XML graph data model. A site schema then represents an XML view definition over a relational database. Thus, we can manage the data of the Web site at three levels: database, XML fragments, HTML files.

We propose a customizable cache system architecture and its implementation. Our solution enables data caching at the various levels of data elaboration within the Web site: database data, XML fragments or HTML files. In addition, Weave comes along with two

declarative languages: one for specifying the Web site's content and one for specifying the customized cache management within the site. Given a site graph, cache management may be specified at a fine grain by attaching caching actions to each site class. Our solution is illustrated using a Web site derived from the TPC/D benchmark database.

We have also proposed a test platform called WeaveBench to assess the performance of various caching strategies for data-intensive Web sites. We have used Oracle v8 for the database, Apache v3 for the Web server, Apache Jserv for the global manager, IBM XML4J for the XML generator and Lotus XSL for the HTML generator. Based on experiments, we have assessed the performance of various caching strategies: dynamic pages (worst case), static files (best case), DB caching, XML caching, HTML caching, and mixed caching (combining DB, XML and HTML caching). The results clearly show that a mixed strategy is generally close to optimal (static files). Finally, we have derived guidelines for specifying cache management based on the Web site specifications.

References

[1] Manuel Afonso, Alexandre Santos, and Vasco Freitas. QoS in Web caching. In *Proc. of the Third International Web Caching Workshop*, Manchester, England, June 15-17 1998. Availble at http://wwwcache.ja.net/events/workshop/papers.html.

[2] IBM Alphaworks. LotusXSL. Available at http://www.alphaworks.ibm.com/tech/LotusXSL.

[3] IBM Alphaworks. XML4J: XML for Java parser. Available at http://www.alphaworks.ibm.com/tech/XML4J.

[4] G. Arocena and A. Mendelzon. WebOQL: Restructuring documents, database and Webs. In *Proc. of Int. Conf. on Data Engineering (ICDE)*, pages 24–33, Orlando, Florida, 1998.

[5] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. To weave the Web. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, 1997.

[6] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. Design and maintenance of data-intensive Web sites. In *Proc. of the Conf. on Extending Database Technology (EDBT)*, pages 436–450, 1998.

[7] M-L. Schneider B. Chidlovskii, C. Roncancio. Semantic cache mechanism for heterogeneous Web querying. In *Proc. of the Int. World Wide Web Conf.*, Toronto, Canada, May 11-14 1999.

[8] Brian Baker. Doing business with the Web: The Informix/Illustra approach. In *Proc. of the Conf. on Extending Database Technology (EDBT)*, pages 364–369, Avignon, France, 1996.

[9] G. Banga, F. Douglis, and M. Rabinovich. Optimistic deltas for WWW latency reduction. In *Proc. of USENIX Technical Conf.*, 1997.

[10] J. F. Barnes and R. Pandey. Providing dynamic and customizable caching policies. In *Proc. of USENIX Symp. on Internet Technologies and Systems*, 1999.

[11] J. Fritz Barnes and Raju Pandey. CacheL: Language support for customizable caching policies. In *Proc. of the Fourth International Web Caching Workshop*, San Diego, California, USA, March 1999. Availble at http://arthur.cs.ucdavis.edu/ barnes/Papers.html.

[12] J. Fritz Barnes and Raju Pandey. Providing dynamic and customizable caching policies. In *USENIX Second Symp. on Internet Technologies and Systems*, Boulder, Colorado, USA, October 1999. Availble at http://arthur.cs.ucdavis.edu/ barnes/Papers.html.

[13] A. Bestavros. Using speculation to reduce server load and service time on the WWW. In *Proc. of the 4th International Conf. on Information and Knowledge Management*, 1995. Available at http://www.cs.bu.edu/techReports/95-006-speculative-service.ps.Z.

[14] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. Proc. of INFOCOM, 1999.

[15] P. Cao, J. Zhang, and K. Beach. Active cache: Caching dynamic contents (objects) on the Web. In *Proc. of Middleware'98*, 1998.

[16] Pei Cao. On the implications of Zipf's law for Web caching. In *Proc. of the Third International Web Caching Workshop*, Manchester, England, June 15-17 1998. Availble at http://wwwcache.ja.net/events/workshop/papers.html.

[17] Pei Cao, Jin Zhang, and Kevin Beach. Active cache: Caching dynamic contents on the web. In *Proc. of IFIP International Conf. on Distributed Systems Platforms and Open Distributed Processing (Middleware)*, pages 373–388, 1998.

[18] S. J. Caughey, D. B. Ingham, and M. C. Little. Flexible open caching for the Web. In *Proc. of the Int. World Wide Web Conf.*, Stanford, California, USA, April 7-11 1997. Available at http://w3objects.ncl.ac.uk/pubs/focw/ps/paper.ps.

[19] J. Challenger, A. Iyengar, and P. Dantzig. A scalable system for consistently caching dynamic Web data. In *Proc. of IEEE INFOCOM'99*, March 1997.

[20] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical Internet object cache. In *In Proc. of the USENIX Technical Conf.*, Jan 1996.

[21] A. Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical Internet object cache. In *Proc. of USENIX Technical Conf.*, pages 153–163, January 1996.

[22] Josephine Cheng and Susan Malaika. *Web Gateway Tools. Connecting IBM and Lotus Applications to the Web.* Wiley Computer Publishing, 1997.

[23] Sophie Cluet, Claude Delobel, Jerome Simeon, and Katarzyna Smaga. Your mediators need data conversion. In *Proc. of ACM SIGMOD Conf. on Management of Data*, Seattle, WA, 1998.

[24] Microsoft Corporation. ISAPI Overview. Available at http://www.microsoft.com/msdn/sdk/platforms/doc/sdk/internet/src/isapimrg.html.

[25] Netscape Communication Corporation. The server-application function and Netscape Server API. Available at http://www.netscape.com/newsref/std/server_api.html.

[26] Transaction Processing Performance Council. The new TPC Web ecommerce benchmark. Available at http://www.tpc.org/.

[27] M. Crovella and P. Barford. The network effects of prefetching. In *Proc. IEEE INFOCOM*, April 1998. Available at ftp://cs-ftp.bu.edu/techReports/97-002-prefetcheff.ps.Z.

[28] Carlos Cunha and Carlos F. B. Jaccoud. Determining WWW user's next access and its application to pre-fetching. In *Proc. of ISCC: The Second IEEE Symp. on Computers and Communications*, July 1997. Available at http://www.cs.bu.edu/students/alumni/carro/Home.html.

[29] B. Davison and H. Hirsch. Predicting sequences of user actions. In *Notes of the AAAI/ICML 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis*, 1998. Available at http://www.cs.rutgers.edu/ davison/pubs/aaai98ws.ps.

[30] B. D. Davison. Adaptive Web prefetching. In *Position paper in Proc. of the 2nd Workshop on Adaptive Systems and User Modeling on the WWW, a part of the Eighth International World Wide Web Conf.*, Toronto, May 11 1999.

[31] Hasan Davulcu, Juliana Freire, Michael Kifer, and I. V. Ramakrishnan. A layered architecture for querying dynamic Web content. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 491–502, Philadephia, Pennsylvania, USA, 1999.

[32] Object Design. eXcelon: an XML application development environment for integrating data. Available at http://www.odi.com/excelon/main.htm.

[33] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. XMLQL: a query language for XML. In *Proc. of the Int. World Wide Web Conf.*, Toronto,CA, 1999.

[34] F. Douglis, A. Haro, and M. Rabinovich. HPP: HTML macro-preprocessing to support dynamic document caching. In *Proc. of USITS'97 – USENIX Symp. on Internetworking Technologies and Systems*, December 1997.

[35] Li Fan, Quinn Jacobson, Pei Cao, and Wei Lin. Web prefetching between low-bandwidth clients and proxies: Potential and performance. SIGMETRICS, 1999.

[36] Mary Fernandez, Daniela Florescu, Jaewoo Kang, Alon Levy, and Dan Suciu. Catching the boat with Strudel: Experiences with a Web-site management system. In *Proc. of ACM SIGMOD Conf. on Management of Data*, Seattle, WA, 1998.

[37] Daniela Florescu, Alon Levy, and Alberto Mendelzon. Database techniques for the World Wide Web: A survey. *SIGMOD Record*, 27(3):59–74, September 1998.

[38] Daniela Florescu, Alon Levy, Dan Suciu, and Khaled Yagoub. Optimization of run time management of data intensive Web sites. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, Edinburgh, UK, 1999.

[39] Piero Fraternali. Tools and approches for developing data-intensive Web applications: a survey. *ACM Computing Surveys*, 1999.

[40] Piero Fraternali and Paolo Paolini. A conceptual model and a tool environment for developing more scalable, dynamic, and customizable Web applications. In *Proc. of the Conf. on Extending Database Technology (EDBT)*, pages 421–435, 1998.

[41] Franca Garzotto, Paolo Paolini, and Daniel Schwabe. HDM - a model-based approach to hypertext application design. *ACM Transactions on Information Systems (TOIS)*, 11(1):1–26, 1993.

[42] S. D. Gribble and E. A. Brewer. System design issues for Internet middleware services: Deduction from a large client trace. In *Proc. of USITS'97 – USENIX Symp. on Internet Technologies and Systems*, pages 207–218, 1997.

[43] Martin Gruber, Kennan Rossi, and Charles Prael. Introduction to the Oracle WebServer. Available at http://www.informatik.fh-trier.de:8080/ows-adoc/content.htm, 1996.

[44] M. Gwyer. Oracle Designer/2000 Web Server Generator. Oracle techincal overview. Available at http://ntsolutions.oracle.com/solution/datamart/pdf/2kweb.pdf, February 1996.

[45] Stathes P. Hadjiefthymiades and Drakoulis I. Martakos. Improving the performance of CGI compliant database gateways. In *Proc. of the Int. World Wide Web Conf.*, Santa Clara, California USA, April 7-11 1997.

[46] IBM. Net.data programming guide. Available at http://www.software.ibm.com/data/net.data, 1997.

[47] Informix. Informix Web Integration Option for Informix Dynamic Server. Informix techincal overview, 1998. Available at http://www.informix.com/informix/whitepapers/.

[48] D. Ingham, S. Caughey, and M. Little. Supporting highly manageable Web services. In *Proc. of the Int. World Wide Web Conf.*, Stanford, California, USA, April 7-11 1997. Availble at http://www6.nttlabs.com/HyperNews/get/PAPER27.html.

[49] A. Iyengar and J. Challenger. Data update propagation: A method for determining how changes to underlying data affect cached objects on the Web. Technical report, IBM Research Division, Yorktown Heights, NY, February 1998.

[50] A. Iyengar, M. S. Squillante, and L. Zhang. Analysis and characterization of large-scale Web server access patterns and performance. Technical report, IBM Research Division, 1998.

[51] Arun Iyengar and Jim Challenger. Improving Web server performance by caching dynamic data. In *Proc. USENIX Symp. on Internet Technologies and Systems*, December 1997. http://www.research.ibm.com/people/i/iyengar/usits97.ps.

[52] Arun Iyengar, Jim Challenger, and Paul Dantzig. A scalable and highly available system for serving dynamic data at frequently accessed Web sites. In *Proc. of ACM/IEEE Supercomputing*, Orlando, Florida, November 1998.

[53] Arun Iyengar, Jim Challenger, and Paul Dantzig. A scalable system for consistently caching dynamic Web data. In *Proc. of IEEE INFOCOM*, New York, New York, March 1999.

[54] Arun Iyengar, Ed MacNair, and Thao Nguyen. An analysis of Web server performance. In *Proc. of the IEEE Global Telecommunications Conf. (GLOBECOM)*, Phoeniz, AZ, November 1997.

[55] Arun K. Iyengar, Edward A. MacNair, Mark S. Squillante, and Li Zhang. A general methodology for characterizing access patterns and analyzing Web server performance. In *Proc. of MASCOTS*, 1998.

[56] Quinn Jacobson and Pei Cao. Potential and limits of Web prefetching between low-bandwidth clients and proxies. In *Proc. of the Third International Web Caching Workshop*, Manchester, England, June 15-17 1998. Availble at http://wwwcache.ja.net/events/workshop/papers.html.

[57] Balachander Krishnamurthy and Craig E. Wills. Proxy cache coherency and replacement—towards a more complete picture. In *Proc. of the 19th IEEE International Conf. on Distributed Computing Systems*, Austin,TX, June 1999. Available at http://www.cs.wpi.edu/ cew/papers/icdcs99.ps.gz.

[58] T. M. Kroeger, D. D. E. Long, and J. C. Mogul. Exploring the bounds of Web latency reduction from caching and prefetching. In *Proc. of the Usenix Symp. on Internet Technologies and Systems*, 1997.

[59] Alexandros Labrinidis and Nick Roussopoulos. On the materialization of WebViews. In *International Workshop on Web and Databases (WebDB), in conjunction with SIGMOD'99*, Philadelphia, USA, June 1999.

[60] Chengjie Liu and Pei Cao. Maintaining strong cache consistency in the World-Wide Web. In *Proc. of the 17th IEEE International Conf. on Distributed Computing Systems*, May 1997. Available at http://www.cs.wisc.edu/ cao/icache/icache.ps.

[61] Evangelos P. Markatos and Catherine E. Chronaki. A top-10 approach to prefetching on the Web. Technical report, ICS-FORTH, Heraklion, Crete, Greece, August 1996. Available at http://www.ics.forth.gr/proj/archvlsi /www.html.

[62] Ian Marshall and Chris Roadknight. Linking cache performance to user behaviour. In *Proc. of the Third International Web Caching Workshop*, Manchester, England, June 15-17 1998. Availble at http://wwwcache.ja.net/events/workshop/papers.html.

[63] G. Mecca, P. Merialdo, P. Atzeni, and V. Crescenzi. The Araneus guide to Web-site development. Technical report, Araneus Project Working Report, AWR-1-99, Dipartimento di Informatica e Automazione, Universita' di Roma Tre, 1999.

[64] Giansalvatore Mecca, Paolo Atzeni, Alessandro Masci, Paolo Merialdo, and Giuseppe Sindoni. The Araneus Web-base management system. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 544–546, Seattle, June 1998.

[65] Giansalvatore Mecca, Alberto O. Mendelzon, and Paolo Merialdo. Efficient queries over Web views. In *Proc. of the Conf. on Extending Database Technology (EDBT)*, pages 72–86, Valencia, Spain, 1998.

[66] Microsoft. ASP technology feature overview. Available at http://msdn.microsoft.com/workshop/server/asp/aspfeat.asp.

[67] Mindcraft. WebStone benchmark information. Available at http://www.mindcraft.com/benchmarks/webstone/.

[68] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. In *Proc. of ACM SIGCOMM'97*, 1997.

[69] Tam Nguyen and V.Srinivasan. Accessing relational databases from the World Wide Web. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 529–540, Montreal, Canada, 1996.

[70] Oracle. Oracle WebServer user's guide. Available at http://www.hmi.de/oracle/doc/, 1996.

[71] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve World Wide Web latency. *ACM SIGCOMM Computer Communicati Review*, July 1996. Available at http://www.cs.berkeley.edu/ padmanab/papers/ccr-july96.ps.gz.

[72] T. Palpanas and Alberto Mendelzon. Web prefetching using partial match prediction. Technical Report CSRG-376. Departement of Computer Science, University of Toronto, 1998.

[73] P. Paolini and P. Fraternali. A conceptual model and a tool environment for developing more scalable, dynamic, and customizable Web applications. In *Proc. of the Conf. on Extending Database Technology (EDBT)*, 1998.

[74] M. Perkowitz and O. Etzioni. Adaptive Web sites: Automatically learning from user access patterns. In *Proc. of the Int. World Wide Web Conf.*, 1997.

[75] Mike Perkowitz and Oren Etzioni. Adaptive Web sites: an AI challenge. In *Proc. of the 15th International Joint Conf. on Artificial Intelligence*, pages 16–23, Nagoya, Japan, 1997.

[76] James Pitkow and Peter Pirolli. Mining longest repeating subsequences to predict World Wide Web surfing. In *USENIX-LRS*, 1999.

[77] Java Apache Project. Apache JServ: a pure Java Servlet engine. Available at http://java.apache.org/.

[78] B. Proll, W. Retschitzegger, and R. R. Wagner. TIScover - a tourism information system based on Extranet and Intranet technology. In *Proc. of the Fourth Americas Conf. on Information Systems (AIS)*, Baltimore, Maryland, August 14-16 1998.

[79] Mike Reddy and Graham P. Fletcher. Intelligent Web caching using document life histories: A comparison with existing cache management techniques. In *Proc. of the Third International Web Caching Workshop*, Manchester, England, June 15-17 1998. Availble at http://wwwcache.ja.net/events/workshop/papers.html.

[80] Ceri S., Fraternali P., and Paraboschi S. Data-driven, one-to-one Web site generation for data-intensive applications. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, Edinburgh - Scotland - UK, September, 7-10 1999.

[81] Jerome Simeon and Sophie Cluet. Using YAT to build a Web server. In *International Workshop on the Web and Databases (WebDB), in conjunction with EDBT'98*, Valencia, Spain, March 27-28 1998.

[82] Giuseppe Sindoni. Incremental maintenance of hypertext views. In *International Workshop on the Web and Databases (WebDB), in conjunction with EDBT'98*, Valencia, Spain, March 27-28 1998.

[83] Ben Smith, Anurag Acharya, and Huican Zhu Tao Yang. Caching equivalent and partial results for dynamic Web content. In *Proc. of USENIX Symp. on Internet Technologies and Systems*, Boulder, Colorado, USA, October 11-14 1999. Available at http://www.cs.ucsb.edu/research/swala/.

[84] The Standard Performance Evaluation Corporation (SPEC). Specweb99 benchmark. Available at http://www.specbench.org/osg/web99/.

[85] T. Stabin and C. E. Glasson. First impression: 7 commercial log processing tools slice & dice logs your way. Available at http://www.netscapeworld.com/netscapeworld/nw08-1997/nw-08-loganalysis.html, 1997.

[86] Sybase. Sybase Web.sql programmer's guide. Available at http://sundoc.bibliothek.uni-halle.de/websql.dir/docs/, 1997.

[87] Motomichi Toyama and T. Nagafuji. Dynamic and structured presentation of database contents on the Web. In *Proc. of the Conf. on Extending Database Technology (EDBT)*, pages 451–465, 1998.

[88] TPCD. Benchmark. Available at http:// www.tpc.org.

[89] Various. Information on CGI. Available at http://www.w3.org/pub/WWW/CGI.

[90] Bert Williams. Transparent Web caching solutions. In *Proc. of the Third International Web Caching Workshop*, Manchester, England, June 15-17 1998. Availble at http://wwwcache.ja.net/events/workshop/papers.html.

[91] Craig E. Wills and Mikhail Mikhailov. Towards a better understanding of web resources and server responses for improved caching. In *Proc. of the Int. World Wide Web Conf.*, Toronto, Ontario, Canada, May 1999. Available at http://www.cs.wpi.edu/ cew/papers/www8.ps.gz.

[92] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. M. Levy. Organization-based analysis of Web-object sharing and caching. In *Proc. of USITS'99 – USENIX Symp. on Internetworking Technologies and Systems*, 1999.

[93] Roland P. Wooster and Marc Abrams. Proxy caching that estimates page load delays. In *Proc. of the Int. World Wide Web Conf.*, Santa Clara, California USA, April 7-11 1997.

[94] XML. Extensible markup language. Available at http://www.w3.org/XML/.

[95] XSL. Extensible stylesheet language. Available at http://www.w3.org/Style/XSL/.

[96] Lixia Zhang, Scott Michel, Khoi Nguyen, and Adam Rosenstein. Adaptive Web caching: Towards a new global caching architecture. In *Proc. of the Third International Web Caching Workshop*, Manchester, England, June 15-17 1998. Availble at http://wwwcache.ja.net/events/workshop/papers.html.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399